

実践 パケット解析 第2版

Wiresharkを使ったトラブルシューティング

Chris Sanders 著

高橋 基信 監訳
宮本 久仁男
岡 真由美 訳

O'REILLY®
オライリー・ジャパン

本書で使用するシステム名、製品名は、それぞれ各社の商標、または登録商標です。
なお、本文中では™、®、©マークは省略している場合もあります。

PRACTICAL PACKET ANALYSIS

2ND EDITION

**Using Wireshark to Solve
Real-World Network
Problems**

by Chris Sanders



**no starch
press**

San Francisco

Copyright © 2011 by Chris Sanders.

Title of English-language original:

PRACTICAL PACKET ANALYSIS, 2ND EDITION, ISBN978-1-59327-266-1, published by No Starch Press.

Japanese-language edition copyright © 2012 by O'Reilly Japan, Inc. All rights reserved.

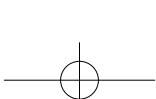
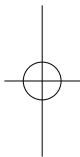
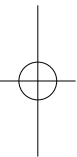
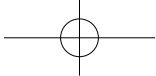
本書は、株式会社オライリー・ジャパンがNo Starch Press, Inc.の許諾に基づき翻訳したものです。日本語版についての権利は、株式会社オライリー・ジャパンが保有します。

日本語版の内容について、株式会社オライリー・ジャパンは最大限の努力をもって正確を期していますが、本書の内容に基づく運用結果について責任を負いかねますので、ご了承ください。

本書、私の人生、そして私のなした業のすべては、神から与えられ、そして受けた信仰の実りにほかなりません。本書を私の神、両親、そして絶えず私に信仰をあかししてくださった方々の一人ひとりに捧げます。

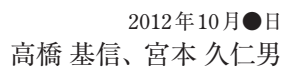
「まことに、あなたがたに告げます。もし、からし種ほどの信仰があったら、この山に、『ここからあそこに移れ』と言えば移るのです。どんなことでも、あなたがたにできないことはありません。」

—— マタイの福音書17:20（聖書 新改訳 © 1970, 1978, 2003 新日本聖書刊行会）



[illegible]

A 5x20 grid of squares, where every square is filled black, representing 100% completion.



賞賛の声

「あらゆるネットワーク管理者に必携の書籍」

—— LINUX PRO MAGAZINE

「素晴らしく、使いやすく、わかりやすい」

—— ARSGEEK.COM

「パケット解析の基礎を完璧に理解する必要があるなら、この本は最適な出発点だ」

—— STATEOFSECURITY.COM

「非常に有益であり、『実践』というタイトルがまさにふさわしい。パケット解析に際して知っておくべきことを読者に示す書籍として秀逸であり、Wiresharkで行う作業を現場に即した例で示している」

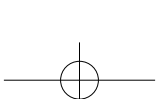
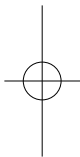
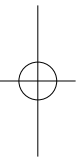
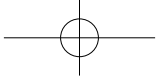
—— LINUXSECURITY.COM

「知らないうちに互いに通信しているホストはないだろうか。自分のマシンは知らないホストと通信していないだろうか。こうした質問に確実な答えを出すうえでは、パケット解析が必須である。Wiresharkはこれに最適なツールのひとつであり、本書はこのツールの学習に最適なもののひとつである」

—— FREE SOFTWARE MAGAZINE

「初心者から中級者に最適」

—— DAEMON NEWS



まえがき

『実践パケット解析 第2版』は、初版発行から約4年後の、2009年後半から2011年中盤に1年半かけて執筆されました。内容はほぼ一新され、キャプチャファイルやシナリオもほぼ完全に新しいものとなっています。初版を気に入っていただけたなら、本書も好きになっていただけたと思います。本書は初版と同じ形式で執筆され、シンプルでわかりやすい説明がつけられています。初版がお気に召さなくても、本書には新しいシナリオがあり、内容も拡充されているので、気に入っていただけるものと思います。

なぜ本書なのか？

なぜ、パケット解析に関するほかの本ではなく本書を買うべきなのか、疑問に思うかもしれません。答えは『実践 パケット解析』（原著名は『Practical Packet Analysis』）というタイトルにあります。現場での経験に勝るものはありません。書籍によってこの経験に達するためには、現場に即したシナリオを用いたパケット解析の実例が一番です。

本書の前半では、パケット解析とWiresharkを理解するための前提となる知識を習得します。後半は、日々のネットワーク管理で遭遇するであろう実例の解説にあてられています。

ネットワーク技術者、ネットワーク管理者、CIO、パソコン技術者、ネットワークセキュリティアナリストの誰もが、本書で解説するパケット解析の技術を理解し、実践することで、多くの有益な情報を得ることができます。

本書のコンセプトとアプローチ

筆者はざっくばらんな人間なので、コンセプトを語るときもやっぱりざっくばらんに語ることになるでしょう。それは本書においても例外ではありません。技術的なコンセプトを語るときは、どうしても堅苦しい技術用語が多くなりますが、それでもできる限りざっくばらんな言い回しを心がけたつもりです。一方で、定義は簡潔かつ明確に、過不足ないようにしたつもりです。要するに、私は偉大なるケンタッキー州出身なので、難しい言葉の使用は最小限に抑えましたということです（本書全体に見ら

れるド田舎風の言い回しはお許しください)。

本当にパケット解析を学びたいと思っているのなら、本書の前半で紹介されているコンセプトを理解する必要があります。これは後半部分を理解するために不可欠だからです。本書の後半部分は非常に実践的になっています。業務で発生するトラブルとまったく同じシナリオはないかもしれませんが、トラブルが発生したときに、本書で習得したコンセプトを適用することがきっとできるはずです。

以下は本書の各章の内容の簡単な説明です。

1章 パケット解析とネットワークの基礎

パケット解析とは何で、どのように動作し、どうやって行うのでしょうか。この章では、ネットワーク通信とパケット解析の基本を解説します。

2章 ケーブルに潜入する

この章では、ネットワーク上にパケットキャプチャツールを配置する方法をいくつか紹介します。

3章 Wireshark概要

Wiresharkの入手方法、使い方、機能、優位性をはじめとする、Wiresharkの基本について解説します。

4章 Wiresharkでのパケットキャプチャのテクニック

Wiresharkが起動したら、次はキャプチャしたパケットの扱い方について知りたくなるところです。この章では、基本的な扱い方を解説します。

5章 Wiresharkの高度な機能

基本を身につけたら、次は応用です。この章ではWiresharkの高度な機能を詳説し、普段はあまり目にしない機能を紹介します。

6章 知っておきたい下位層プロトコル

この章では、最低限知っておきたい下位層の通信プロトコル——TCP、UDP、IPなど——が、パケットレベルでどのように見えるかを紹介します。各プロトコルで起こるトラブルを理解するうえで、まずは、どのように動作するかを理解しましょう。

7章 知っておきたい上位層プロトコル

引き続きこの章では、最低限知っておきたい3つの上位層のプロトコル——HTTP、DNS、DHCP——が、パケットレベルでどのように見えるかを紹介します。

8章 現実世界のシナリオの第一歩

この章は、基本的なトラフィックの解析と、初歩的な現場に即したシナリオから構成されます。各シナリオは、発生したトラブル、解析方法、対処策の順に記載され、読み進めやすい形式になっています。この章のシナリオは、数台のコンピュータしか登場しない基本的なもので、解析の手間も

それほどではなく、パケット解析を始めるのにちょうどよい難易度になっています。

9章 ネットワークの遅延と戦う

ネットワークのトラブルで一番多いのは、ネットワークの遅延です。この章では、ネットワークの遅延に関するトラブルの解決に焦点を当てます。

10章 セキュリティとパケット解析

IT分野において、ネットワークセキュリティはもっとも関心の高い話題でしょう。10章では、パケット解析によってセキュリティ関連のトラブルの解決に結びつくようなシナリオをいくつか紹介します。

11章 無線LANのパケット解析

この章は、無線LANのパケット解析の入門編です。有線LANと無線LANのパケット解析の違いについて、無線LANのトラフィックの実例を通じて説明します。

付録A USBポートの通信キャプチャ

付録Aは監訳者による日本語版オリジナルの記事です。Wiresharkを用いてUSBデバイスとホストの間で行われてる通信をキャプチャする方法について解説します。

付録B pcap-ng形式 ↔ pcap形式のデータ変換

付録Bは監訳者による日本語版オリジナルの記事です。Wireshark 1.8以降で標準のデータ形式となったpcap-ng形式で保存されたデータを以前から使われてきたデータ形式であるpcap形式に変換する方法、逆にpcap形式で保存されたデータをpcap-ng形式に変換する方法を紹介します。

付録C 推薦文献

付録Cとして、本書で学んだパケット解析のテクニックを使っていくうえで、有用なツールやWebサイトを集めました。

本書の使い方

本書は2つの使い方を想定しています。

- 1章ずつ順に読んでいくことで、パケット解析の理解を深めるための勉強用のテキストとして使う方法。現場に即したシナリオが掲載されている後半部分は特に重要になります。
- 本書をリファレンスとして使う方法。頻繁に使用するわけではないWiresharkの機能をいちいち覚えておく必要はありません。そういった機能を使うときのリファレンスとして本書が本棚に置いてあれば、なにかと役に立つと思います。また業務でパケット解析を行う際に便利なリファレンスとなるような図表、略図、手法も紹介しています。

サンプルのキャプチャファイルについて

本書で使われているキャプチャファイルはすべて、No Starch Pressの本書のページ<http://www.nostarch.com/packet2.htm>から入手できます。本書を最大限活用するために、これらのファイルをダウンロードしたうえで、実際にファイルを使って、サンプルの流れを追ってみることを強くお勧めします。

Rural Technology 基金

『実践パケット解析』から生まれた最大の成果について触れないわけにはいきません。本書の初版が発行されて間もなく、私は最大の夢のひとつの頂点となる501(c)(3)非営利団体（日本のNPO法人にあたる）を創設しました。

地方の学生の場合、たとえ素晴らしい成績であったとしても、都市や郊外の学生と比べると技術に触れる機会が格段に少なくなります。Rural Technology Fund (RTF) は、地方と都会のコミュニティ間のデジタル格差を埋める目的で、2008年に設立されました。地方における奨学金プログラム、コミュニティ連携、その他さまざまな宣伝活動や技術の推進活動などを行っています。

奨学金は、地方在住で、コンピュータ技術に情熱を持ち、この分野でさらに教育を受けたいと考える学生たちを対象としています。この奨学金資金とするため、原書の著者印税の100%をRural Technology 基金に全額寄付することにしました。Rural Technology 基金についてもっと知りたい、あるいはどのように貢献できるかを知りたいければ、<http://www.ruraltechfund.org/>を参照してください。

筆者の連絡先

私が執筆した書籍についての読者からのフィードバックをいつも楽しみにしています。質問やコメントを送りたい、脅迫したい、あるいは結婚のプロポーズをしたい、なんでもかまいません。筆者と連絡を取りたい場合は直接chris@chrissanders.orgへどうぞ。ブログ<http://www.chrissanders.org/>も頻繁に更新しています。Twitterで[@chrissanders88](https://twitter.com/chrissanders88)をフォローいただくこともできます。

本書の表記

本書は、以下の表記を使用します。

太字 (Bold)

新しい用語、強調やキーワードフレーズは太字で表記します。

等幅 (Constant Width)

プログラムのコード、コマンド、配列、要素、文、オプション、スイッチ、変数、属性、キー、

関数、型、クラス、名前空間、メソッド、モジュール、プロパティ、パラメータ、値、オブジェクト、イベント、イベントハンドラ、XMLタグ、HTMLタグ、マクロ、ファイルの内容、コマンドからの出力は、等幅で表記します。

等幅太字 (Constant Width Bold)

ユーザーが入力するコマンドやテキストは、等幅太字で表記する。重要な部分を強調するために使う場合もあります。

等幅イタリック (Constant Width Italic)

ユーザーが入力する値で置き換えられる部分は、等幅斜体で表記します。



このアイコンは、ヒントや提案や一般的注意事項を表します。



このアイコンは、警告や要注意事項を表します。

意見と質問

本書（日本語翻訳版）の内容については、最大限の努力をもって検証および確認していますが、誤りや不正確な点、誤解や混乱を招くような表現、単純な誤植に気づかれることもあるでしょう。本書を読んで気づいたことは、今後の版で改善できるように知らせてください。将来の改訂に関する提案なども歓迎します。

株式会社オライリー・ジャパン

〒160-0002 東京都新宿区坂町26番地27 インテリジェントプラザビル1F

電話 03-3356-5227

FAX 03-3356-5261

電子メール japan@oreilly.co.jp

本書に関する技術的な質問や意見については、次のあて先に電子メール（英文）を送ってください。

info@nostarch.com

本書に関連するファイルは本書のWebページからダウンロードできます。

<http://www.oreilly.co.jp/books/9784873115696/>

<http://oreilly.com/catalog/9781593272661/> (原書)

<http://nostarch.com/packet2.htm> (原書)

監訳者のWebページには、正誤表や追加情報が掲載されている。以下のアドレスでアクセスできます。

<http://★★★★★★★★★★★★/>

オライリーに関するその他の情報については、次のオライリーのWebサイトを参照してください。

<http://www.oreilly.co.jp/>

<http://www.oreilly.com/>

謝辞

本書は非常に多くの人々の、直接的および間接的な貢献によって完成しました。

何よりもまず、賞賛のすべては神のものです。書籍を執筆していると、やる気になることもあれば、落ち込んだりするときもあります。ストレスを感じると、神が気持ちを落ち着かせてくれます。いらいらすると、平安を与えてくれます。混乱すると、解決方法を示してくれます。疲れると、休息を与えてくれます。うぬぼれると、冷静にさせてくれます。本書、私の経歴、そして存在そのものが、神と神の御子イエス・キリストを抜きにはありえません。

お父さん。さまざまなことから励ましを受けてきましたが、私を誇りに思うというその言葉ほど、私を幸せにしてくれたものはありません。それを伝えてくれたことに、感謝の言葉もありません。

お母さん。本書の第2版は、お母さんがこの世を去ってちょうど10年目となる日の直前に出版されました。お母さんが私を見守っていて、誇りに思ってくれているのはわかっています。これからもっと誇りに思ってもらえるように頑張ります。

DebiおばさんとRandyおじさん。お二人は、はじめから、私の最大の支援者です。私の家族は大人数ではないですが、家族のみんな、特におじさんとおばさんのことを大切に思っています。なかなか会えませんが、第二の両親のようにしてくれることに、感謝の言葉もありません。

Tina Nance。以前のように会えなくなりましたが、いつも二人目のお母さんのように思っています。あなたの献身と信頼がなければ、今していることは成し遂げられなかったと思います。

Jason Smith。誰よりも頻繁に愚痴を聞いて、気持ちを落ち着かせてくれました。素晴らしい友人として、そして同僚として、さまざまなプロジェクトで助言を与えてくれて、そして一度は半年近くも車庫を使わせてくれてありがとう。

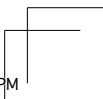
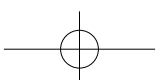
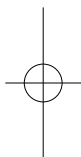
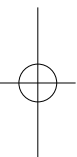
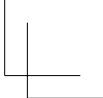
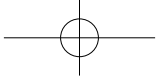
今も昔も、素晴らしい同僚に囲まれている人は、素晴らしい人になれるといつも信じてきました。私はこの仕事を通じて、優秀で素晴らしい人々と一緒に仕事ができるという幸運に恵まれています。皆さんは家族です。

Mike Poor。あなたは疑いなく、私のパケット解析における目標です。その仕事とやり方は、私にひらめきを与え、行動を助けてくれています。

Tyler Reguly。本書の技術的な編集をしてくれたことに本当に感謝します。楽しい作業ではなかったと思いますが、間違いなく必要な作業でした。心より感謝しています。

そしてGerald CombsとWireshark開発チームにも感謝しています。Wiresharkを素晴らしい解析ツールにしたのは、Geraldとはじめとする大勢の開発者の貢献のおかげです。その努力がなかったら、本書は存在していません。仮に存在したとしても、tcpdumpに基づいた、まったく面白くないものだったでしょう。

BillとNo Starch Pressのスタッフは、このケンタッキー出身の若僧に、一度ばかりか二度もチャンスを与えてくれました。チャンスをくれて、辛抱してくれて、そして私の夢の実現を支援してくれて、本当にありがとうございました。



目次

監訳者まえがき	vii
賞賛の声	ix
まえがき	xiii
1章 パケット解析とネットワークの基礎	1
1.1 パケット解析とパケットキャプチャツール	1
1.1.1 パケットキャプチャツールの評価	2
1.1.2 パケットキャプチャツールの仕組み	3
1.2 コンピュータはどのように通信するのか	4
1.2.1 プロトコル	4
1.2.2 OSI参照モデル	5
1.2.3 データのカプセル化	8
1.2.4 ネットワークハードウェア	10
1.3 トラフィックの分類	15
1.3.1 ブロードキャスト	15
1.3.2 マルチキャスト	16
1.3.3 ユニキャスト	17
1.4 まとめ	17
2章 ケーブルに潜入する	19
2.1 プロミスキューモードの使用	20
2.2 ハブで構成されたネットワークでのキャプチャ	21
2.3 スイッチで構成されたネットワークでのキャプチャ	22
2.3.1 ポートミラーリング	23
2.3.2 ハブの使用	24
2.3.3 タップを使う	26
2.3.4 ARPキャッシュポイズニング	28
2.4 ルータで構成されたネットワークでのキャプチャ	33

2.5	パケットキャプチャツールを実際に設置する	34
3章	Wireshark概要	37
3.1	Wiresharkの歴史	37
3.2	Wiresharkの利点	37
3.3	Wiresharkのインストール	39
3.3.1	Windowsでのインストール	39
3.3.2	Linuxでのインストール	41
3.3.3	Mac OS Xでのインストール	42
3.4	Wiresharkの基本	43
3.4.1	はじめてのパケットキャプチャ	43
3.4.2	Wiresharkのメインウィンドウ	45
3.4.3	Wireshark設定	46
3.4.4	パケットの色分け	47
4章	Wiresharkでのパケットキャプチャのテクニック	51
4.1	キャプチャファイルの操作	51
4.1.1	キャプチャファイルの保存とエクスポート	51
4.1.2	キャプチャファイルのマージ	53
4.2	パケットの操作	54
4.2.1	パケットの検索	54
4.2.2	パケットのマーキング	55
4.2.3	パケットの印刷	55
4.3	時刻の表示形式と基準時刻表示	56
4.3.1	時刻の表示形式	56
4.3.2	基準時刻表示	57
4.4	キャプチャオプションの設定	58
4.4.1	Captureの設定	59
4.4.2	Capture File 設定	60
4.4.3	Stop Capture 設定	61
4.4.4	Display Options 設定	62
4.4.5	Name Resolution 設定	62
4.5	フィルタを使う	62
4.5.1	キャプチャフィルタ	62
4.5.2	ディスプレイフィルタ	68
4.5.3	フィルタの保存	71
5章	Wiresharkの高度な機能	73
5.1	ネットワークのエンドポイントと対話	73
5.1.1	エンドポイントを参照する	74

5.1.2	ネットワークの対話を見る	75
5.1.3	[Endpoints] と [Conversations] ダイアログを用いたトラブルシューティング	75
5.2	プロトコル階層統計	77
5.3	名前解決	78
5.3.1	名前解決を有効にする	79
5.3.2	名前解決の欠点	79
5.4	プロトコル分析機構	80
5.4.1	分析機構の変更	80
5.4.2	分析機構のソースコードを見る	82
5.5	TCPストリームの表示	83
5.6	パケット長	84
5.7	グラフ表示	85
5.7.1	IOグラフを見る	85
5.7.2	ラウンドトリップタイムグラフ	87
5.7.3	フローグラフ	88
5.8	エキスパート情報	89

6章 知っておきたい下位層プロトコル 93

6.1	ARP (Address Resolution Protocol)	93
6.1.1	ARPヘッダ	95
6.1.2	パケット1: ARPリクエスト	96
6.1.3	パケット2: ARPレスポンス	97
6.1.4	gratuitous ARP	98
6.2	IP (Internet Protocol)	99
6.2.1	IPアドレス	99
6.2.2	IPv4ヘッダ	101
6.2.3	生存時間 (TTL)	102
6.2.4	IPフラグメンテーション (断片化)	104
6.3	TCP	107
6.3.1	TCPヘッダ	107
6.3.2	TCPポート	108
6.3.3	TCPの3ウェイハンドシェイク	111
6.3.4	TCPのティアダウン (切断)	114
6.3.5	TCPリセット	115
6.4	UDP	116
6.4.1	UDPヘッダ	116
6.5	ICMP	117
6.5.1	ICMPヘッダ	118
6.5.2	ICMPタイプとメッセージ	118
6.5.3	エコー要求とエコー応答	119

6.5.4	traceroute	121
7章	知っておきたい上位層プロトコル	125
7.1	DHCP	125
7.1.1	DHCPパケット構造	125
7.1.2	DHCP更新処理	127
7.1.3	DHCPのリース更新	132
7.1.4	DHCPオプションとメッセージタイプ	132
7.2	DNS	132
7.2.1	DNSのパケット構造	133
7.2.2	単純なDNSクエリ	134
7.2.3	DNSの問い合わせタイプ	136
7.2.4	DNSの再帰	137
7.2.5	DNSゾーン転送	140
7.3	HTTP	143
7.3.1	HTTPでブラウズする	143
7.3.2	HTTPでデータをアップロードする	145
7.4	まとめ	146
8章	現実世界のシナリオの第一歩	147
8.1	パケットレベルでのSNSの分析	147
8.1.1	Twitterトラフィックのキャプチャ	148
8.1.2	Facebookトラフィックをキャプチャする	152
8.1.3	TwitterとFacebookの比較	154
8.2	ESPN.comのトラフィックをキャプチャする	154
8.2.1	Conversationsウィンドウの利用	154
8.2.2	[Protocol Hierarchy Statistics] ダイアログの利用	155
8.2.3	DNSトラフィックを参照する	157
8.2.4	HTTPリクエストを見る	157
8.3	現場でのトラブル	159
8.3.1	インターネットに接続できない：設定の問題	159
8.3.2	インターネットに接続できない：不適切なりダイレクト	162
8.3.3	インターネットに接続できない：上位の問題	165
8.3.4	不安定なプリンタ	168
8.3.5	孤立する支社	171
8.3.6	イライラする開発者	175
8.4	まとめ	180
9章	ネットワークの遅延と戦う	181
9.1	TCPのエラーリカバリ機能	181
9.1.1	TCP再送	182

9.1.2 重複ACKと高速再送	184
9.2 TCPのフロー制御	189
9.2.1 ウィンドウサイズの調整	190
9.2.2 ゼロウィンドウ通知でのデータフローの一時停止	191
9.2.3 TCPスライディングウィンドウの実例	192
9.3 TCPエラー制御とフロー制御パケット	195
9.4 高遅延の原因を突き止める	196
9.4.1 正常な通信	197
9.4.2 通信の遅延：回線遅延	197
9.4.3 通信の遅延：クライアントの遅延	198
9.4.4 通信の遅延：サーバの遅延	199
9.4.5 遅延を見つけるフレームワーク	199
9.5 ネットワークベースラインの確立	200
9.5.1 サイトのベースライン	201
9.5.2 ホストベースライン	202
9.5.3 アプリケーションベースライン	203
9.5.4 ベースラインについての追記	203
9.6 まとめ	204
10章 セキュリティとパケット解析	205
10.1 偵察	205
10.1.1 SYNスキャン	206
10.1.2 OSフィンガープリント	210
10.2 侵入	213
10.2.1 Operation Aurora	213
10.2.2 ARPキャッシュポイズニング	218
10.2.3 リモートアクセス型のトロイの木馬	223
10.3 まとめ	231
11章 無線LANのパケット解析	233
11.1 物理面での考察	233
11.1.1 一度に1つのチャンネルをキャプチャする	233
11.1.2 無線LANの電波干渉	234
11.1.3 電波干渉を検出、解析する	235
11.2 無線LANカードのモード	236
11.3 Windows上での無線LANのパケットキャプチャ	238
11.3.1 AirPcapの設定	238
11.3.2 AirPcapを使ったパケットキャプチャ	240
11.4 Linux上での無線LANのパケットキャプチャ	241
11.5 802.11のパケット構造	242

11.6	[Packet List] ペインに無線LANの情報を追加する	244
11.7	無線LAN特有のフィルタ	245
11.7.1	特定のBSSIDでフィルタリング	246
11.7.2	パケット別のフィルタリング	246
11.7.3	周波数によるフィルタ	247
11.8	無線LANのセキュリティ	248
11.8.1	WEP 認証の成功	248
11.8.2	WEP 認証の失敗	250
11.8.3	WPA 認証の成功	251
11.8.4	WPA 認証の失敗	252
11.9	まとめ	253
付録A USBポートの通信キャプチャ		255
A.1	LinuxマシンにつなげたUSBデバイスとの通信をキャプチャする	255
A.1.1	Wiresharkをインストールする	255
A.1.2	USBデバイスをつなげる	255
A.1.3	Wiresharkを起動し、USBインターフェイスを選択する	256
A.1.4	キャプチャ結果の確認	257
A.1.5	他のLinux環境でキャプチャするには	257
A.2	Windowsで認識したUSBデバイスのI/Oをモニタする	258
A.2.1	VirtualBoxをインストールする	258
A.2.2	ユーザーを追加する	258
A.2.3	ホストOSを再起動する	258
A.2.4	VirtualBoxから認識させたいUSBデバイスをつなげる	258
A.2.5	VirtualBoxを起動し、当該デバイスをVirtualBox上から使えるようにする	259
A.2.6	WindowsとWiresharkを起動し、対象となるポートを選択してキャプチャを行う ...	260
A.2.7	キャプチャ結果	261
A.3	まとめ	261
A.4	参考文献	262
付録B pcap-ng形式 ↔ pcap形式のデータ変換		263
B.1	pcap形式とpcap-ng形式の概要	263
B.2	pcapとpcap-ngでデータ変換が必要な理由	264
B.3	変換方法	264
B.3.1	pcap形式 → pcap-ng形式	264
B.3.2	pcap-ng形式 → pcap形式	265
B.3.3	pcapやpcap-ngのファイルサイズが巨大になる場合	265
付録C 推薦文献		267
C.1	パケット解析ツール	267

C.1.1 tcpdumpとWindump 267

C.1.2 Cain & Abel 267

C.1.3 Scapy 268

C.1.4 Netdude 268

C.1.5 Colasoft Packet Builder 268

C.1.6 CloudShark 269

C.1.7 pcapr 269

C.1.8 NetworkMiner 270

C.1.9 Tcpreplay 270

C.1.10 ngrep 270

C.1.11 libpcap 271

C.1.12 hping 271

C.1.13 Domain Dossier 271

C.1.14 PerlとPython 271

C.2 パケット解析に役立つ情報源 271

C.2.1 Wireshark ホームページ 271

C.2.2 SANS Security Intrusion Detection In-Depth Course 272

C.2.3 Chris Sanders のブログ 272

C.2.4 Packetstan ブログ 272

C.2.5 Wireshark University 272

C.2.6 IANA 272

C.2.7 TCP/IP Illustrated (Addison-Wesley) 272

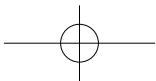
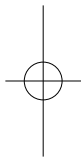
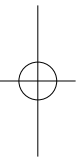
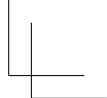
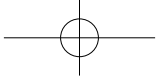
C.2.8 The TCP/IP Guide (No Starch Press) 273

索引 274

コラム目次

「本物の」ハブを見つける 25

ネットワークマップ 34



1章

パケット解析とネットワークの基礎

コンピュータネットワーク上では、スパイウェアの感染といった単純なものからルータの設定エラーといった複雑なものまで、毎日数え切れないほどのトラブルが発生しています。そのすべてを即座に解決するのは不可能ですが、こうしたトラブルの対処に必要な知識とツールをしっかりと用意することで、最良の備えとなるはずです。

ネットワークのトラブルは、すべてパケットレベルまで掘り下げることができます。そこでは、かわいらしい見た目のアプリケーションがその醜い実装をさらけ出し、信用できるように見えるプロトコルも悪意あるものであることを図らずも証明してしまうのです。ネットワークのトラブルを深く理解するためには、パケットレベルでの視点が不可欠です。パケットは嘘をつきません。紛らわしいメニュー構造や、目を引くグラフィック、信頼できない社員によってごまかされることもありません。パケットレベルでは、(暗号化によるもの以外) いかなる隠し立てもできません。パケットレベルでできることが増えるほど、ネットワークを管理下におき、トラブルを解決できるようになります。これがパケット解析の世界です。

本書はパケット解析の世界に頭から飛び込んでいきます。現場に即したシナリオを通じて、ネットワーク遅延との戦い方、アプリケーション起因のボトルネックの特定、ハッカーの追跡術を学んでいきます。本書を読み終える頃には、高度なパケット解析の技術を体得しているはずです。その技術を用いれば、ネットワーク上で起こる多くの困難なトラブルを解決することができますでしょう。

本章では、ネットワーク通信に焦点を当て、基本的なところから始めることで、さまざまなシナリオに対応するための基礎を習得することができますでしょう。

1.1 パケット解析とパケットキャプチャツール

パケット解析 (しばしばパケットキャプチャやプロトコル解析とも呼ばれます) とは、ネットワーク上で起っている事象の理解を助けるために、ネットワークを流れるデータをキャプチャして解析する作業のことです。通常パケット解析は、パケットキャプチャツールを使って行います。これは、回線上

を行き来している生のネットワークデータをキャプチャするツールです[†]。パケット解析は次のようなことに有用です。

- ネットワーク特性の把握
- 誰がネットワーク上にいるかの確認
- 誰、もしくは何が帯域を使っているかの確認
- ネットワークの使用率がピークになる時間の特定
- 潜在的な攻撃や悪意ある行為の特定
- 安全性が低くリソース喰いなアプリケーションの調査

パケットキャプチャツールには、フリーと商用どちらも多くの種類があり、各々異なる設計思想を持っています。著名なものとしては、tcpdump、OmniPeek、Wireshark（本書で扱う）があります。OmniPeekとWiresharkはGUIがあります。

1.1.1 パケットキャプチャツールの評価

使用するパケットキャプチャツールを決めるには、以下を含むいくつかの観点を考慮する必要があります。

サポートされているプロトコル

パケットキャプチャツールはさまざまなプロトコルを解析することができます。ほとんどのパケットキャプチャツールが、一般的なネットワーク層のプロトコル（IPv4やICMPなど）、トランスポート層のプロトコル（TCPやUDPなど）、アプリケーション層のプロトコル（DNSやHTTPなど）を解析することができます。一方で、あまり一般的でないものや、新しいもの（IPv6、SMBv2、SIPなど）はサポートしていない場合があります。パケットキャプチャツールを選ぶときは、使用する予定のプロトコルがサポートされているかを確認しましょう。

操作性

パケットキャプチャツールの画面、インストールの容易性、一般的な機能の操作性を検討しましょう。経験に応じたプログラムを選択することが肝要です。パケット解析の経験がほとんどないのであれば、tcpdumpのような高度なコマンドラインのパケットキャプチャツールは避けたほうがよいでしょう。逆に経験豊富なら、高度なプログラムのほうがよいかもしれません。パケット解析の経験を積むと、シナリオによっては複数のパケットキャプチャツールを組み合わせるのがよいという場合もあるかもしれません。

コスト

パケットキャプチャツールの素晴らしいところは、商用の製品に匹敵するフリーの製品が数

[†] 監訳注：「スニファー」と呼ぶことがありますが、これは本来製品の名前で（本書執筆時点では米ネットワーク・アソシエーツ・テクノロジーの）登録商標です。以前商用のパケットキャプチャツールといえば「スニファー」がデファクトスタンダードであった時代があったため、現場では広く用いられていますが、注意してください。

多く存在することです。商用製品とフリーの製品のもっとも大きな違いは、レポーティング機能です。商用製品には、通常何らかのレポーティング機能がありますが、フリーの製品にはまず含まれていません。

サポート

パケットキャプチャツールの基本を習得しても、問題を解決するためのサポートが必要になるときがあるでしょう。サポートを評価する際には、開発者向けのドキュメント、公開されているフォーラムやメーリングリストを探してみてください。Wiresharkのようなフリーのパケットキャプチャツールでは開発者向けのサポートはあまりないかもしれませんが、ユーザーのコミュニティがそれを補ってあまりある場合がよくあります。ユーザーや貢献者のコミュニティでは、議論のための掲示板、Wikiやブログを提供しており、使用しているパケットキャプチャツールについて深く知るための手助けをしてくれます。

OSのサポート

残念ながら、すべてのパケットキャプチャツールがどんなOSでも使えるわけではありません。サポートが必要なOSすべてで使えるものを選びましょう。コンサルタントであれば、さまざまなOSでパケットをキャプチャし解析する必要があるので、多くのOSで動作するツールが必要になります。あるコンピュータ上でキャプチャしたパケットを別のコンピュータで参照する場合についても考えておく必要があります。OS間の差異のため、機器ごとに異なるツールを使わざるを得ない場合もあります。

1.1.2 パケットキャプチャツールの仕組み

パケットキャプチャツールの処理は、ソフトウェアとハードウェアが連携して行われます。この処理は以下の3つのステップに分けることができます。

キャプチャ

最初のステップでは、パケットキャプチャツールがケーブルからバイナリの生データをキャプチャします。これは通常、キャプチャしたいネットワークに接続されているネットワークカードをプロミスキャスモードに切り替えることによって行われます。プロミスキャスモードでは、自分が宛先になっているトラフィックだけではなく、ネットワークセグメント上を流れるすべてのトラフィックをネットワークカード経由でキャプチャできます。

変換

次のステップでは、キャプチャされたバイナリのデータを参照可能な形式に変換します。高度なコマンドラインベースのパケットキャプチャツールの多くは、ここまでしかやりません。ここでは、非常に基本的な変換しか行われません。解析のほとんどはユーザーに任されています。

解析

最後のステップでは、変換されたデータが実際に解析されます。パケットキャプチャツールは、キャプチャされたデータを元にプロトコルを特定し、プロトコルに応じた解析を開始します。

1.2 コンピュータはどのように通信するのか

パケット解析をきちんと理解するためには、コンピュータ同士がどうやって通信しているのかをきちんと理解することが必要です。この節ではOSI参照モデル、ネットワークのデータフレーム、それらをサポートするハードウェアといったネットワークプロトコルの基礎を勉強します。

1.2.1 プロトコル

現在のネットワークは、さまざまなプラットフォームで動作する多種多様なシステムで構成されています。これらの間で通信を行うために、**プロトコル**と呼ばれる共通の言語が使われます。一般的なプロトコルとしては、TCP、IP、ARP、DHCPといったものがあります。**プロトコルスタック**とは、連携して動作するプロトコルを論理的にグループ化したものです。

プロトコルを理解するには、人の話し言葉や書き言葉を司っている規則と同じようなものと捉えてみるのがよいでしょう。すべての言語には、どのように動詞を活用するか、どのようにあいさつするか、どのように感謝するのが適切かといった規則があります。プロトコルも同じようなもので、どのようにパケットをルーティングするか、どのようにコネクションを開始するか、どのようにデータの受信を通知するかといった事項を定義しています。

プロトコルは機能に応じて単純にも複雑にもなり得ます。性格がまったく異なるプロトコルが数多く存在しますが、多くのプロトコルは以下のような機能を持っています。

コネクションの開始

コネクションを開始するのはクライアントか、サーバか。コネクション確立前にやり取りされるべき情報とは。

コネクションのオプションのネゴシエーション

このプロトコルの通信は暗号化されているか。通信するコンピュータ間でやり取りされる暗号鍵は、どのように共有されるのか。

データのフォーマット

パケットに含まれるデータの順序は？ 受信した通信機器が処理するデータの順序は？

エラー検出と訂正

パケットが宛先に届くまでに時間がかかりすぎた場合、どうなるのか。サーバとの通信が短時間で確立できなかった場合、クライアントはどのように対処するのか。

コネクションの切断

あるコンピュータに対して、別のコンピュータとの通信の切断をどのように通知するのか。
通信をきちんと終了するために最後にやり取りすべき情報は何か。

1.2.2 OSI参照モデル

プロトコルは、OSI参照モデルと呼ばれる業界標準の参照モデルを元に、機能ごとに分けられています。OSI参照モデルは、ネットワーク通信の処理を図1-1のような7つの階層に分けています。この階層モデルのおかげで、ネットワーク通信というものが理解しやすくなっています。最上層のアプリケーション層はネットワークのリソースにアクセスするプログラムそのものを表しています。最下層は物理層で、ネットワーク上で実際にデータの転送を行う層です。各層のプロトコルは、その上位層また下位層のプロトコルによってデータが適切に処理されるよう、連携して機能します。

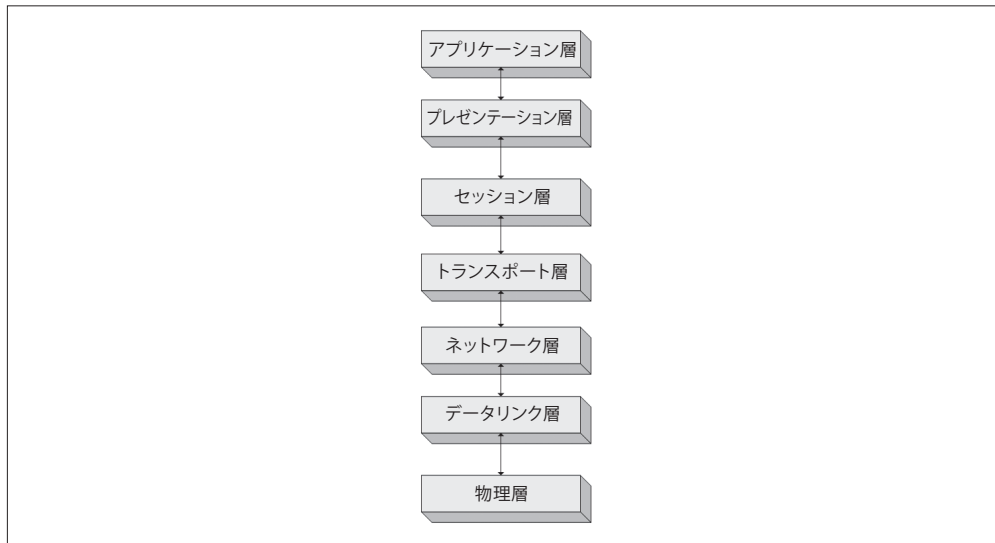


図1-1 OSI参照モデルの7つの層の階層図



OSI参照モデルは1983年にISO（International Organization for Standardization：国際標準化機構）によって、ISO7498として公開されました。OSI参照モデルは業界が推奨している標準以上のものではありません。プロトコルの開発者は、このモデルに厳密に準拠する必要はありません。実際、現存するネットワークモデルはOSI参照モデルだけではありません。たとえば、TCP/IPモデルとしても知られているDoD（Department of Defense：米国国防総省）モデルを好む人もいます。

OSI参照モデルの各階層には、次のような固有の機能があります。

アプリケーション層（第7層）

OSI参照モデルの最上位層は、ユーザーがネットワーク上のリソースにアクセスするための手段を提供します。これは通常エンドユーザーから見える唯一の層であり、ネットワーク上のすべての活動の基点となるインターフェイスを提供します。

プレゼンテーション層（第6層）

この層は、受信したデータをアプリケーション層が読み取ることのできる形式に変換します。この層でデータをエンコード、デコードする方法は、送受信するデータのアプリケーション層のプロトコルに依存します。この層では、データのセキュリティ維持のための暗号化や復号も行います。

セッション層（第5層）

この層は2台のコンピュータ間の「対話」すなわちセッションを制御し、通信機器間の接続の確立、管理、切断を行います。セッション層は、接続が全二重か半二重かを制御するとともに、通信を唐突に遮断するのではなく適切に切断するための制御も行います。

トランスポート層（第4層）

トランスポート層の主要な目的は、下位層に信頼できるデータ転送サービスを提供することです。フロー制御、データの分割と再構築、誤り制御といった機能により、トランスポート層は2点間のデータのやり取りをエラーなしで行えるわけです。信頼性の高いデータ転送を担保することは極めて難しいため、OSI参照モデルでは、1つの層をその目的に割り当てています。トランスポート層は、接続指向のサービスと接続レスのサービスの両方を提供します。ファイアウォールやプロキシサーバには、この層で動作するものもあります。

ネットワーク層（第3層）

この層は、物理的なネットワークを越えて転送されるデータのルーティングを提供します。OSI参照モデルの中でもっとも複雑な層のひとつであり、ネットワーク上のコンピュータの論理アドレス（IPアドレスなど）のアドレス指定（アドレッシング）を行います。この層では、パケットの分割や、場合によっては誤り検出も行います。ルータはこの層で動作します。

データリンク層（第2層）

この層は、物理的なネットワーク上でデータを転送する手段を提供します。主な機能は、物理的な通信機器を特定するためのアドレス指定スキーム（MACアドレスなど）を提供することです。ブリッジとスイッチはデータリンク層で動作する物理的な通信機器です。

物理層（第1層）

物理層はOSI参照モデルの最下層であり、ネットワーク上でデータを転送するための物理的な媒体です。この層では、電圧、ハブ、ネットワークアダプタ、リピータ、ケーブル仕様と

いった、使用されるハードウェアの物理的、電気的な特性を定義します。物理層は、コネクションを確立および切断し、通信リソースを共有する手段を提供し、信号をデジタルからアナログ、またはその逆に変換します。

OSI参照モデルの各層において、一般的に使用されているプロトコルの代表的なものを表1-1に示します。

表1-1 OSI参照モデルの各層で使用される代表的なプロトコル

層	プロトコル
アプリケーション層	HTTP, SMTP, FTP, Telnet
プレゼンテーション層	ASCII, MPEG, JPEG, MIDI
セッション層	NetBIOS, SAP, SDP, NWLink
トランスポート層	TCP, UDP, SPX
ネットワーク層	IP, IPX
データリンク層	Ethernet, Token Ring, FDDI, AppleTalk

OSI参照モデルは標準として推奨される以上のものではありませんが、頭に入れておくべきです。本書を読み進めるにつれ、プロトコルの階層に応じて、ネットワークのトラブルに対する対処が変わることがわかるでしょう。ルータの問題は「第3層の問題」となり、ソフトウェアの問題は「第7層の問題」として扱われるのです。



本書の執筆中の議論で、あるユーザーがネットワーク上のリソースにアクセスできないと文句を言ってきた話を同僚がしてくれました。このトラブルはユーザーが間違ったパスワードを入力したために起きたもので、同僚はこれを「第8層の問題」と言っていました。第8層はユーザー層を意味する非公式な用語で、パケットを扱っている人々の間ではよく使われています。

OSI参照モデルにおいて、データの流れはどのようになっているのでしょうか。ネットワーク上を転送されるデータの流れは、送信側のシステムのアプリケーション層から始まります。データは各層ごとの方法でOSI参照モデルの7つの階層を送信側のシステムの物理層まで下っていき、受信側のシステムに送られます。受信側のシステムは物理層でデータを受信し、データは最上層のアプリケーション層まで受信側のシステムの各層を上がっていきます。

OSI参照モデルの各層でさまざまなプロトコルによって提供されるサービスは重複しません。言い換えると、ある層のあるプロトコルが提供しているサービスと同じものを、ほかの層のプロトコルが提供することはありません。異なる層のプロトコルが類似の目的のための機能を備えていたとしても、その働きは多少異なっています。

送信側と受信側のコンピュータで、同じ層のプロトコルは相補的な関係にあります。たとえば送信側のコンピュータの第7層のプロトコルが転送されるデータを暗号化する場合、受信側のコンピュー

タの第7層のプロトコルはデータを復号することを求められます。

図1-2は、通信している2台のコンピュータにおけるOSI参照モデルを図示したものになります。片方のコンピュータの最上層から最下層を通り、もう片方のコンピュータに到達した後、その逆をたどることで通信が成立します。

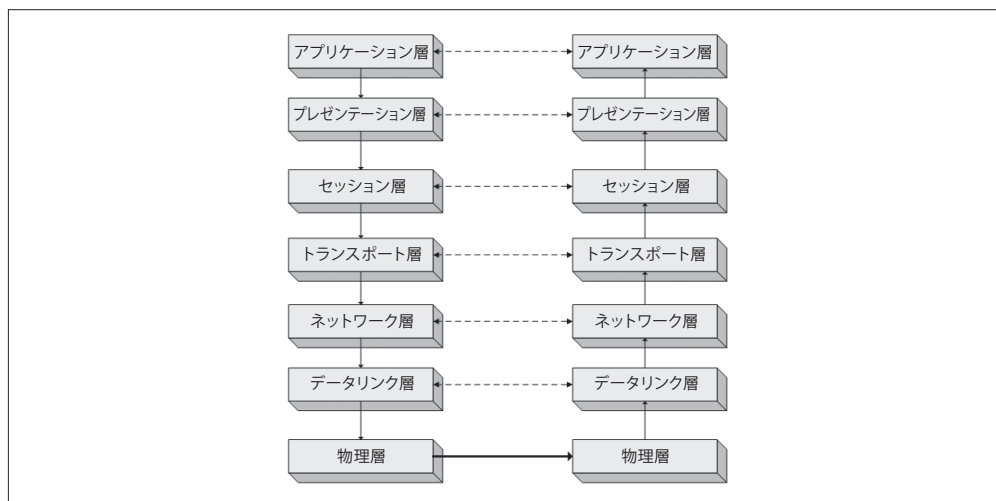


図1-2 送信側と受信側のコンピュータの双方において、同じ階層構造で機能するプロトコル

OSI参照モデルの各層は、その直上または直下の層としか通信できません。たとえば、第2層は第1層もしくは第3層としかデータの送受信ができません。

1.2.3 データのカプセル化

OSI参照モデルの異なる層のプロトコルが通信するには、データをカプセル化する必要があります。スタックの各層では、その層が通信を行うのに必要とするビット情報を、ヘッダやフッタとして通信するデータに追加することが求められます。たとえばトランスポート層がセッション層からデータを受信した場合、トランスポート層は次の層にデータを渡す前にヘッダ情報を追加します。

カプセル化処理とは、PDU (Protocol Data Unit : プロトコルデータユニット) を生成することを意味します。PDUには、送信されるデータと追加されたヘッダおよびフッタ情報のすべてが含まれます。データがOSI参照モデルに従って階層を下っていく際に、PDUはさまざまなプロトコルが追加するヘッダ情報とフッタ情報によって大きくなっていきます。PDUは物理層に到達した時点で最終的な形態となり、受信側のコンピュータに送られます。受信側のコンピュータは、データがOSI参照モデルの階層が上がっていくにつれ、プロトコルのヘッダおよびフッタをPDUから取り除いていきます。PDUがOSI参照モデルの最上層に到達するときには、元々のデータしか残っていません。

★原書の図1-2のデータに不備（配置画像が埋め込まれていない）があり、原書データ内の予備画像（横の点線矢印があるもの）を流用しました。これはこれでわかりやすいと思うのですが、よろしいでしょうか？



パケットという単語は、OSI参照モデルの各層が追加するヘッダとフッタをすべて含んだ完全なPDUのことを意味します。

カプセル化されたデータがどのように機能するかを理解するのは少々ややこしいので、構築、送信、受信されるパケットの実例を、OSI参照モデルとの関連で見ましょう。パケット解析の際には、セッション層やプレゼンテーション層についてはあまり触れることがないので、この例では（そして本書のほかの例でも）登場しないことを心に留めておいてください。

`http://www.google.com/` をブラウザしようとしているコンピュータを例にとりて説明しましょう。この処理を実行するには、要求パケットを生成し、送信側のクライアントコンピュータから、受信側のサーバコンピュータへと送信する必要があります。ここではTCP/IPのセッションはすでに開始されているものとしします。図1-3はこの例におけるデータのカプセル化を表したものです。

処理は、クライアントコンピュータのアプリケーション層から始まります。ウェブサイトをブラウザするので、アプリケーション層のプロトコルとしてHTTPが用いられ、`index.html` ファイルを `google.com` からダウンロードする指示が発行されます。

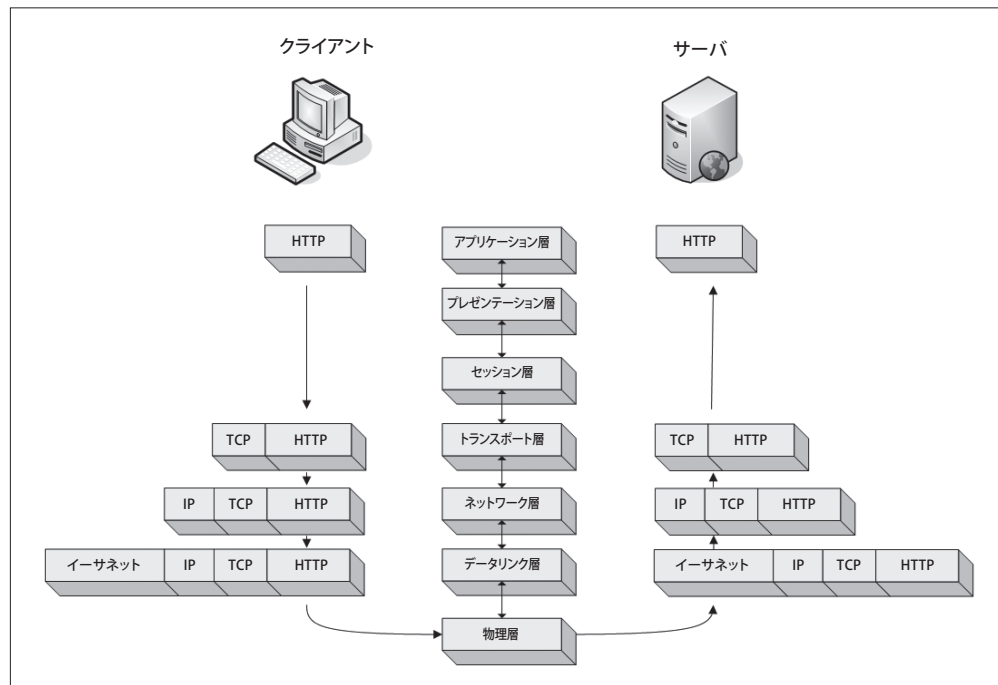


図1-3 クライアントとサーバ間でデータがカプセル化される模様

アプリケーション層のプロトコルが実行したい指示を発行したら、次に着目するのはパケットを目的地に到達させることです。パケットのデータはスタックを下り、トランスポート層へと渡されます。HTTPはTCPの上にあるアプリケーション層プロトコルです。そのためTCPは、パケットが確実に送信されるようにするトランスポート層プロトコルとして機能し、TCPヘッダを生成します。これにはシーケンス番号などパケットに追加されたデータが含まれており、パケットが適切に送信されるようになっています。



あるプロトコルが別のプロトコルの「上にある」という表現をしばしば使うのは、OSI参照モデルが上下という概念を用いているためです。HTTPなどのアプリケーション層プロトコルは、ある特定のサービスを実現しますが、そのサービスを提供するにはTCPが必要となります。今後学んでいきますが、DNSはUDPの上に、TCPはIPの上にあります。

TCPは仕事を終わると、パケットの論理アドレスを処理する第3層プロトコルであるIPにパケットを渡します。IPは論理アドレス情報を含むヘッダを生成し、パケットをデータリンク層のイーサネットに渡します。物理アドレスであるイーサネットアドレスはイーサネットのヘッダに保持されます。ここで完成したパケットが物理層に渡され、0と1の形式でネットワーク上を転送されます。

パケットはネットワークケーブルを転送され、最終的にGoogleのウェブサーバへとたどり着きます。ウェブサーバはパケットを下位層から順に読み取っていきます。最初に、パケットがどのサーバに対するものかを判断するためにネットワークカードが用いるイーサネットアドレスが含まれる、データリンク層が読み取られます。この層の情報が処理されると、第2層の情報が取り除かれ、第3層の情報が処理されます。

IPアドレス情報も第2層の情報と同じように読み取られ、アドレスが適切で、パケットが分割されていないことが確認されます。その後、この情報は取り除かれ、次の層が処理できるようになります。

引き続き、第4層のTCP情報が読み取られ、パケットが順番に到着していることが確認されます。第4層のヘッダ情報を取り除くと、アプリケーション層のデータのみが残るので、ウェブサイトを提供しているウェブサーバのアプリケーションへ渡せるようになります。クライアントから送信されたこのパケットに応答する際、サーバはまずTCPのACKパケットを送って、リクエストが受信され、引き続きindex.htmlファイルが送信されてくることを受け取ったことをクライアントに知らせておくべきです。

パケットはすべて、使用するプロトコルにかかわらず、この例で説明したように構築されて処理されます。ただし、ネットワークを流れるすべてのパケットがアプリケーション層プロトコルから生成されるわけではないことを知っておく必要があります。第2層、第3層、第4層で生成された情報しか含まれないパケットも存在するということです。

1.2.4 ネットワークハードウェア

ここからは、汚れ仕事を引き受けてくれるネットワークハードウェアを見ていきましょう。ここではハブ、スイッチ、ルータといった一般的なネットワークハードウェアに焦点を絞ります。

1.2.4.1 ハブ

ハブ[†]とは、通常図1-4のNETGEARのハブのように、RJ-45のポートを複数持つただの箱にすぎません。ハブには4ポートといった非常に小型なものから、企業用としてラックマウント用に設計された48ポートといった大型のものまで存在します。

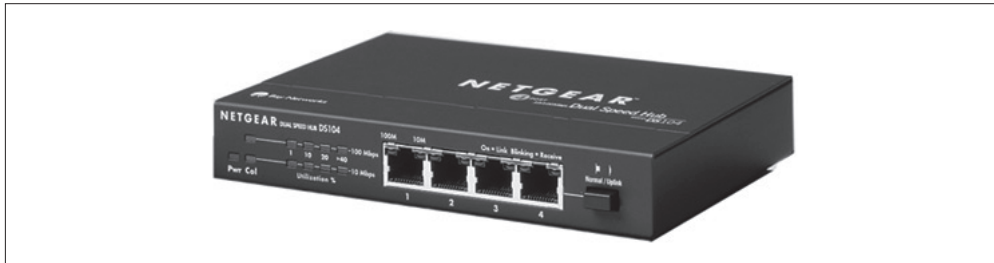


図1-4 典型的な4ポートのイーサネットハブ

ハブは不要なネットワークトラフィックを大量に生み出し、(データの送受信を同時に行うことができない) 半二重モードでしか動作しないため、最近の集約化されたネットワークで見かけることはまずないでしょう(代わりにスイッチが使用されます)。しかし、2章で説明する「ハブの使用」テクニックを用いる場合、ハブがパケット解析に非常に重要となるため、ハブの動きは知っておく必要があります。

ハブはOSI参照モデルの物理層で動作する、データの中継を行う通信機器以上のものではありません。この通信機器は、あるポートに送信されたパケットをすべてのポートに伝送(中継)します。たとえば、コンピュータが4ポートハブのポート1に接続されていて、ポート2に接続されているコンピュータにデータを送信する場合、ハブはパケットをポート1、2、3、4のすべてに送信します。ポート3とポート4に接続されているコンピュータは、パケットのイーサネットヘッダにあるMAC(Media Access Control)アドレスで宛先を確認し、パケットが自分宛でないことを確認すると、ドロップ(破棄)します。図1-5はコンピュータAがコンピュータBにデータを送信する例を示したものです。コンピュータAがデータを送信すると、ハブに接続されているすべてのコンピュータがこのデータを受信します。しかし、実際にデータを受け取るのはコンピュータBのみで、B以外のコンピュータはそれを破棄します。

たとえば、マーケティング部で働いている人のみでなく、その企業の社員全員に、メールの題名に「マーケティング部の皆さまへ」と書いたメールを送信したとしましょう。マーケティング部の社員はメールが自分宛であることがわかりますから、そのメールを開封するでしょう。しかし他の社員はメールが自分宛でないことがわかるので、おそらくそれを破棄します。多くの不要な通信と無駄な時間がなぜ発生するかがおわかりになるでしょう。これがハブの機能です。

[†] 監訳注：ここで述べているハブは、シェアードハブ、リピータハブのことです。

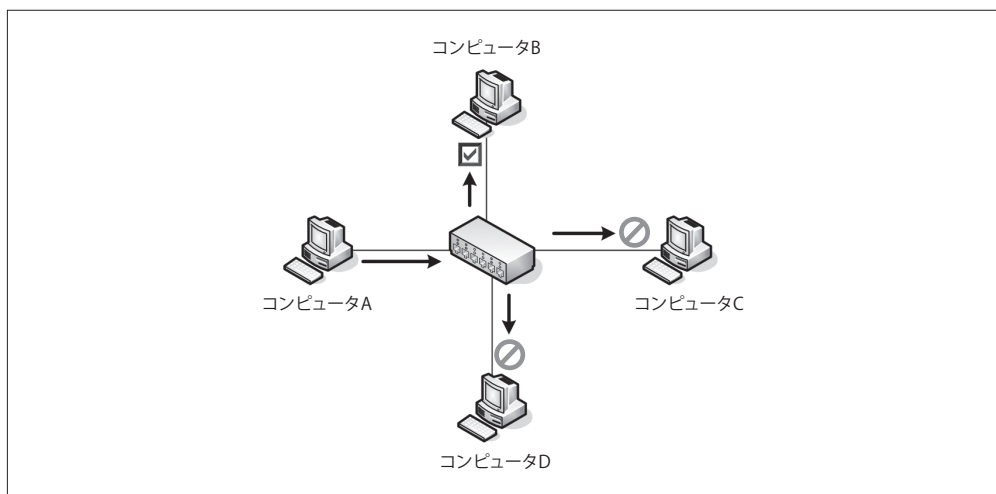


図1-5 コンピュータAがコンピュータBにハブを経由してデータを送信する際のトラフィックの流れ

商用の集約化されたネットワークにおいて、ハブに替わる最良の機器はスイッチです。スイッチはデータの送受信が同時にできる**全二重モード**の通信機器です。

1.2.4.2 スイッチ

ハブと同じく、スイッチ[†]はパケットを中継するよう設計されています。しかしハブのようにすべてのポートにデータを送信するのではなく、宛先となるコンピュータのみにデータを送信します。スイッチの見た目はハブとよく似ています (図1-6)。

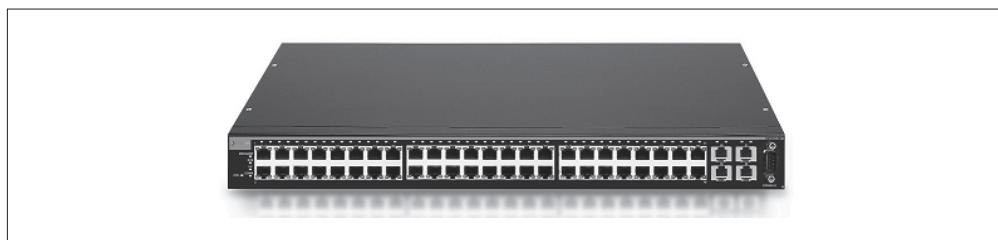


図1-6 ラックマウント型の24ポートイーサネットスイッチ

Cisco製のものなど、市場にはいくつかの大型のスイッチが出回っており、ベンダー固有のソフトウェアやWebブラウザベースのインターフェイスで管理することができます。これらのスイッチは**マネジメントスイッチ**と呼ばれ、ネットワークを管理する際に便利なさまざまな機能を持っています。これには、特定のポートの有効化、無効化、ポートの詳細表示、設定の変更、リモートからの再起動といった

[†] 監訳注：ここでいうスイッチは、マネジメントスイッチもしくはスイッチングハブのことです。

機能が含まれます。

スイッチはパケットの転送を処理するための高度な機能も持っています。特定の機器と直接通信できるようにするため、スイッチは通信機器を MAC アドレスで識別します。つまり、スイッチは OSI 参照モデルのデータリンク層で動作するということです。

スイッチは、接続されているすべての通信機器の第2層のアドレスを、トラフィックの見張り番のような働きをする **CAM テーブル** に記録しています。パケットが送信されると、スイッチはパケット内にある第2層のヘッダ情報を読み取り、CAM テーブルを参照してどのポートにパケットを送信するかを決定します。スイッチは特定のポートにしかパケットを送信しないため、ネットワークトラフィックを劇的に減らすことができます。

図1-7はスイッチを経由したトラフィックの流れを図で示しています。コンピュータAはデータを宛先として想定しているコンピュータBのみに送信しています。ネットワーク上では同時に複数の通信ができますが、情報はスイッチと宛先のコンピュータ間で直接やり取りされ、他のコンピュータには送られません。

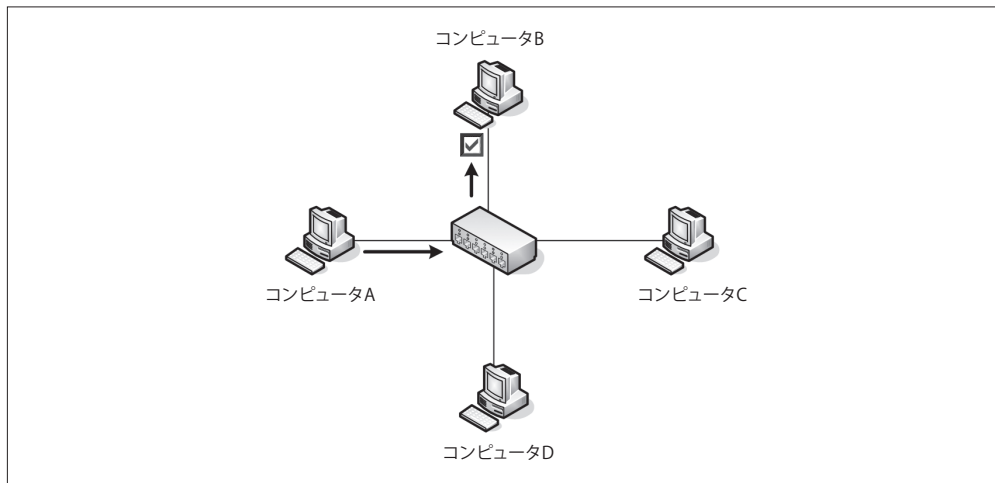


図1-7 スwitchを経由してコンピュータAがコンピュータBにデータを送信する際のトラフィックの流れ

1.2.4.3 ルーター

ルーターはスイッチやハブより上位層の機能を持つ高度なネットワーク機器です。ルーターはさまざまな形のものがありますが、多くは前面にインジケータランプ (LED)、背面にポートがあります。ポートの数はネットワークの大きさに依存します。図1-8はルーター製品の一例です。



図1-8 中小規模ネットワーク用の小型ルータ

ルータはOSI参照モデルの第3層で動作し、複数のネットワーク間でパケットを転送します。ネットワーク間のトラフィックの流れを制御することを、**ルーティング**と呼びます。さまざまな種類のパケットに対して他のネットワークにルーティングする方法を指示するプロトコルを、**ルーティングプロトコル**と呼びます。これにはいくつかの種類があります。ルータは、通常ネットワーク上の通信機器を識別するために、IPアドレスのような第3層のアドレスを使用します。

ルーティングの概念をイメージする方法のひとつは、いつかの通りがある町内を考えてみることです。図1-9のように、コンピュータを家とその住所、ネットワークセグメントを通りだと考えてみます。ある通りにある家は、同じ通りに面しているすべての家と簡単に行き来することができます。これはスイッチに接続することによって、同一ネットワークセグメント上のすべてのコンピュータと通信できることとよく似ています。一方、別の通りの友人を訪ねるのは、同じセグメント上にないコンピュータと通信することに似ています。

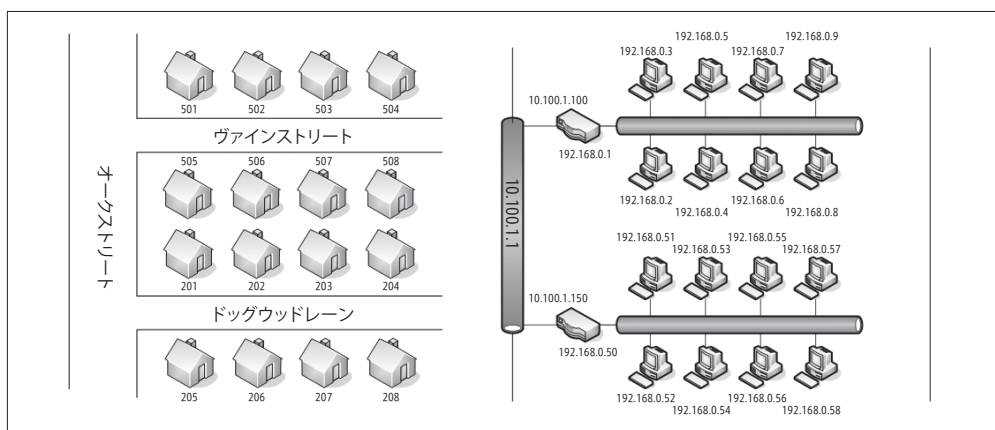


図1-9 ルーティングと町内の通りとの比較

図1-9のヴァインストリート503からドッグウッドレーン202に行く必要があるとします。そのためには、オークストリートを通してドッグウッドレーンに行かなければいけません。これを、ネットワーク

セグメントをまたがる場合で考えてみてください。192.168.0.3の通信機器が192.168.0.54の機器と通信する必要がある場合、ルータを経由して10.100.1.1のネットワークを通り、宛先の通信機器が存在するネットワークセグメントに到達する前に、そのセグメントのルータを経由します。

ネットワーク上のルータの大きさや数は、ネットワークの大きさや機能によって変わります。個人やホームオフィスのネットワークの場合は、ネットワークの中央に置かれたルータ1台のみで構成されているでしょうし、巨大企業のネットワークではいくつものルータがさまざまな部門に置かれ、それらすべては中央の巨大なルータや第3層（L3）スイッチ（スイッチが高度化したもので、ルータの機能が内蔵されている）に接続されているでしょう。

多くのネットワーク構成図を見ることで、さまざまなポイントを経由するデータの流れを理解することができるようになりますでしょう。図1-10はルーティングされたネットワークの一般形を示しています。この例では、2つのネットワークが1つのルータで接続されています。ネットワークAのコンピュータがネットワークBのコンピュータと通信する場合、送信されるデータは必ずルータを経由しなければなりません。

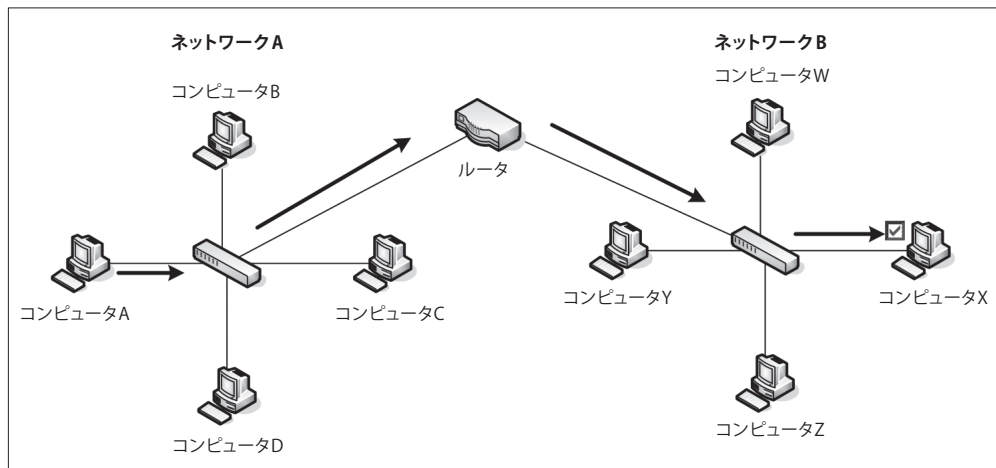


図1-10 コンピュータAがコンピュータXにルータを介してデータを送信したときのトラフィックの流れ

1.3 トラフィックの分類

ネットワークトラフィックは、ブロードキャスト、マルチキャスト、ユニキャストの3つに分類することができます。これらはそれぞれ異なる特徴を持っています。これによってネットワークハードウェアのパケットの扱い方が決まります。

1.3.1 ブロードキャスト

ブロードキャストパケットは、ネットワークセグメント上のすべてのポートに送信されます。これは

ポートがハブ、スイッチのいずれであっても変わりません。

しかしながら、すべてのブロードキャストトラフィックが同じように生成されているわけではなく、第2層の形式と第3層の形式があります。第2層の形式の場合、MACアドレスFF:FF:FF:FF:FF:FFがブロードキャスト専用アドレスとして予約されており、このアドレス向けのトラフィックはネットワークセグメント全体にブロードキャストされます。第3層にも専用のブロードキャストアドレスがあります。

IPネットワークの場合、アドレス帯の中でもっとも高位のIPアドレスがブロードキャストアドレスとして用いられます。たとえばアドレス帯が192.168.0.xxxで、サブネットマスクが255.255.255.0となっているネットワークの場合、192.168.0.255がブロードキャストアドレスとなります。

さまざまな媒体を通して接続されている多くのハブやスイッチからなる巨大ネットワーク上では、あるスイッチから送信されたブロードキャストパケットが、スイッチからスイッチへと中継され、ネットワーク上のはかのスイッチに存在するすべてのポートまで到達します。ブロードキャストパケットが到達する範囲をブロードキャストドメインと呼びます。これはルータを経由せずにコンピュータからコンピュータに直接到達できるネットワークセグメントを指します。図1-11は小さなネットワーク上の2つのブロードキャストドメインの例を示しています。ルータに到達するまでがブロードキャストドメインなので、ブロードキャストパケットは特定のブロードキャストドメイン内にのみ到達します。

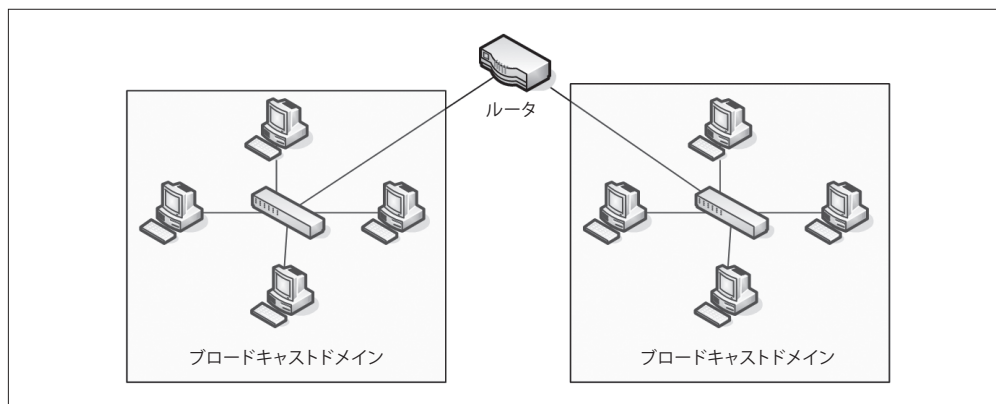


図1-11 ブロードキャストドメインはルータに到達するまでの範囲

先ほどルーティングを町内の家にたとえて説明しましたが、ブロードキャストドメインの動作についても同じことが言えます。ブロードキャストドメインは家のある通りだと考えてみてください。もしポーチに立って叫んだら、その通りにいる人たちはそれを聞くことができます。別の通りの人と話したい場合は、ポーチからブロードキャストする（叫ぶ）のではなく、直接その人と話す方法を見つける必要があります。

1.3.2 マルチキャスト

マルチキャストは、1つの送信元から複数の宛先に同時にパケットを送信する手段です。できる限り

少ない帯域を使い、できる限り単純にこの処理を行うためのものです。宛先に到達するためにデータの複製された回数がトラフィック最適化の度合になります。マルチキャストのトラフィックを実際に扱う方法は、個々のプロトコルの実装に大きく依存しています。

マルチキャストの一般的な実装は、パケットを受信するコンピュータをマルチキャストグループとしてグループ化し、そのグループにアドレスを割り当てる方式であり、IPマルチキャストもこの方式です。マルチキャストグループにアドレスを割り当てることによって、パケットを受け取るべきでないコンピュータにパケットを送信しないようにします。IPでは、マルチキャストに一定のアドレス帯を割り当てています。224.0.0.0から239.255.255.255の範囲のIPアドレスは、マルチキャストトラフィックであると考えてよいでしょう。

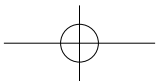
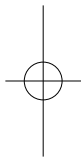
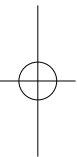
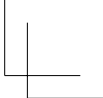
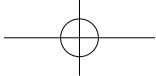
1.3.3 ユニキャスト

ユニキャストパケットはコンピュータからコンピュータへ直接送信されます。ユニキャストがどう機能するかは、使用するプロトコルによって決まります。

たとえば、Webサーバと通信したい機器があるとします。これは1対1の接続なので、通信の処理はクライアントの機器がWebサーバ（だけ）にパケットを送信するところから始まります。これが、ユニキャストトラフィックの一例となります。

1.4 まとめ

本章で解説した内容は、パケット解析の基礎知識として不可欠な基本中の基本です。ネットワークのトラブルシューティングを行う前に、ネットワーク通信で何が起きているかを理解しなければいけません。次の章ではこうした考えに基づき、ネットワーク通信のより高度な原理について話を進めていきます。



2章

ケーブルに潜入する

パケット解析が有益なものとなるかどうかの鍵は、データを適切にキャプチャするうえで、パケットキャプチャツールをどこに設置するかは決定です。パケット解析を行うエンジニアの間では、これをケーブルを監視する、ネットワークに潜入 (tap) する、あるいはケーブルに潜入するなどと言います[†]。簡単に言えば、ネットワーク上の適切な位置にパケットキャプチャツールを設置する作業です。

残念ながらパケットのキャプチャは、単純にノートPCをネットワークポートに接続してパケットをキャプチャすればよいというものではありません。実際には、パケットを解析するよりもパケットキャプチャツールを配置するネットワーク上の位置を決めるほうが難しいこともあります。

なぜパケットキャプチャツールの配置が難しいかというと、ネットワーク機器の種類が非常に多いからです。典型的な状況を図2-1に示します。最近のネットワークで使われている主なネットワーク機器であるハブ、スイッチ、ルータの3つは、それぞれがまったく違った形で通信データを扱うため、解析するネットワークにパケットキャプチャツールを設置する際にはその点を十分考慮しなければなりません。

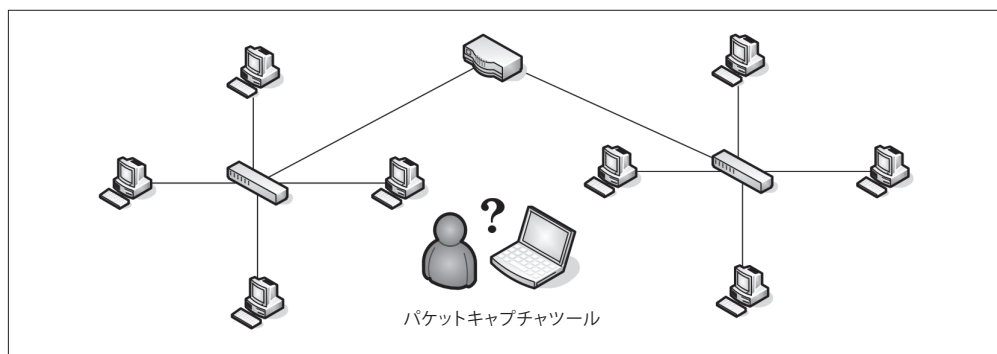


図2-1 パケットキャプチャツールを設置する位置を決めることが一番難しいかもしれない

[†] 監訳注：監訳者のまわりでは「キャプチャを仕込む」、「キャプる」といった言われ方をしていることが多いようですが、これが一般的かどうかはわかりません。

この章の目的は、さまざまなネットワークポロジにおけるパケットキャプチャツールの設置方法を理解することです。まずは潜入したネットワークを流れるすべてのパケットを実際に見るにはどうすればよいかを見ていきましょう。

2.1 プロミスクラスモードの使用

ネットワーク上のパケットを監視するには、プロミスクラスモードをサポートしている NIC (Network Interface Card: ネットワークインターフェイスカード) が必要です。ネットワークを流れるすべてのパケットを参照することができるようにするのが、プロミスクラスモードです。

1章のブロードキャストトラフィックで学んだように、あるホストが実際には自分が宛先でないパケットを受け取る、といったことがよくあります。特定の IP アドレスに対応する MAC アドレスを決定するのに使われる ARP はどんなネットワークにおいても必需品ですが、宛先以外のホストに送られるトラフィックの好例です。一致する MAC アドレスを見つけるために、ARP は正しいクライアントが応答してくれるよう祈りつつ、ブロードキャストドメインにあるすべての通信機器に、ブロードキャストパケットを送信します。

ブロードキャストドメイン (ある機器が他の機器にルータを経由せずに直接送信できるネットワークセグメント) には複数のホストが含まれますが、送信された ARP ブロードキャストパケットに反応するのは1台だけです。ネットワーク上のすべてのホストが ARP ブロードキャストパケットを実際に処理するのは、とてつもなく非効率だからです。代わりに、通信機器の NIC が自分宛でないパケットは自分には役に立たないと認識し、ホスト上の処理に回さず廃棄します。

パケットの廃棄は、宛先以外のホストの処理効率を向上させますが、パケット解析にとってはあまり都合がよくありません。パケット解析を行う側としては、ネットワークを流れるすべてのパケットを参照して、重要な情報の断片を見逃したくないからです。

NIC のプロミスクラスモードを使えば、トラフィックのすべてを確実にキャプチャすることが出来ます。プロミスクラスモードで操作する場合、アドレスにかかわらず、NIC はすべてのパケットをホストに渡します。パケットがホストに引き渡されると、パケットキャプチャツールのアプリケーションがそれを取得して解析します。

最近の NIC のほとんどがプロミスクラスモードをサポートしています。Wireshark は libpcap/WinPcap ドライバを同梱しているため、Wireshark の GUI から NIC を直接プロミスクラスモードに切り替えることができます (libpcap/WinPcap については3章で詳しく解説します)。

本書の目的を実現するには、NIC とプロミスクラスモードの使用が可能な OS が必要になります。プロミスクラスモードでキャプチャする必要がないのは、キャプチャしているインターフェイスの MAC アドレスに直接送られるトラフィックのみを参照したい場合だけです。



Windows を含むほとんどの OS で、NIC のプロミスクラスモードは、高い権限がないと使用できません。プロミスクラスモードを使用するための権限が合法的に取得できないなら、そのネットワークでパケットキャプチャを行うべきではありません

2.2 ハブで構成されたネットワークでのキャプチャ

ハブで構成されたネットワークでのキャプチャはとても簡単です。1章ですでに学んだとおり、トラフィックはハブのすべてのポートに流れます。したがって、ハブに接続されている機器のトラフィックを解析するためには、ハブの空いているポートにパケットキャプチャツールがインストールされた機器を接続するだけでよいのです。その機器宛、またその機器発のすべての通信だけでなく、そのハブに接続されたすべての機器の通信も参照できるようになります。図2-2にあるように、パケットキャプチャツールは、ハブに接続されているすべての機器の通信を参照できます。

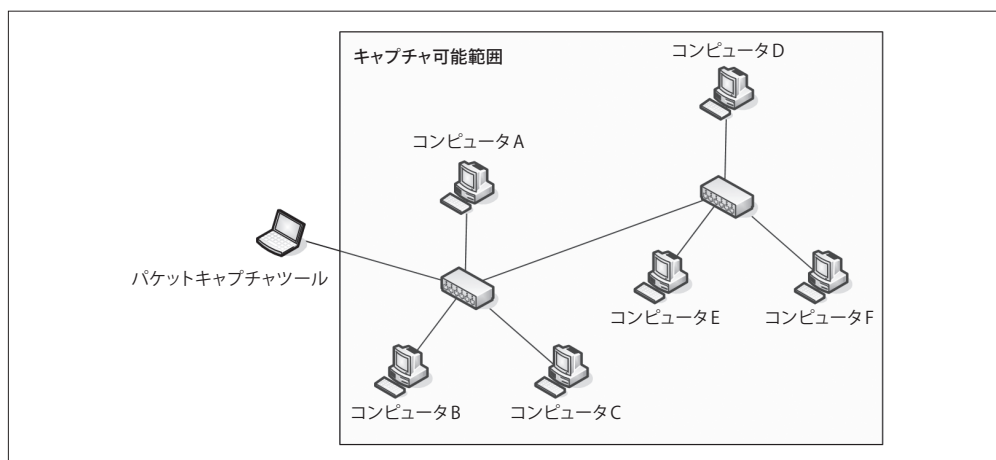


図2-2 ハブで構成されたネットワークでは、すべてを参照できる



残念ながら、ハブで構成されたネットワークはネットワーク管理者の頭痛の種となるため、今ではほとんど使われていません。ハブは一度に1つの通信しか扱えないため、ハブを通して接続されている機器は、通信しようとしているほかの機器と帯域を取り合うことになります。2台以上の機器が同時に通信しようすると、図2-3に示すようなパケットの衝突が発生し、結果としてパケットが消失します。機器はパケットを再送して消失を補おうとするため、ネットワークが輻輳し、さらなる衝突が発生します。トラフィックが増加して衝突が増え、機器はパケットを3回4回と再送信しなければならぬため、ネットワークのパフォーマンスが劇的に落ちてしまいます。現在のネットワークがハブでなくスイッチを使用している理由はそこにあります。

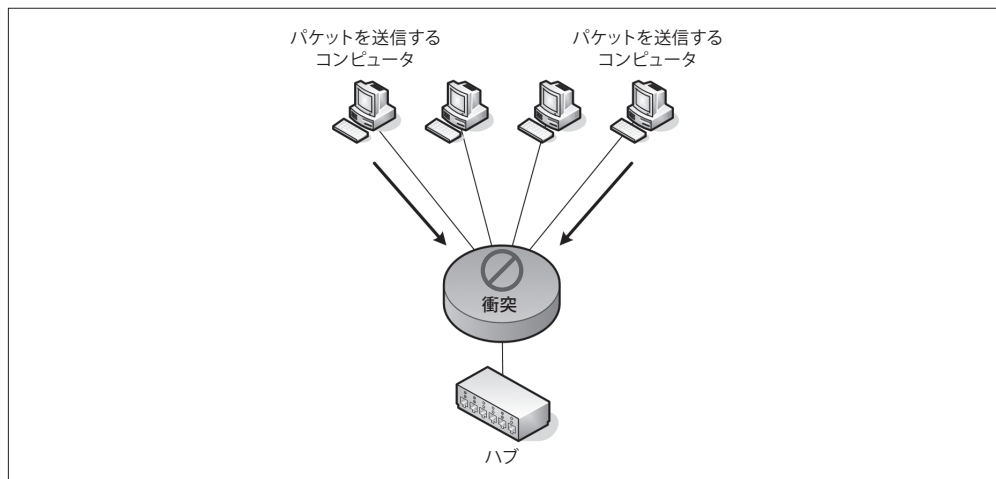


図2-3 2台の機器が同時にパケットを送信すると、パケットの衝突が起こる

2.3 スイッチで構成されたネットワークでのキャプチャ

1章で説明したように、現在のネットワーク環境で一番よく使われているネットワーク機器はスイッチです。スイッチはブロードキャスト、ユニキャスト、マルチキャストのデータを効率的に転送します。さらに、スイッチは全二重通信が可能なため、データの送信と受信を同時に行えます。一方、スイッチで構成されたネットワーク上でのキャプチャは、ハブで構成されたネットワークほど単純ではありません。図2-4に示すとおり、スイッチに接続されたパケットキャプチャツールは、ブロードキャストパケットと、ツールがインストールされている機器宛のパケットしか見ることができないのです。

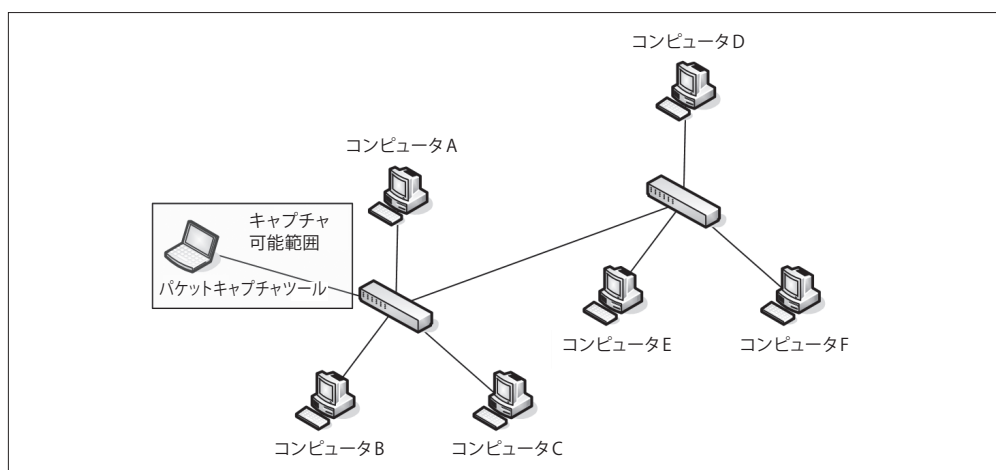


図2-4 スイッチで構成されたネットワークでは、パケットキャプチャツールがインストールされている機器が接続されているポートしかキャプチャ可能範囲とならない

スイッチで構成されたネットワークで自分宛以外の機器の通信をキャプチャする主な方法としては、ポートミラーリング、ハブの使用、タップの使用、ARPキャッシュポイズニングの4つがあります。

2.3.1 ポートミラーリング

ポートミラーリング（ポートスパニング）は、スイッチで構成されたネットワークでキャプチャ対象の機器のパケットをキャプチャする一番簡単な方法です。ポートミラーリングを使用するには、キャプチャ対象の機器が接続されているスイッチを、コマンドまたはWebマネジメントインターフェイスを使って操作する必要があります。加えて、スイッチがポートミラーリングをサポートしていること、そのスイッチに、キャプチャ用の機器を接続するための空きポートがあることも必要です。

ポートミラーリングを使用するには、スイッチのコマンドラインインターフェイスを使い、特定のポートの通信を他のポートにコピー（ミラーリング）するようなコマンドを入力する必要があります。たとえば、ポート3のパケットをキャプチャする際は、キャプチャ用コンピュータをポート4に接続し、ポート3からポート4にミラーリングする設定を行うことで、対象の機器の通信を見ることができるようになります。図2-5はポートミラーリングを示したものです。

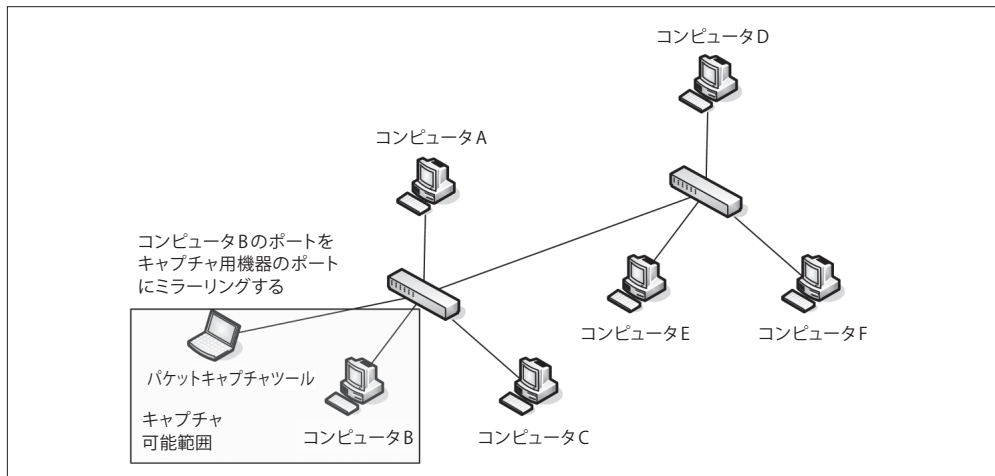


図2-5 ポートミラーリングによって、キャプチャ可能範囲を拡大できる

ポートミラーリングのための設定は、スイッチのベンダーによって異なります。ほとんどのスイッチでは、コマンドラインインターフェイスにログインし、ポートミラーリングコマンドを入力する必要があります。表2-1は、一般的なポートミラーリングコマンドの一覧です。



ポートミラーリングのオプションをWebベースのGUIから設定できるスイッチもありますが、あまり一般的ではなく、標準化もされていません。しかし、GUIを使ってポートミラーリングを効率的に設定できるなら、使わない手はありません。

表2-1 ポートミラーリングを有効化するコマンド

メーカー	コマンド
Cisco	set span <ミラーリング元ポート> <ミラーリング先ポート>
Enterasys	set port mirroring create <ミラーリング元ポート> <ミラーリング先ポート>
Nortel	port-mirroring mode mirror-port <ミラーリング元ポート> monitor-port <ミラーリング先ポート>

ポートミラーリングを使用するときは、ミラーリングしているポートのスループットに注意してください。スイッチの中には、2台以上の機器の通信を同時に解析できるようにするため、複数のポートを1つのポートにミラーリングできるものがあります。しかしながら、たとえば24ポートのスイッチで、100Mbps、全二重で通信する23ポートの通信を1つのポートにミラーリングした場合を考えてみてください。4,600Mbpsものトラフィックが1つのポートに流れるかもしれません。そうなれば、トラフィックが単一ポートの物理的な限界を超えた時点で、パケットの消失やネットワークの遅延を引き起こすことになります。こうした状況では、スイッチが処理しきれないパケットをすべて破棄したり、内部回路が「停止」してしまい、通信がまったくできなくなってしまうことがあります。パケットをキャプチャする際には、このような状況にならないように気をつけてください。

2.3.2 ハブの使用

スイッチで構成されたネットワーク上でパケットをキャプチャするもうひとつの方法は、ハブを使用することです。パケットをキャプチャしたい機器と解析用の機器を、ハブに接続することで同じネットワークセグメント上に置いてしまうのです。多くの人々は、そのようにハブを使用することは不正な行為だと思っていますが、ポートミラーリングが使えない環境で、かつキャプチャしたい機器が接続されているスイッチに物理的に触ることが可能であれば、ハブの使用はキャプチャを実現する完璧な方法と言えます。

ハブを使用して機器の通信をキャプチャするために必要なのは、ハブと数本のネットワークケーブルだけです。必要なものが揃ったら、次のように接続します。

1. スイッチのある場所に行き、キャプチャ対象の機器のケーブルを抜きます。
2. キャプチャ対象の機器のケーブルをハブにつなぎます。
3. キャプチャ用の機器とハブをケーブルでつなぎます。
4. ハブとスイッチをケーブルでつなぎます。

これで、キャプチャ用の機器とターゲットマシンが同じブロードキャストドメイン上に存在することになります。**図2-6**のように、キャプチャ対象の機器が送受信するパケットは、ハブに接続されているすべての機器にブロードキャストされることになり、パケットキャプチャツールがパケットをキャプチャできるようになります。

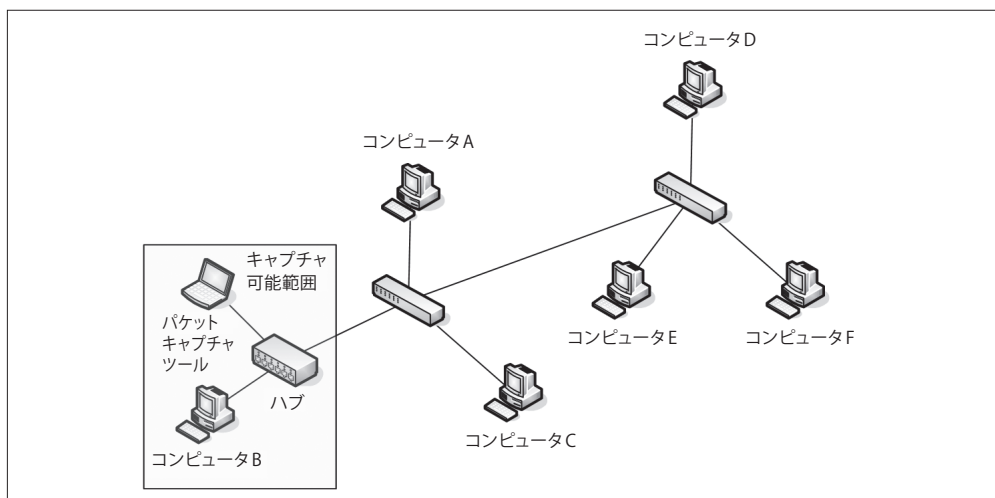


図2-6 ハブを使うことで、キャプチャ対象の機器とキャプチャ用の機器を同じブロードキャストドメインに置くことができる

「本物の」ハブを見つける

ハブを使用する場合、インチキのラベルが貼られたスイッチではなく、本物のハブを使うよう気をつけてください。実際には性能の低いスイッチの機能しか持たないにも関わらず、ハブとして宣伝、販売するという悪習を持つメーカーが存在するからです。ハブを使わないと、キャプチャ対象の機器のパケットではなく、自分のパケットしか見ることはできません。

ハブを見つけたら、それが本物のハブかどうかをテストしましょう。本物であればラッキーです。本物のハブかどうかを判断する最良の方法は、2台の機器に接続して、片方の機器がもう1台との通信をキャプチャできるかどうか、またネットワーク上のほかの機器やプリンタなどの機器との通信をキャプチャできるかどうかを確認することです。それができれば、本物のハブだということになります。

ハブは時代遅れなため、現在ではあまり製造されていません。店頭で本物のハブを購入するのはほとんど不可能なため、少々努力が必要です。地元の学校での中古品オークションは貴重な機会です。公立学校は廃棄処分する前にオークションを行うことが義務付けられているうえに、古い機器が放置されていることがよくあります。白い豆とコーンブレッドよりも安い値段で、数台のハブを中古オークションで入手した人々を知っています。またeBayも良い入手先ですが、ハブだと偽ったスイッチをつかまされることがあるので、注意が必要です[†]。

[†] 監訳注：国内では、秋葉原などの中古ショップを注意深く探さないと、中古のハブに巡り合うことは難しくなっていました。商品説明にリピータハブ、シェアードハブなどと明記してあれば大丈夫でしょう。

たいていの場合、ハブを使うと全二重の通信が半二重になります。ハブの使用はケーブルに潜入する最良の方法とは言えませんが、ポートミラーリングをサポートしていない場合は、この方法を使うしかありません。ただしハブには電源が必要であり、これを見つけるのが難しい場合もあることを覚えておいてください。



ついでに言っておくと、ケーブルを抜こうとしている機器の使用者に、きちんと警告したという事実を作っておくことをお勧めします。使用者が会社のCEOだったりする場合はなおさらです。

2.3.3 タップを使う

「ステーキがあるのに、どうしてチキンを食べるのか」という言い回しはよく知られています（南部出身なら「揚げポローニャがあるのに、どうしてハムなんだ」と言うかもしれませんが）。この選択は、タップの使用とハブの使用にも当てはまります。

ネットワークタップ（タップ）は、2つの通信機器の間に設置して、この2点間のパケットをキャプチャするためのハードウェアです。ハブの場合、ネットワーク内にハードウェアを設置して、必要なパケットをキャプチャします。ハブとの違いは、ネットワーク解析のために特別に設計されたハードウェアを使うという点です。

ネットワークタップには、主に**統合型**と**非統合型**の2種類があります。どちらのタップも2台の機器の間に設置され、通信をキャプチャします。統合型タップと非統合型タップの主な違いは、非統合型には図2-7で示したように4つのポートがあるのに対し、統合型には3つしかないことです。



図2-7 Barracuda非統合型タップ

タップには通常電源が必要ですが、なかにはコンセントにつながなくても、短時間であればパケットキャプチャが可能な電池内蔵型もあります。

2.3.3.1 統合型タップ

統合型タップの使い方は簡単です。双方向の通信をキャプチャする物理的な監視ポートが1つしかないからです。

統合型タップを使って、スイッチに接続された1台の機器のすべての通信をキャプチャするには、次のように行います。

1. 機器をスイッチから外します。
2. ネットワークケーブルの一方を機器につなぎ、もう一方をタップの「入力」ポートにつなぎます。
3. 別のケーブルの一方をタップの「出力」ポートにつなぎ、もう一方をスイッチにつなぎます。
4. 最後のケーブルの一方をタップの「監視」ポートにつなぎ、もう一方をパケットキャプチャツールとして機能する機器につなぎます。

統合型タップは図2-8のように接続されているはずです。この時点で、パケットキャプチャツールはタップに接続した機器のすべての通信をキャプチャすることになります。

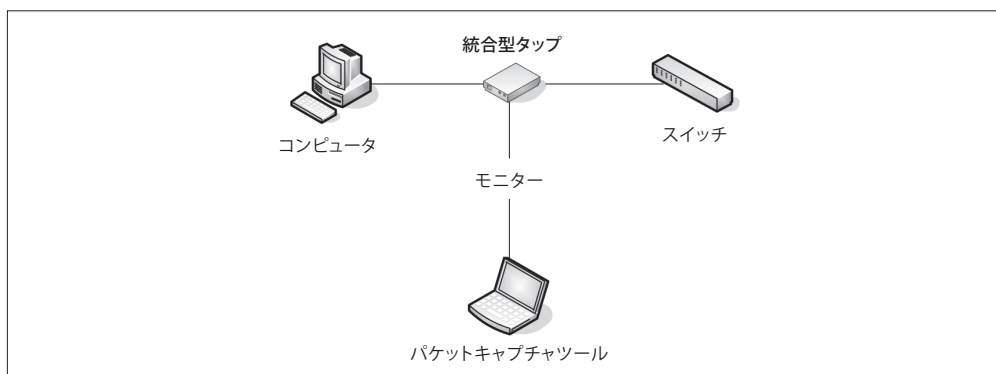


図2-8 統合型タップを使ってネットワークトラフィックをキャプチャする

2.3.3.2 非統合型タップ

非統合型タップは統合型よりも少し複雑ですが、通信のキャプチャという点ではより柔軟性があります。双方向通信の監視に使うポートが1つだけあるのではなく、ポートが2つあるからです。片方の監視ポートを一方の通信のキャプチャに用い（タップに接続されたコンピュータからの通信）、もう一方を逆方向の通信（機器に入っていく通信）に使うことができます。

スイッチに接続された1台のコンピュータのすべての通信をキャプチャするには、次のように行います。

1. 機器をスイッチから外します。
2. ネットワークケーブルの一方を機器につなぎ、もう一方をタップの「入力」ポートにつなぎます。
3. 別のケーブルの一方をタップの「出力」ポートにつなぎ、もう一方をネットワークスイッチにつなぎます。
4. 3本目のケーブルの一方をタップの「監視A」ポートにつなぎ、もう一方をパケットキャプチャツールとして機能する機器のNICにつなぎます。

5. 最後のケーブルの一方を「監視B」ポートにつなぎ、もう一方をパケットキャプチャツールとして機能する機器のもうひとつのNICにつなぎます。

非統合型タップは図2-9のように接続されているはずです。

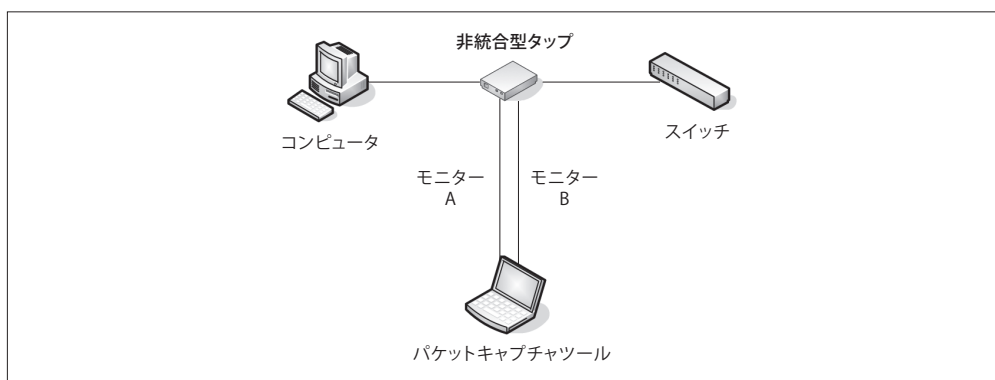


図2-9 非統合型タップを使ってネットワークトラフィックをキャプチャする

2.3.3.3 ネットワークタップを選ぶ

2種類のタップの違いを説明しましたが、どちらがよいのでしょうか。多くの場合、必要なケーブル本数が少なく、パケットキャプチャツールとして機能する機器に2つのNICを必要としない統合型が好まれます。しかし大量の通信をキャプチャする場合や、一方向の通信のみをキャプチャする場合は、非統合型タップがよいでしょう。

150ドル程度のシンプルなイーサネットタップから、5桁の価格となる企業クラスの光ファイバタップまで、多種多様なタップが購入できます。筆者はNet OpticsとBarracuda Networksのタップを使っていますが、非常に満足しています。ほかにも優れたタップが数多くあると思います。

2.3.4 ARPキャッシュポイズニング

ケーブルに潜入するテクニックのなかでもお気に入りのひとつが、ARPキャッシュポイズニングです。ARPプロトコルの詳細については6章で説明します。ここでは、このテクニックの動作を理解するために最低限必要な説明を行います。

2.3.4.1 ARP処理

1章では、パケットのアドレスには第3層のものと第2層のものという2種類があることを学びました。第2層のアドレス（MACアドレス）は、第3層のアドレスと連携して使われます。本書では、業界標準の用語にしたがって、第3層のアドレスをIPアドレスと呼びます。

第3層のネットワーク機器はすべて、IPアドレスを使って通信を行います。スイッチはOSI参照モデルの第2層で動作し、第2層のMACアドレスしか認識しないため、機器が生成するパケットには

MACアドレスの情報も格納する必要があります。適切な機器にパケットを転送するために、MACアドレスが不明な場合は、既知の第3層のIPアドレスを使ってこれを取得しなければなりません。この変換のプロセスは、ARP (Address Resolution Protocol) と呼ばれる第2層のプロトコルによって行われます。

ARP処理は、イーサネットネットワークに接続している1台の機器が別の機器と通信しようとしたときに開始されます。送信元の機器はまずARPキャッシュを確認し、宛先の機器のIPアドレスに対応するMACアドレスが格納済みでないかを確認します。格納されていなければ、1章で説明したように、データリンク層のブロードキャストアドレスFF:FF:FF:FF:FF:FFに、ARPリクエストを送ります。このパケットはブロードキャストパケットであるため、特定のイーサネットセグメントにあるすべての機器が受信することになります。パケットは「MACアドレスXX:XX:XX:XX:XX:XXを所有するIPアドレスは?」と質問しているわけです。

このIPアドレスを所有しない機器はARPリクエストを単に破棄します。宛先になる機器は、ARPレスポンスによってMACアドレスを返答します。元々の送信元の機器は、これによって宛先の機器との通信に必要なデータリンク層の情報を取得し、素早く検索できるよう、ARPキャッシュにこの情報を格納します。

2.3.4.2 ARPキャッシュポイズニングの動作

ARPキャッシュポイズニングは、ARPスプーフィングとも呼ばれ、偽りのMAC (第2層) アドレスを含むARPメッセージをイーサネットのスイッチまたはルータに送信し、別の機器のトラフィックに割り込む処理です。図2-10はこの動作を図示したものになります。

ARPキャッシュポイズニングは、スイッチで構成されたネットワークに潜入する高度な手法です。これは通常、通信への割り込みやDoS攻撃 (Denial of Service attack: サービス不能攻撃) を仕掛ける目的で、偽装したアドレスのパケットをホストやネットワーク機器に送信するために、攻撃者によって使用されます。しかしながら、スイッチで構成されたネットワーク上で、特定の機器のパケットをキャプチャする手段としても使うことができます。

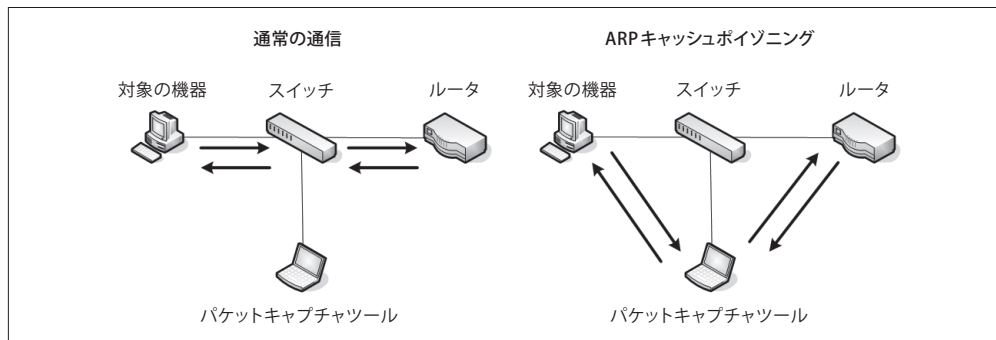


図2-10 ARPキャッシュポイズニングによって、対象の機器の通信に割り込む

2.3.4.3 Cain & Abelの使用

ARPキャッシュポイズニングを使用するには、まず必要なツールを入手し、いくつかの情報を確認しておく必要があります。ここでは、有名なセキュリティツールであり、Windowsに対応しているOxid.it (<http://www.oxid.it/>) のCain & Abelを使います。Webサイトの説明に従ってダウンロードし、インストールしてください[†]。

Cain & Abelをインストールする前に、解析用ホストのIPアドレス、トラフィックをキャプチャしたい対象の機器、その機器が接続されているルータの情報などを確認しておく必要があります。

Cain & Abelを起動すると、ウィンドウの上部にいくつかのタブが確認できるはずです (ARPキャッシュポイズニングは、Cain & Abelの多くの機能のひとつにすぎません)。ARPキャッシュポイズニングは、[Sniffer] タブから実行することができます。[Sniffer] タブをクリックすると、図2-11のような空の表が表示されるはずです。

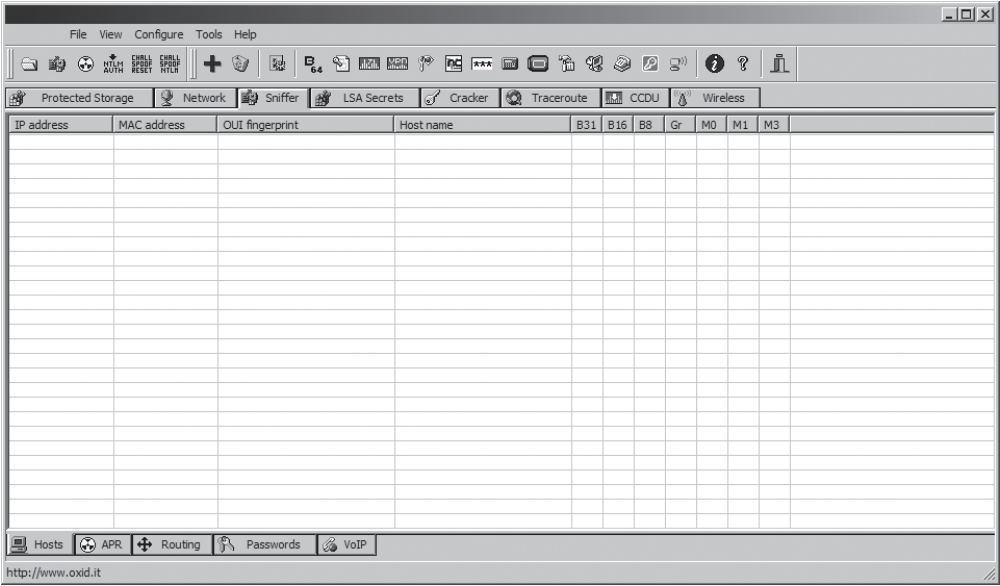


図2-11 Cain & Abelの [Sniffer] タブ

この表に情報を格納させるため、Cain & Abelの内蔵パケットキャプチャツールを起動して、ネットワーク上のホストをスキャンする必要があります。これは以下の手順で行います。

- 1. ツールバーの左から2番目、NICのアイコンに似たアイコンをクリックします。
- 2. キャプチャに用いるインターフェイスを聞かれます。ARPキャッシュポイズニングを行いたいネットワークに接続しているインターフェイスを選択してください。選択したら [OK] ボタンを

[†] 監訳注：Cain & Abelはパスワードクラッキングツールであるため、ダウンロードがブロックされ、企業内ネットワークの場合はネットワーク管理者にアラートが上がったりする場合がありますので、注意してください。

クリックします (Cain & Abelのパケットキャプチャ機能を有効にするため、必ずこのボタンを押してください)。

3. [+] アイコンをクリックし、ネットワーク上のホスト一覧を作成します。すると図2-12のような [MAC Address Scanner] ダイアログが表示されるので、[All hosts in my subnet] ラジオボタンを選択して (あるいは適宜アドレス範囲を指定して)、[OK] ボタンをクリックします。

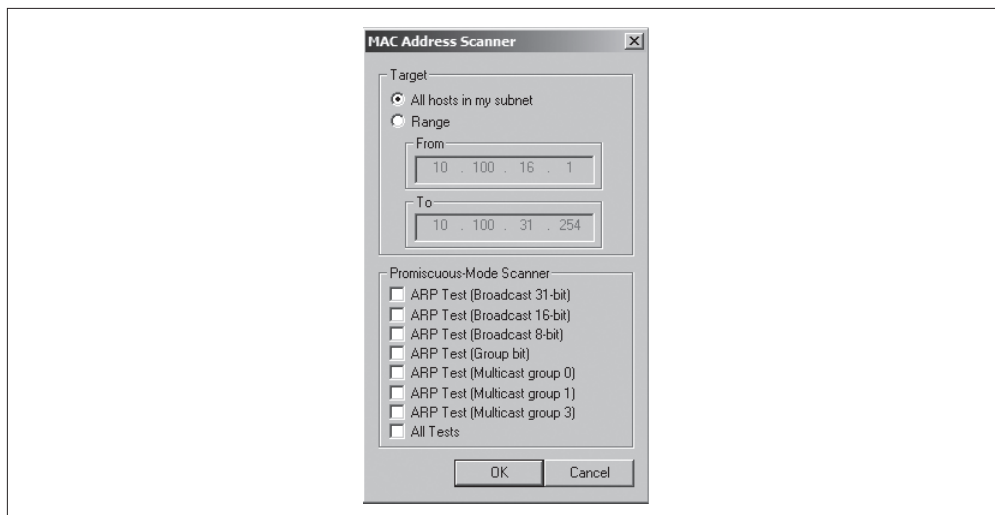


図2-12 Cain & Abelのネットワーク検索ツール

これで表には、ネットワークに接続されているホストのMACアドレス、IPアドレス、ベンダー情報などの一覧が表示されるはずです。これらの情報を利用して、ARPキャッシュポイズニングを実行します。

ウィンドウの下部には、[Sniffer] タブで利用できる機能のタブが表示されているはずです。ホスト一覧を作成したら、[APR] (ARP Poison Routing) タブでの作業に移ります。タブをクリックして、[APR] ウィンドウへ切り替えます。

[APR] ウィンドウでは、2つの空の表が表示されます。以降の設定を完了すると、上の表にはARPキャッシュポイズニングが行われた機器の情報が、下の表にはARPキャッシュポイズニングを行っている機器を経由して通信しているすべての機器の情報が表示されます。

ARPキャッシュポイズニングの設定手順は以下のとおりです。

1. 上の表の空白部分をクリックし、次にツールバー上にある [+] ボタンをクリックします。
2. 表を2つ横に並べたダイアログが表示され、左側の表に、ARPキャッシュポイズニングによる通信の割り込みが可能なホストの一覧が表示されます。トラフィックをキャプチャする対象となる標的ホストのIPアドレスをクリックすると、右側の表には残りのホストの一覧が表示されます。

3. 右側の表で、標的ホストの上位にあたるルータのIPアドレスをクリックし、[OK] ボタンをクリックします (図2-13)。これで、標的ホストとルータのIPアドレスが、[APR] ウィンドウの上の表に表示されます。
4. 最後に、ツールバー上にある黄色と黒の放射能マーク (☢) をクリックします。これで、Cain & AbelのARPキャッシュポイズニング機能が有効になり、標的ホストとルータとの間の通信に割り込むことができるようになります。

これで、パケットキャプチャツールを使って通信を解析できるようになりました。パケットのキャプチャが終了したら、黄色と黒の放射能マーク (☢) を再度クリックすることで、ARPキャッシュポイズニングを停止できます。

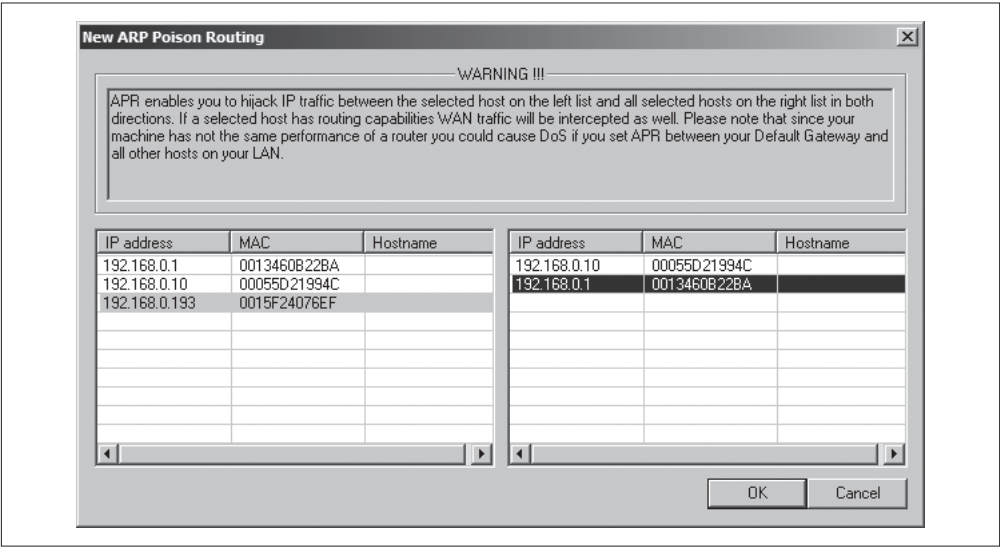


図2-13 ARPキャッシュポイズニングを有効にする機器を選択する

2.3.4.4 ARPキャッシュポイズニングを使う際の注意

ARPキャッシュポイズニングは、標的ホストの役割に注意して実行してください。たとえば、標的とする機器が1Gbpsでネットワークに接続されているファイルサーバのような、ネットワーク的に高負荷を発生させている機器の場合、この方法を用いるべきではありません (特に解析用ホストが100Mbpsしか使えない場合)。

このような状況でARPキャッシュポイズニングにより通信の割り込みを行うと、標的ホストが送受信するトラフィックのすべてが最初に解析用ホストに流れ込んでしまうため、そこが通信処理のボトルネックになってしまいます。これでは、解析用ホストがDoS攻撃を受けているのと同じ状態になり、ネットワークのパフォーマンスを下げるだけでなく、解析もうまくいかないでしょう。



非対称ルーティングという機能を使えば、解析用ホストにすべてのパケットが流れ込む事態を回避することができます。この方法についての詳しい情報は、oxid.itのドキュメント (http://www.oxid.it/ca_um/topics/apr.htm) を参照してください。

2.4 ルータで構成されたネットワークでのキャプチャ

スイッチで構成されたネットワークでネットワークに潜入する方法は、そのままルータで構成されたネットワークでも使えます。ルータを含むネットワーク特有の考慮点は、複数のネットワークセグメントにまたがるトラブルを解決する際のパケットキャプチャツール設置の重要性です。

すでに学んだとおり、ブロードキャストドメインはルータによって区切られます。ルータは、トラフィックを次のルータへと中継します。データが複数のルータを経由するような環境では、ルータの前後でトラフィックを解析することが重要です。

ルータをいくつか経由して、いくつかのネットワークセグメントが接続されているネットワークで発生した通信トラブルについて考えてみましょう。このネットワークでは、各セグメントがデータの格納、参照を行うために上位のセグメントと通信します。図2-14のように、下位のサブネットであるネットワークDからネットワークAのホストと通信できないというトラブルが発生したとしましょう。

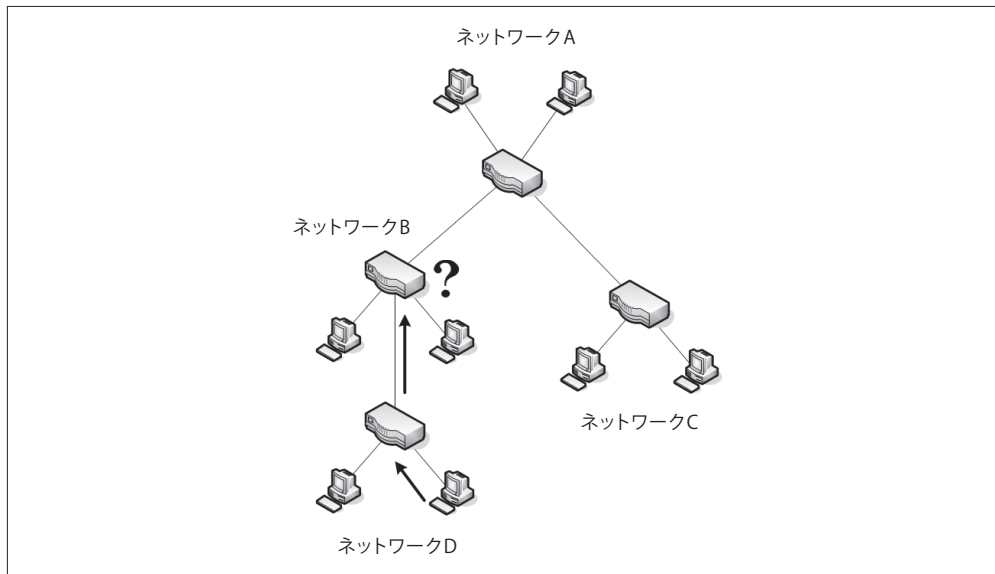


図2-14 ネットワークD上のホストがネットワークAのホストと通信できない

ほかのネットワークのホストと通信できないトラブルを抱えているネットワークD上のホストのトラフィックをキャプチャすると、ほかのネットワークへ送信されるデータはきちんと確認できるにもか

わらず、そこから返ってくるはずのデータを確認できなかったとします。この場合、パケットキャプチャツールの設置場所を再考し、上位のネットワークセグメント（ネットワークB）でキャプチャを始めてみると、何が起きているかがはっきりするはずです。ここでは、ネットワークBのルータがパケットを破棄しているか、ルーティングを誤っていたことにしましょう。これでトラブルの原因がルータの設定にあることがわかり、設定を治せば悩みは解決です。このシナリオは少々おおざっぱではありますが、ここで言いたかったことは、複数のルータやネットワークセグメントが存在する場合、全体像を把握するには、パケットキャプチャツールを何度か移動させる必要があるかもしれないということです。

この例からも、トラブルを特定するためには、複数セグメント上の複数の機器のトラフィックをキャプチャする必要がある場合が多い理由がわかると思います。

ネットワークマップ

ここまでの設置場所についての説明の中で、いくつかのネットワークマップを見てきました。ネットワークマップ（ネットワーク構成図）には、ネットワーク上のホストやネットワーク機器がどのように接続されているかが示されています。

パケットキャプチャツールの設置場所を決定するには、ネットワークを視覚化するのが一番です。すでにネットワークマップがあるなら、トラブルシューティングや解析の際非常に役立ちますので、常に手元に置いておきましょう。もっと詳細なネットワークマップを作成したいと感じるかもしれません。トラブルシューティング作業の半分以上が、集めたデータの正当性の確認であることもよくあります。

2.5 パケットキャプチャツールを実際に設置する

スイッチで構成された環境におけるパケットのキャプチャについて、4つの方法を見てきました。これ以外に1台のホストにパケットキャプチャツールのアプリケーションをインストールするだけ（直接インストール方法）という方法もあります。5つもあると、どの方法が最適かを判断するのに少々悩んでしまうかもしれません。表2-2は各方法についての汎用的なガイドラインを示しています。

表2-2 スイッチで構成された環境におけるキャプチャ方法のガイドライン

方法	ガイドライン
ポートミラーリング	<ul style="list-style-type: none">ネットワークに影響を与えず余分なパケットが生成されないため、通常好まれるホストをネットワークから切断せずに設定可能なため、ルータやサーバのポートをミラーリングする場合に便利
ハブの使用	<ul style="list-style-type: none">ホストを一時的にネットワークから切断しても問題ない場合に理想的な方法複数のホストからのパケットをキャプチャする場合は、衝突やパケットの消失が起こる可能性があり、不向き本物のハブは通常10Mbpsであるため、現在の100/1000Mbps環境ではパケット消失の恐れあり
タップの使用	<ul style="list-style-type: none">ホストを一時的にネットワークから切断しても問題ない場合に理想的な方法光ファイバ接続の場合はタップしか使えないタップは現代の高速ネットワークに対応するよう設計されているため、ハブよりも優れている予算が厳しい場合にはコスト的に困難
ARP キャッシュポイゾニング	<ul style="list-style-type: none">キャプチャするホストを経由するようにトラフィックの流れを変更するため、ネットワークに余分なパケットが流れてしまい、好ましくないポートミラーリングが使えない場合、ホストをネットワークから切断せずにパケットを即座にキャプチャする必要がある際に有効
直接インストール	<ul style="list-style-type: none">ホストに問題がある場合、パケットが消失したり変更されてしまったりする可能性があるため推奨されないホストのNICをプロミスキャスモードにする必要がないテスト環境、性能の確認や基準値の測定、他で作成されたキャプチャファイルの検査には最適

キャプチャは、可能な限り透過的でなければなりません。痕跡を一切残さずにデータを収集するのが理想です。法医学の検査官が犯行現場に手を加えたくないのと同様、私たちもキャプチャしたパケットに手を加えたくないのです。

あとの章では、実践的なシナリオを通じて、ケースバイケースでデータの最適なキャプチャ方法を解説していきます。当面は、パケットのキャプチャに使用する最適な方法を決めるのに、図2-15のフローチャートが役立つでしょう。このフローチャートは一般的なものであり、ネットワークに潜入する際の選択肢のすべてを網羅しているわけではないことは留意しておいてください。

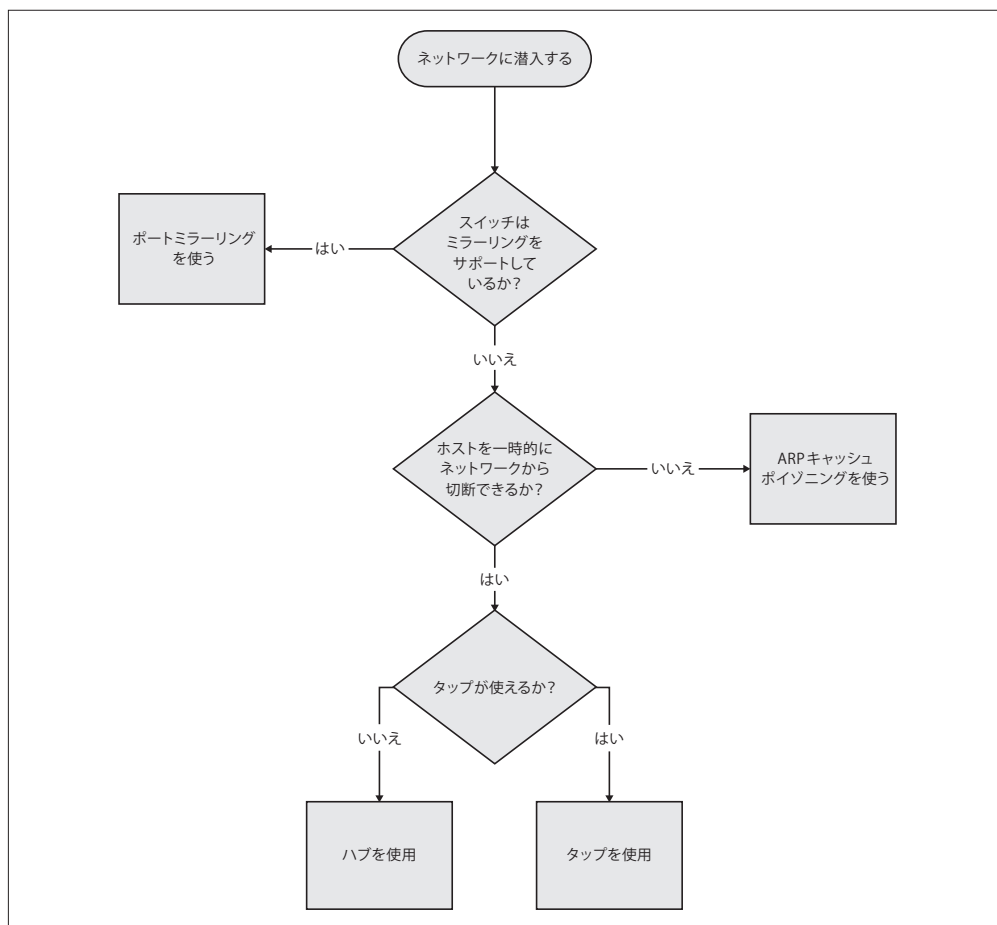


図2-15 ネットワークに潜入するための最適な方法を決定するフローチャート

3章

Wireshark 概要

1章で触れたように、パケット解析に使うパケットキャプチャツールにはさまざまな種類がありますが、本書ではWiresharkを取り上げています。この章ではWiresharkの概要を説明します。

3.1 Wiresharkの歴史

Wiresharkには長い歴史があります。Wiresharkはカンザスシティにあるミズーリ大学でコンピュータサイエンスを学んだジェラルド・ コームズ (Gerald Combs) が必要に迫られて開発したのが始まりです。最初のバージョンは、1998年にGPL (GNU Public License) によりEtherealという名前で公開されました。

Etherealが公開されてから8年後、コームズは新たなキャリアを求めてそれまで勤めていた企業を退職しました。残念ながら退職先の企業がEtherealの商標を持っていたため、コームズは契約上Etherealブランドを用いることができなくなってしまいました。代わりに、コームズとEtherealの開発チームは、2006年半ばにWiresharkという新たな商標でプロジェクトを再開しました。

Wiresharkは劇的な成長を遂げ、開発には500人もの人がかかわっています。Etherealという名前のプログラムはもう開発されていません。

3.2 Wiresharkの利点

Wiresharkには、日々のパケット解析に便利な機能がいくつもあります。1章で述べたパケットキャプチャツールの評価項目に従ってWiresharkを評価してみましょう。

サポートされているプロトコル

現在WiresharkはIPやDHCPのような一般的なものから、AppleTalkやBitTorrentのような特定のメーカーやソフトウェアでしか使われないものまで、本書執筆時点で850以上のプロトコルをサポートしています。Wiresharkはオープンソースモデルとして開発されており、Wiresharkが更新されるたびに新しいプロトコルが追加されています。



滅多にないことですが、Wiresharkがサポートしていないプロトコルを必要としている場合、自分でそのプロトコルをサポートするコードを書いてWiresharkの開発者に提供し、同梱してもらうこともできます(もちろんそのコードが承認されればですが)。

操作性

Wiresharkはほかのパケットキャプチャツールと比較しても遜色ないインターフェイスが備わっており、見やすいコンテキストメニューとレイアウトを備えたGUIベースのアプリケーションです。プロトコルごとの色分けや生データのグラフィカルな表示といった、操作性を向上させる機能もいくつか備わっています。tcpdumpのような難解なコマンドラインインターフェイスの代替アプリケーションと違い、Wiresharkはパケット解析を始めようという人にとって使いやすいツールとなっています。

コスト

Wiresharkはオープンソースで、GPLライセンスのもと無償で入手することができます。個人利用、商用利用を問わず、誰でもWiresharkをダウンロードして任意の目的で使うことができます。



Wiresharkは無償ですが、間違ってお金を払っている人もいます。eBayでパケットキャプチャツールを検索すると、Wiresharkの「プロフェッショナル企業ライセンス」を、39.95ドルという低価格で売りつけようとしている人がたくさんいることに驚くでしょう。もちろんこれは茶番ですが、本当に買いたいなら、私に電話をください。私がケンタッキーで売りに出しているビーチ沿いの物件について話をしましょう†。

サポート

ソフトウェアの善し悪しはそのサポートによって決まると言っても過言ではありません。Wiresharkのようなフリーで公開されているソフトウェアに公式サポートがあるとは限りません。オープンソースのソフトウェアのサポートがユーザー頼みであることが多いのはそのためです。幸運なことに、Wiresharkのユーザーコミュニティは、オープンソースのプロジェクトの中でも非常に活発です。WiresharkのWebページには、オンラインドキュメント、開発者のためのWiki、FAQ、主要な開発者も参加しているメーリングリストに登録するための方法といったリンクが載っています。CACE TechnologiesのSharkNetプログラムを利用すれば、有償サポートを受けることもできます。

OSのサポート

WiresharkはWindows、Mac OS X、Linuxベースのプラットフォームなど、現在主要なOS

† 監訳注：ケンタッキーは内陸の州なので、「ビーチ沿いの物件」は存在しません。

のすべてをサポートしています。サポートしている OS の一覧は、Wireshark の Web ページで見ることができます。

3.3 Wiresharkのインストール

Wireshark のインストールは驚くほど簡単です。ただし、インストール前に以下のシステム要件を満たしているかどうかを確認しておいてください。

- 400MHz 以上の CPU
- 128MB の RAM
- 75MB 以上のディスク領域
- プロミスキヤスモードをサポートしている NIC
- WinPcap キャプチャドライバ

WinPcap キャプチャドライバは、pcap というパケットキャプチャ API の Windows 版です。単純にインストールするだけで、このドライバは OS とやり取りして生のパケットデータをキャプチャしたり、フィルタを適用したり、NIC をプロミスキヤスモードに切り替えたりしてくれます。

WinPcap は (<http://www.winpcap.org/> から) 個別にダウンロードすることもできますが、通常は Wireshark のパッケージからインストールすることをお勧めします。パッケージに同梱されている WinPcap は Wireshark での動作が確認されたバージョンです。

3.3.1 Windowsでのインストール

Wireshark を Windows にインストールする最初の一步は、Wireshark の Web ページ (<http://www.wireshark.org/>) から、最新版の Wireshark を入手することです。Web サイトの「Download」へと進み、ミラーサイトを選択してください[†]。パッケージをダウンロードしたら、以下の手順でインストールしてください。

1. .exe ファイルをダブルクリックし、表示されたダイアログで [Next] ボタンをクリックします。
2. ライセンスを読み、同意するなら [I Agree] ボタンをクリックします。
3. 図 3-1 のダイアログでインストールする Wireshark のコンポーネントを選択します。ここでは何も変更せず [Next] ボタンをクリックします。

[†] 監訳注：本書監訳時点では、Web サイトの「Download Wireshark」へと進み、使用している OS に合致したパッケージを選択してください。

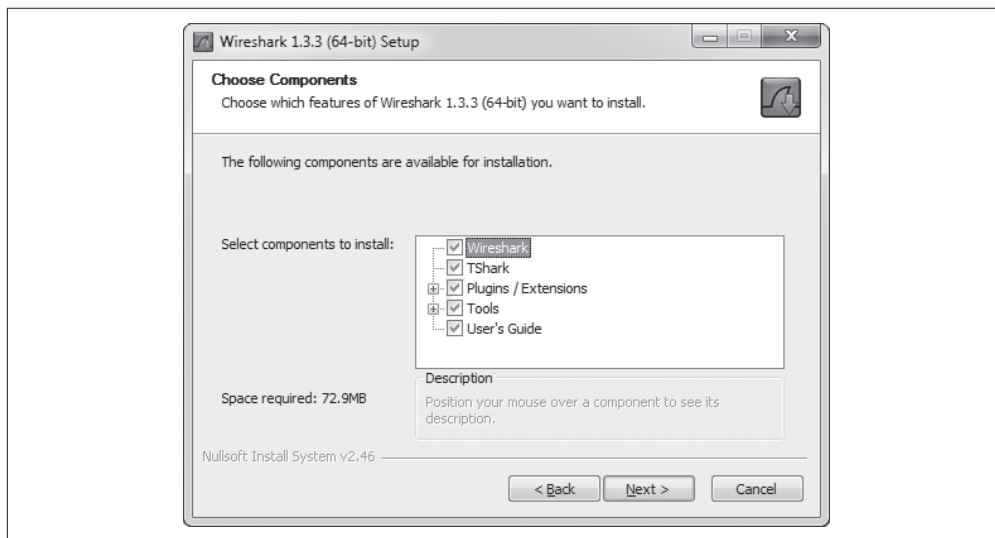


図3-1 インストールするコンポーネントを選択

4. [Additional Tasks] ダイアログでは [Next] ボタンをクリックします。
5. Wiresharkのインストール先を指定し、[Next] ボタンをクリックします。
6. WinPcapをインストールするかを尋ねるダイアログが表示されるので、[Install WinPcap] チェックボックスが図3-2のようにチェックされていることを確認し、[Install] ボタンをクリックしてインストールを開始します。

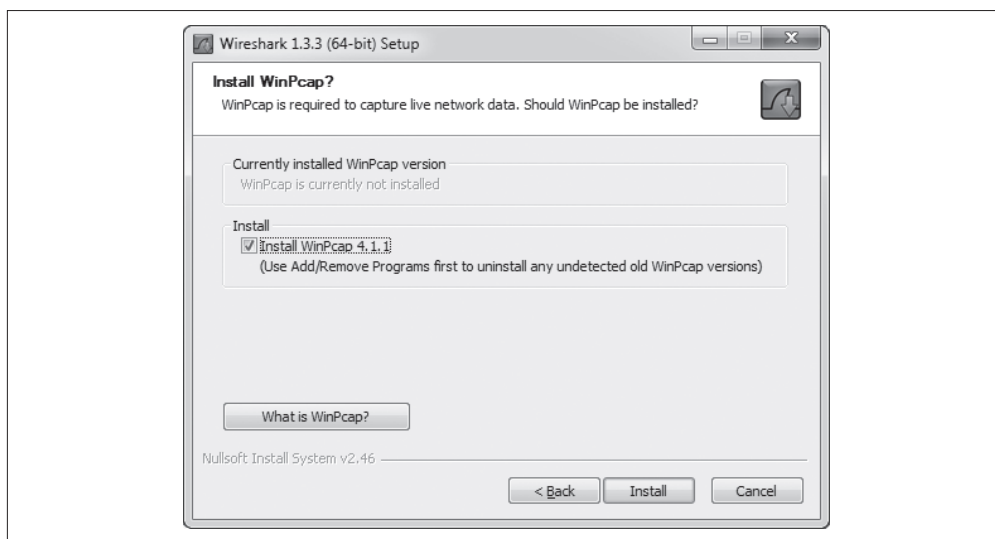


図3-2 WinPcapドライバのインストールを選択する

7. Wiresharkのインストールの途中で、WinPcapのインストールが始まります。継続を確認するダイアログが表示されるので [Next] ボタンをクリックし、ライセンスを読んでから [I Agree] ボタンをクリックしてください。
8. WinPcapがインストールされます。終了したら [Finish] ボタンをクリックします。
9. Wiresharkのインストールが継続されます。終了したら [Next] ボタンをクリックします。
10. インストール終了の確認ウィンドウで、[Finish] ボタンをクリックします。

3.3.2 Linuxでのインストール

WiresharkをLinuxにインストールする最初の一步は、適切なインストールパッケージのダウンロードです。ただし、すべてのLinuxディストリビューションがパッケージを提供しているわけではないので、パッケージがないからといって驚かないでください。

通常システムにソフトウェアをインストールする際には、root権限が必要となります。しかしソフトウェアをソースからコンパイルして独自にインストールする場合は、通常root権限なしでインストール可能です。

3.3.2.1 RPMベースのシステム

Red Hat Enterprise LinuxのようなRPMベースのディストリビューションでは、WiresharkのWebサイトから適切なインストールパッケージをダウンロードしたうえで、コンソールを開き、次のように入力します（ファイル名はダウンロードしたパッケージのものに適宜変えてください）[†]。

```
rpm -ivh wireshark-0.99.3.i386.rpm
```

依存するパッケージがインストールされていない場合は、それらをインストールしてから再度Wiresharkをインストールしてください。

3.3.2.2 DEBベースのシステム

DebianやUbuntuのようなDEBベースのディストリビューションでは、システムのリポジトリからWiresharkをインストールすることができます。コンソールを開き、次のように入力します。

```
apt-get install wireshark
```

3.3.2.3 ソースからのコンパイル

使っているLinuxディストリビューションにパッケージ管理機構がない場合、Wiresharkをインストールする最善の方法はソースからのコンパイルです。以下の手順を順に行っていきます。

1. WiresharkのWebサイトからソースパッケージをダウンロードします。

[†] 監訳注：最近のFedoraやCentOSなどのシステムではwireshark-gnomeパッケージをインストールしてください。wiresharkパッケージはコマンドライン版のパッケージとなっています。またRPMファイルを直接インストールするのではなく、yumを使ったほうがよいでしょう。

2. 次のように入力してアーカイブを展開します（ファイル名はダウンロードしたパッケージのものに適宜変えてください）。

```
tar -jxvf wireshark-1.2.2.tar.bz2
```

3. 新規に作成されたディレクトリにファイルが展開されますので、そこにcdします。
4. root権限を持つユーザーでコマンド./configureを実行して、使っているLinuxディストリビューションに適したビルドを行うためにソースを準備します。デフォルト以外のインストール設定をしたい場合は、ここでオプションを指定します。依存するライブラリなどが見つからない場合はおそらくエラーが発生するでしょう。問題がなければ、図3-3のような成功を示すメッセージが表示されるはずです。

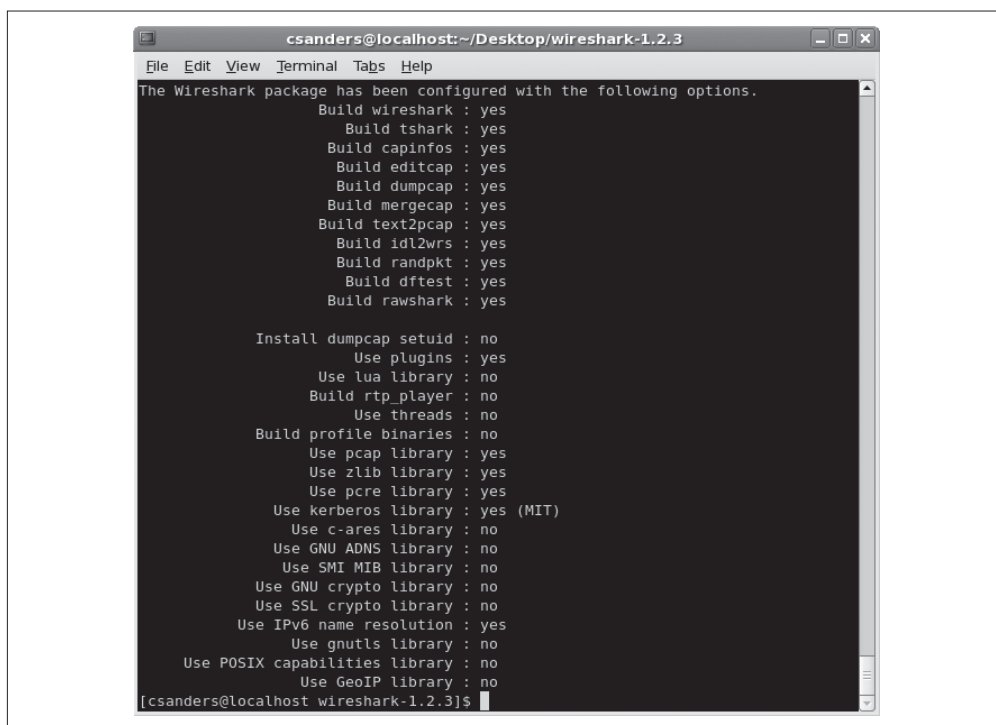


図3-3 ./configureコマンドが成功した場合の表示

5. make コマンドを入力し、ソースからバイナリをビルドします。
6. make install コマンドで最終的なインストールを行います。

3.3.3 Mac OS Xでのインストール

Mac OS X Snow LeopardにWiresharkをインストールするには若干の注意が必要ですが、インス

ツールそのものは難しいものではありません。以下にインストールの手順を示します[†]。

1. WiresharkのWebサイトからDMGパッケージをダウンロードします。
2. Wireshark.appをApplicationsフォルダにコピーします。
3. Wireshark.appのUtilitiesフォルダを開きます。
4. Finderで[Go (移動)]をクリックし、[Go To Folder (フォルダへ移動)]を選択します。続けて、`/usr/local/bin/`と入力しディレクトリを開きます。
5. Command Line フォルダの中身を `/usr/local/bin/` にコピーします。これにはパスワードの入力が必要です。
6. Utilities フォルダの中のChmodBPF フォルダをStartupItems フォルダにコピーします。これを実行しインストールを完了するには、再度パスワードの入力が求められます。

3.4 Wiresharkの基本

Wiresharkを首尾よくインストールできたら、あとは使ってみるだけです。さっそく完全版のパケットキャプチャツールを起動して、パケットを見て……何も表示されません！

Wiresharkを起動しただけではあまり面白くありません。面白いものを見るには、データを収集する必要があります。

3.4.1 はじめてのパケットキャプチャ

Wiresharkでパケットのデータを解析するには、まずパケットをキャプチャしなければなりません。「ネットワークに障害がないのにどうやってパケットをキャプチャするのだろうか？」と疑問に思うかもしれません。

まず、ネットワークには、常に何らかの障害があります。疑うのならネットワークのユーザー全員にメールを送信して、何の問題もないかどうかを確認してみてください。

次に、パケット解析は、障害があるときにしか行わないものではありません。実際のところ、ネットワーク管理者はトラブルシューティング中より、障害のないときのネットワークの解析に時間を割いています。ネットワークのトラブルシューティングを効果的に行うためには、ネットワークが正常な状態のときの情報と比較するためのベースラインが必要なのです。たとえばパケットを解析してDHCPの障害を解決しようとする場合、きちんと動作しているときのDHCPトラフィックがどのように流れているかを理解しておく必要があります。

つまり、日々のネットワークの動きから異常を見つけ出すためには、正常な状態を知っておかなければならないということです。ネットワークが正常なときにベースラインを確立しておけば、正常なときのトラフィックがどのようなものかがわかります。

[†] 監訳注：監訳時点で最新版のバージョン1.8.0では、DMGパッケージをダウンロード後、単にインストーラを起動するだけでインストールできました。本文の手順は、原書執筆時点で最新版だったMac OS X Leopard用のパッケージをインストールする手順のようです。

それではさっそくパケットをキャプチャしてみましょう！

1. Wiresharkを起動します。
2. [Capture] メニューの [Interfaces] をクリックします。パケットをキャプチャできるNICの一覧が、IPアドレスとともにダイアログ上に表示されます。
3. 図3-4からキャプチャしたいNICを選択するか、単にウェルカムページの [Interface List] セクションの下にあるNICをクリックしたうえで、[Start] ボタンをクリックします。これでキャプチャデータが表示されるはず[†]です。

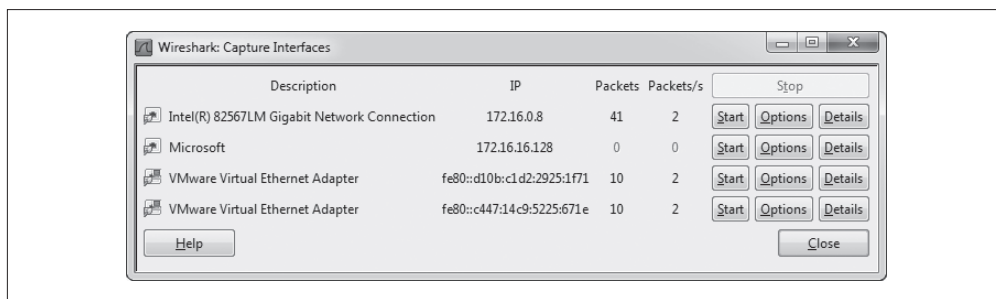


図3-4 パケットをキャプチャするNICを選択する

4. 数分待つて十分なパケットをキャプチャできたら、[Capture] メニューの [Stop] をクリックします。

以上の手順でパケットキャプチャを終了すると、Wiresharkのメインウィンドウにデータが表示さ

[†] 監訳注：監訳時点で最新版のバージョン1.8.0では、複数NICの同時キャプチャを実現するため、原書が前提としている1.6系以前とはインターフェイスが異なっています。図3-4相当のバージョン1.8.0での画面を図3-5に示します。

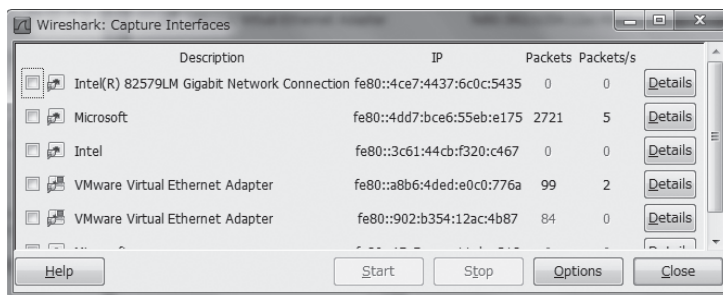


図3-5 パケットをキャプチャするNICを選択する (バージョン1.8.0)

ここでキャプチャしたいNICの左にあるチェックボックスを選択（複数選択可）し、[Start] ボタンをクリックすることで選択したNIC上のパケットのキャプチャが開始されます。ウェルカムページからキャプチャを行う場合は、[Start] ボタンの下にあるNIC一覧からキャプチャを行いたいNICをクリック（複数選択可）してから、[Start] ボタンをクリックします。

れます。大量のデータに圧倒されるかもしれませんが、Wiresharkのメインウィンドウの機能をひとつずつ理解していけば、すぐにわかるようになります。

3.4.2 Wiresharkのメインウィンドウ

パケット解析中に一番よく見るのが、このメインウィンドウでしょう。ここにはキャプチャされたすべてのパケットが、わかりやすい形式で表示されています。先ほどキャプチャしたパケットを使って、図3-6のようなWiresharkのメインウィンドウを見ていきましょう。

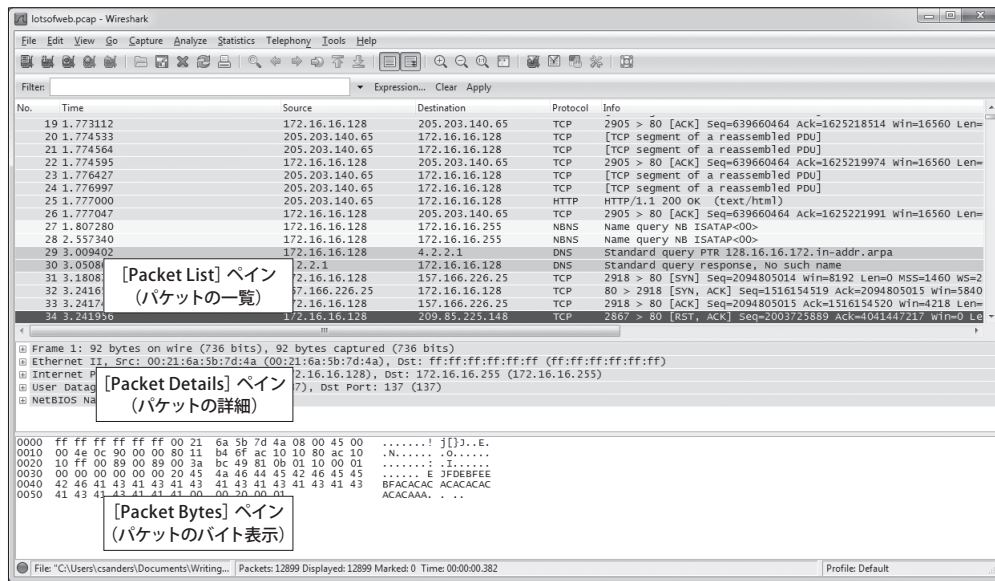


図3-6 3ペイン形式のメインウィンドウ

メインウィンドウの3つのペインの表示は互いに連携しています。[Packet List] ペイン（上段）でパケットをクリックして選択することで、[Packet Details] ペイン（中段）にそのパケットの詳細が表示されます。パケット選択後に[Packet Details] ペインでパケットの各部分をクリックすると、[Packet Bytes] ペイン（下段）で該当部分に対応するバイト列が表示されます。



図3-6では、[Packet List] ペインにいくつかのプロトコルが表示されていることがわかります。プロトコルの階層の違いは、見た目では区別されません。すべてのパケットが、ネットワークから受け取ったとおりに表示されます。

以下に、各ペインの詳細を示します。

Packet List (パケットの一覧) ペイン

上段のペインには、キャプチャファイルに存在するパケットの一覧が、パケット番号、パケットがキャプチャされた相対時刻、パケットの送信元と宛先、パケットのプロトコル、パケットの概要とともに表示されます。



本書中でのトラフィックという言葉は、[Packet List] ペインに表示されているすべてのパケットのことだと思ってください。たとえばDNSトラフィックと言ったときには、[Packet List] ペインに表示されているすべてのDNSプロトコルのパケットのことです。

Packet Details (パケットの詳細) ペイン

中段のペインには、特定のパケットの詳細が階層構造で表示されます。この表示は最初折りたたまれていますが、展開することで特定のパケットに関するすべての情報を見ることができます。

Packet Bytes (パケットのバイト表示) ペイン

下段のペインには、整形される前の生のパケットが表示されています。多分一番意味不明でしょう。これは、ネットワークを行き来するパケットの生情報であり、このままでは解析が非常に困難です。

3.4.3 Wireshark設定

Wiresharkは必要に応じてさまざまなカスタマイズが可能です。Wiresharkの設定画面は、メインウィンドウの [Edit] メニューから [Preferences] をクリックすると表示されます。[Preferences] ダイアログには、カスタマイズ可能なオプションが多数含まれています (図3-7)。

WiresharkのPreferences画面は、6つの主なセクションに分かれています[†]。

[User Interface] セクション

Wiresharkのデータ表示方法を設定できます。ここでは、ウィンドウの場所を記憶するかどうか、ペインのレイアウト、スクロールバーの位置、[Packet List] ペインのカラムのレイアウト、データを表示する際のフォント、ウィンドウの色といった多くのオプションが好みに応じて変更できます。

[Capture] セクション

デフォルトのNIC、プロミスキヤスモードをデフォルトで使用するか、[Packet List] ペインをリアルタイムに更新するかといった、パケットキャプチャに関するオプションを設定できます。

[†] 監訳注：監訳時点で最新の1.8.0では [Filter Expressions] セクションが追加され、7つになっています。

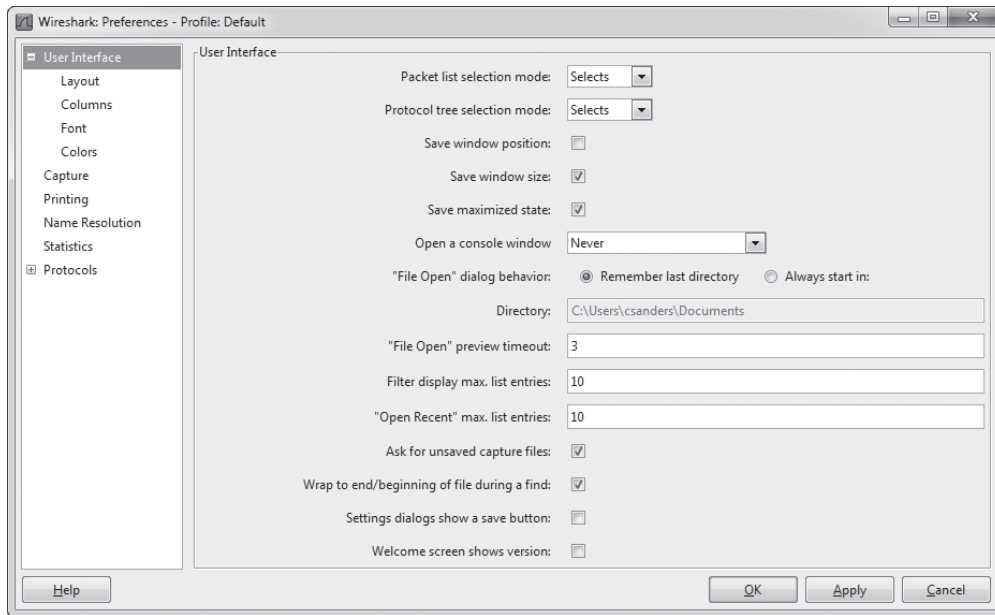


図3-7 PreferencesダイアログでWiresharkをカスタマイズできる

[Printing] セクション

Wiresharkでデータを印刷する際のさまざまなオプションを設定できます。

[Name Resolution] セクション

Wiresharkが持つ (MAC アドレス、トランスポート層などの) アドレスをわかりやすい名前に解決する機能を有効化するかを設定できます。また、同時に実行可能な名前解決リクエストの最大数も設定できます。

[Statistics] セクション

Wiresharkの統計機能に関するオプションが設定できます。

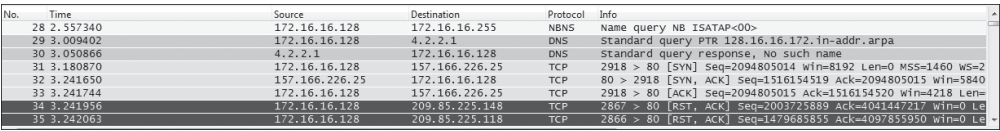
[Protocols] セクション

Wiresharkで解析が可能なさまざまなパケットのキャプチャや表示に関連するオプションを設定できます。プロトコルのすべてに設定オプションがあるわけではありません。これらのオプションは、何らかの理由がない限り変更しないほうがよいでしょう。

3.4.4 パケットの色分け

読者の皆さんが私と同類であれば、鮮やかな画面やきれいな色合いを楽しんでくれると思います。そうすると、図3-8の例のような、色とりどりの [Packet List] ペインも魅力的ではないでしょうか (図はモノクロですが、意味はわかりますよね)。これらの色は適当に決められているようにも見えますが、

そうではありません。



No.	Time	Source	Destination	Protocol	Info
28	2.557340	172.16.16.128	172.16.16.255	NBNS	Name query NB ISATAP-00a
29	3.009402	172.16.16.128	4.2.2.1	DNS	Standard query PTR 128.16.16.172.in-addr.arpa
30	3.050866	4.2.2.1	172.16.16.128	DNS	Standard query response, No such name
31	3.180870	172.16.16.128	157.166.226.25	TCP	2918 > 80 [SYN] Seq=2094805014 win=8192 Len=0 MSS=1460 WS=2
32	3.241650	157.166.226.25	172.16.16.128	TCP	80 > 2918 [SYN, ACK] Seq=1516154519 Ack=2094805015 win=5840
33	3.241744	172.16.16.128	157.166.226.25	TCP	2918 > 80 [ACK] Seq=2094805015 Ack=1516154520 win=4218 Len=0
34	3.244106	172.16.16.128	209.85.225.118	TCP	2866 > 80 [RST, ACK] Seq=2093238589 Ack=101144218 win=0 Len=0
35	3.242063	172.16.16.128	209.85.225.118	TCP	2866 > 80 [RST, ACK] Seq=1479685855 Ack=4097855950 win=0 Len=0

図3-8 Wiresharkによって、プロトコルごとに色分けされている

各パケットの色には意味があり、プロトコルによって色分けされています。たとえば、DNSトラフィックは青、HTTPトラフィックは緑といった具合です。色分けされているおかげで、[Packet List] ペインに表示されている各パケットのProtocolフィールドを1つ1つ確認しなくても、プロトコルを素早く見分けることができます。巨大なキャプチャファイルを扱うときに、この色分けの機能のおかげで解析が大幅にスピードアップすることを実感できるでしょう。

色分けのルールは、図3-9の [Coloring Rules] ダイアログで簡単に確認できます。このダイアログを開くには、メニューから [View] を選択し、[Coloring Rules] をクリックします。

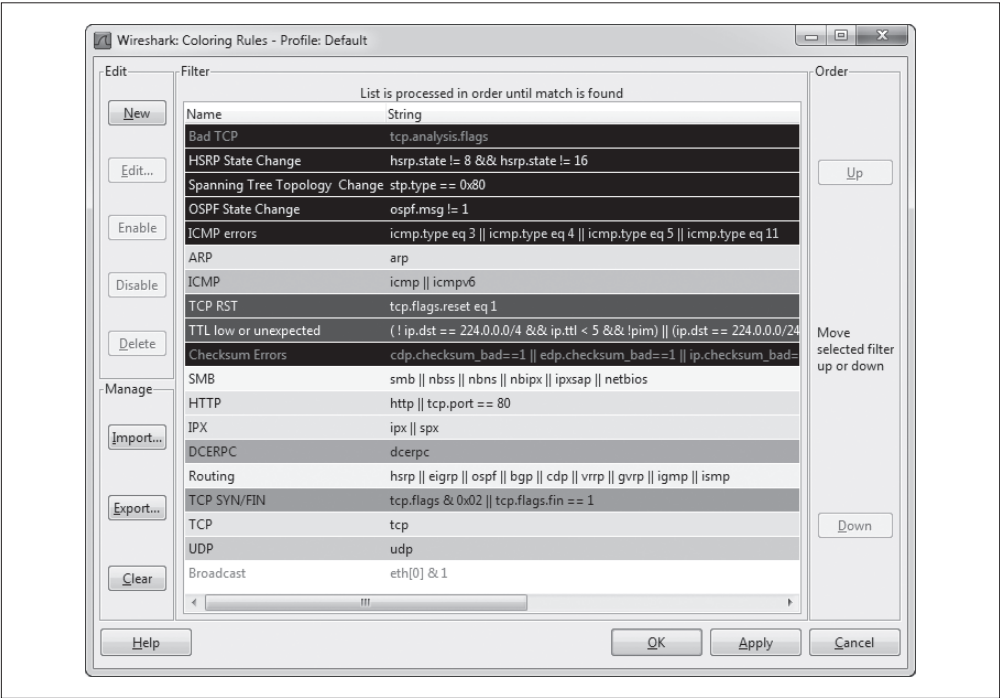


図3-9 [Coloring Rules] ダイアログで、パケットの色分けルールを設定する

ここで、自分の色分けルールを定義したり、既存のものを変更したりすることができます。たとえば、HTTPトラフィックの背景色をデフォルトの緑からラベンダーに変える手順は以下のとおりです。

1. Wiresharkを起動し、[Coloring Rules] ダイアログを表示します（[View] → [Coloring Rules] を選択）。
2. 色分けルールの一覧からHTTPの色分けルールをクリックして選択します。
3. [Edit] ボタンをクリックします。図3-10のような [Edit Color Filter] ダイアログが表示されます。

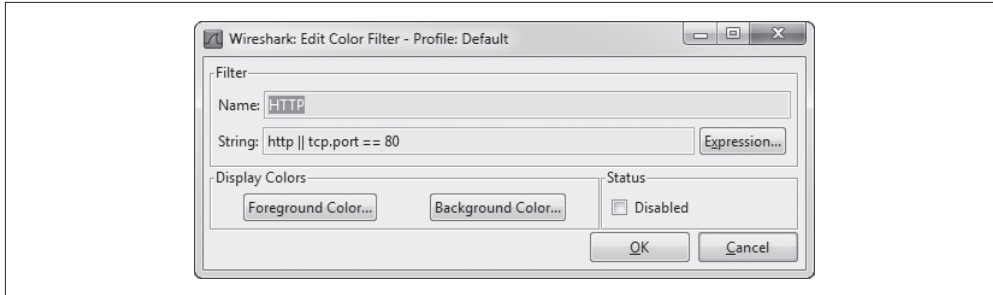


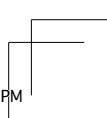
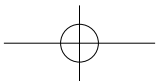
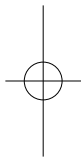
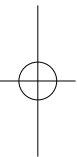
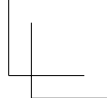
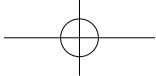
図3-10 [Edit Coloring Filter] ダイアログで、文字色と背景色を設定できる

4. [Background Color] ボタンをクリックします。
5. 選択画面で好みの色を選択し、[OK] ボタンをクリックします。
6. [OK] ボタンを2回押して設定内容を反映し、メインウィンドウに戻ります。設定した色が反映されているはずです。

Wiresharkを使っていると、あるプロトコルが他のプロトコルより多いことに気づくでしょう。色分けされていることでそれがわかりやすくなっています。たとえばDHCPサーバに障害が起こりIPアドレスの割り当てがうまくいかなかった場合、DHCPプロトコルを黄色に（あるいは他のわかりやすい色に）色分けするだけで、DHCPトラフィックを簡単に見分けることができ、パケット解析が効率化します。

独自のカスタムフィルタを作成することで、色分けルールを拡張することもできます。

Wiresharkを起動し、実行したところで、パケット解析の準備が整いました。次章ではキャプチャしたパケットを扱うテクニックについて説明します。



4章

Wiresharkでの パケットキャプチャのテクニック

前章でWiresharkのイロハを紹介したので、実際にパケットをキャプチャし、解析してみましょう。本章ではキャプチャファイル、パケットの操作、時刻の表示形式についてのテクニックを紹介します。またパケットのキャプチャに関する、より高度なオプションを扱うことで、フィルタの世界に飛び込んでいきましょう。

4.1 キャプチャファイルの操作

パケット解析を行ってみると、きちんとした解析は、キャプチャ後に行う必要があることに気づくと思います。通常は、パケットを何回かキャプチャして保存してから、それらを一括して解析することになります。そのため、Wiresharkにはキャプチャしたパケットをキャプチャファイルとして保存し、あとで解析することを可能とする機能が付いています。複数のキャプチャファイルをマージすることもできます。

4.1.1 キャプチャファイルの保存とエクスポート

キャプチャしたパケットを保存するには、メニューから [File] → [Save As] を選択します。すると図4-1のように [Save File As] ダイアログが表示されます。ここでキャプチャしたパケットの保存場所とファイル形式を選択します。ファイル形式を指定しない場合は、.pcap形式で保存されます[†]。

[†] 監訳注：監訳時点で最新版のバージョン1.8.0では、.pcapng形式で保存されます。

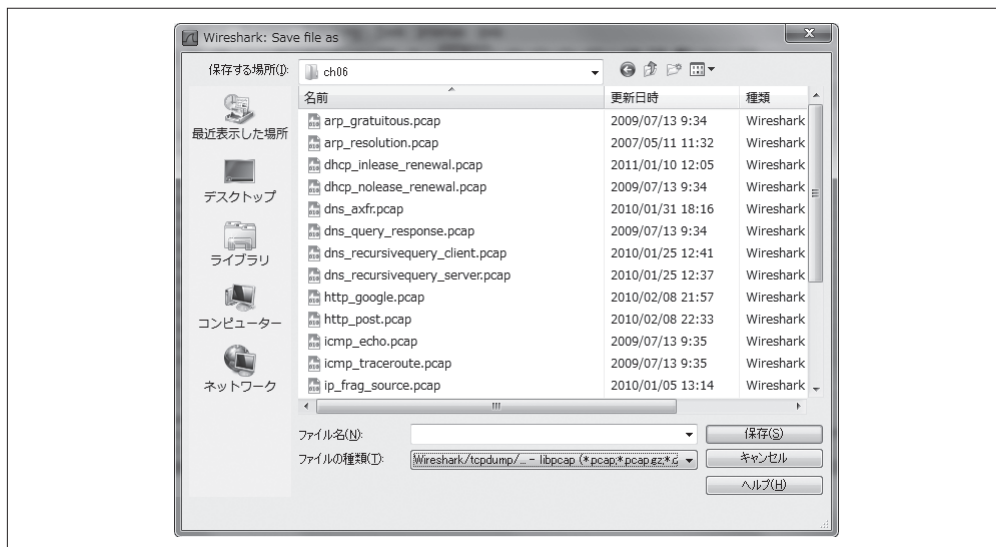


図4-1 [Save File As] ダイアログからキャプチャファイルを保存 (バージョン1.8.0)

[Save File As] ダイアログには、指定した範囲のパケットのみを保存するという強力な機能があります[†]。これは、膨れ上がったキャプチャファイルのサイズを小さくするのに非常に便利です。ある範

[†] 監訳注：監訳時点で最新版のバージョン1.8.0では、この機能は [Save As] ではなく [Export Specified Packets] という別のメニューになっています。[Export Specified Packets] ダイアログを図4-2に示します。

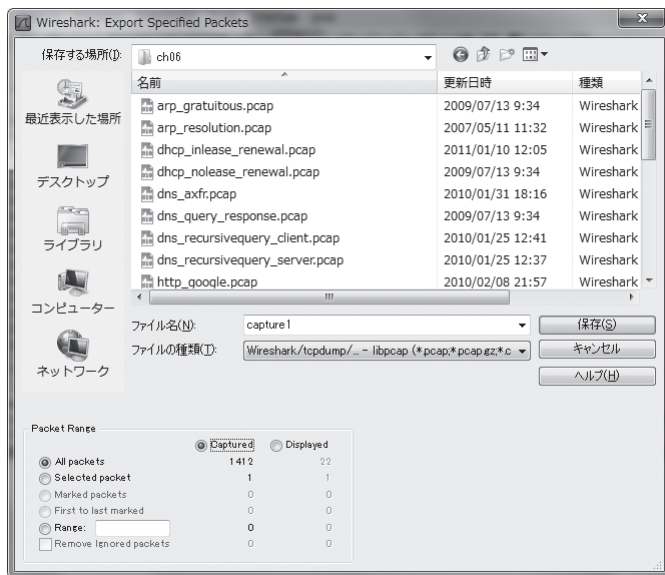


図4-2 [Export Specified Packets] ダイアログ (バージョン1.8.0)

囲のパケット番号のパケット、マーキングされたパケット、ディスプレイフィルタによって表示されたパケット（マーキングとフィルタについては本章で後ほど扱います）など、特定のパケットのみを保存することができます。

Wiresharkでは、別の方法で参照したり、別のパケット解析ツールにインポートしたりするために、テキスト、ポストスクリプト、CSV、XMLといった形式でキャプチャデータをエクスポートすることができます[†]。エクスポートするには、メニューから [File] → [Export] を選択し、エクスポートするファイルのファイル形式を選択してください。[Save As] から保存するときにも、[ファイルの種類] から保存形式を選択することができます。

4.1.2 キャプチャファイルのマージ

パケット解析をしていると、複数のキャプチャファイルをマージしたくなることがあります。これは、2つのデータストリームを比較したり、別々にキャプチャした同じトラフィックのストリームを組み合わせたりするときなどに、よく行われます。

キャプチャファイルをマージするには、マージしたいキャプチャファイルを開き、メニューから [File] → [Merge] を選択して、[Merge with Capture File] ダイアログを開きます（図4-3）。すでに開いているファイルにマージしたいファイルを選択してから、マージ方法を選択します。マージ方法には、[Prepend packets to existing file]（現在表示されているパケットの前にマージするキャプチャ

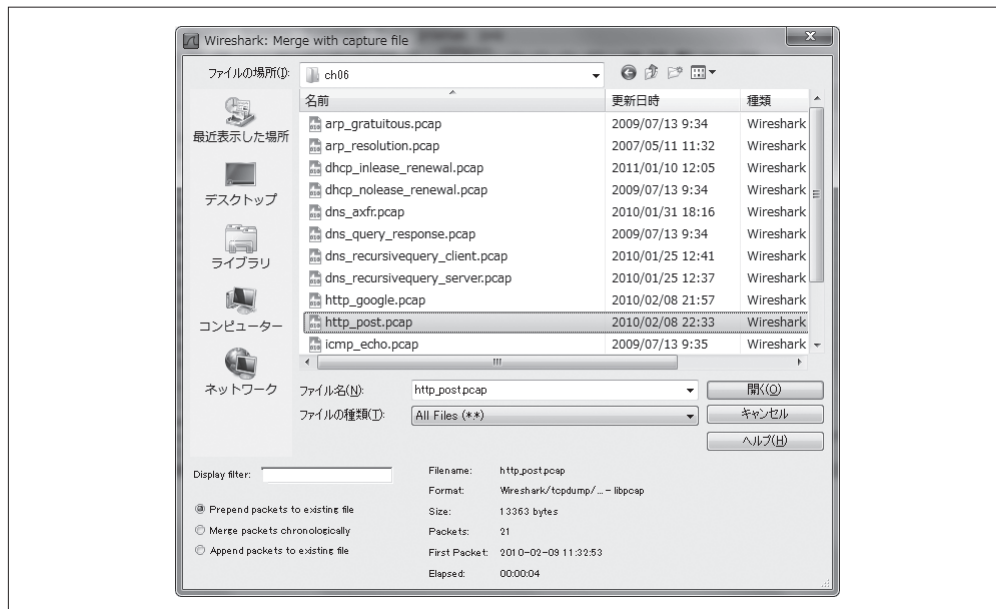


図4-3 [Merge with Capture File] から2つのファイルをマージする

[†] 監訳注：監訳時点で最新版のバージョン1.8.0では、この機能は削除されています。

ファイルのパケットを追加する)、[Append packets to existing file] (現在表示されているパケットのあとにマージするキャプチャファイルのパケットを追加する)、[Merge packet chronologically] (タイムスタンプに沿って時系列に追加する) の3つがあります。

4.2 パケットの操作

パケット解析を始めると、膨大な量のパケットと対峙することになります。数万、数百万とパケットの数が膨れ上がってくると、よほど効率的に解析しないと対応しきれなくなるでしょう。このためWiresharkでは、特定のルールにマッチするパケットを抽出してマーキングすることができるようになっていました。また見やすくするためにパケットを印刷することもできます。

4.2.1 パケットの検索

特定のルールにマッチするパケットを検索するには、図4-4のように、Ctrl + F キーを押して [Find Packet] ダイアログを開きます。

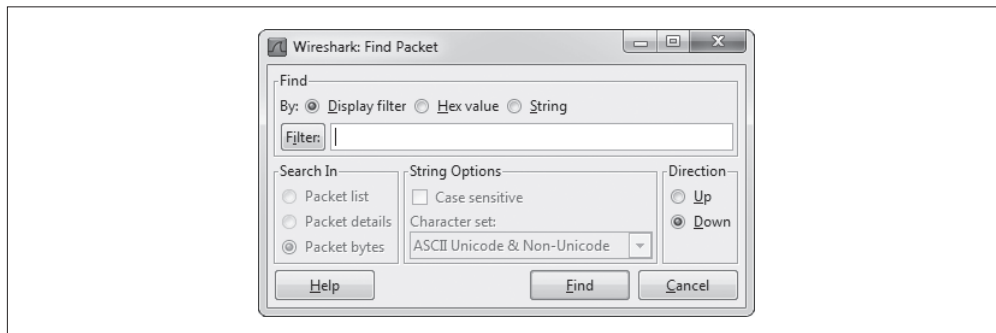


図4-4 Wiresharkで特定のルールにマッチするパケットを検索する

パケットの検索には3つのオプションがあります。

- [Display filter] オプションでは、条件式ベースのフィルタを入力することで、条件式にマッチしたパケットのみを検索します。
- [Hex value] オプションでは、16進数 (バイトをコロンで区切った形式) で指定されたパケットを検索します。
- [String] オプションでは、文字列で指定されたパケットを検索します。

表4-1にそれぞれの例を示します。

表4-1 パケット検索の例

検索のタイプ	例
Display filter	not ip ip address==192.168.0.2 arp
Hex value	00:ff ff:ff 00:AB:B1:f0
String	Workstation1 UserB domain

そのほかにオプションとして、検索対象のペインの指定、使用する文字コードの指定、検索方向の設定が可能です。文字列検索については、検索対象となるペインの指定、使用する文字コードの指定[†]、大文字小文字を区別するかどうかの指定が可能です。

オプションを設定し、テキストボックスに検索の条件式を入力して、[Find] ボタンをクリックすると、最初に条件にマッチしたパケットが表示されます。次候補を検索する場合にはCtrl + Nキーを、前候補を検索する場合にはCtrl + Bキーを押してください。

4.2.2 パケットのマーキング

条件にマッチするパケットを抽出したら、マーキングしておくことができます。マーキングしたパケットだけを個別に保存しておきたい場合や、色をつけてあとから簡単に見つけられるようにしておきたい場合などに、マーキングは便利です。マーキングされたパケットは、図4-5のように黒地に白文字となり目立つようになります（キャプチャされたパケットをファイルに保存するときに、マーキングしたパケットのみを保存することも可能です）。

パケットをマーキングするには、[Packet List] ペインでパケットを右クリックし、表示されたメニューから [Mark Packet] を選んでください。パケットをクリックしてからCtrl + Mキーを押すことでもマーキングできます。マーキングを解除するには、Ctrl + Mキーをもう一度押します。パケットは好きなだけマーキングすることが可能です。複数のパケットをマーキングした場合、Shift + Ctrl + NキーまたはShift + Ctrl + Bでマーキングされたパケット間をジャンプすることができます。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.0.8	157.166.224.25	TCP	3426 > 80 [SYN] Seq=1745901259 win=8192 Len=0 MSS=1460 WS=2
2	0.024063	157.166.224.25	172.16.0.8	TCP	80 > 3426 [SYN, ACK] Seq=2324576412 Ack=1745901260 win=5840 Len=0 MSS=1460 WS=7

図4-5 マーキングされたパケットはハイライト表示される。この例では1番目のパケットがマーキングされ、暗い色になっている

4.2.3 パケットの印刷

パケット解析は画面上で行うことが多いでしょうが、時には印刷する必要があるかもしれません。筆者はパケットを印刷して机の上に貼っておき、別の解析を行っているときでもその内容をすぐに参

[†] 監訳注：日本語文字列を入力した場合は、UTF-8で符号化されたバイト列が検索されます。

照できるようにしています。特に報告書を作成する場合など、パケットをPDF形式で印刷できる機能は非常に便利です。

キャプチャしたパケットを印刷するには、メニューから [File] → [Print] を選択してください。図4-6のような [Print] ダイアログが表示されます。

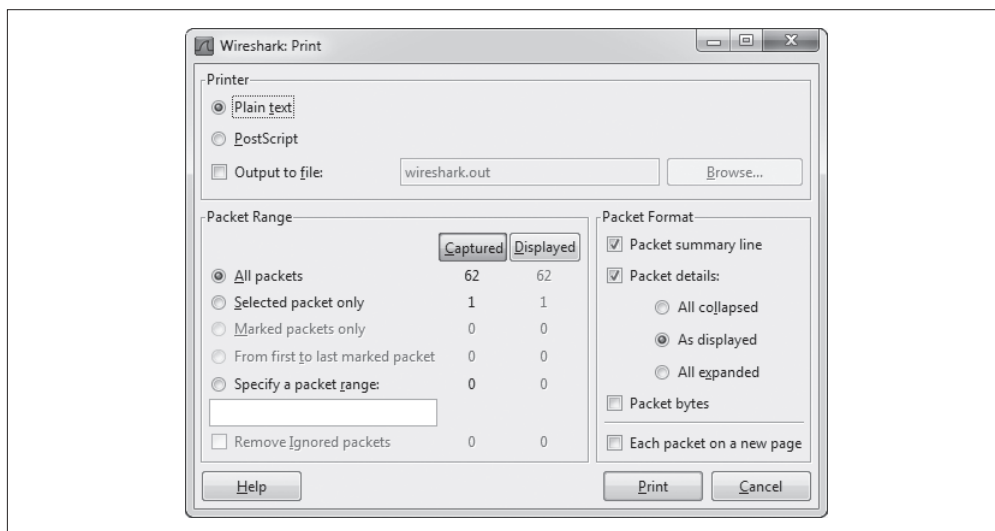


図4-6 [Print] ダイアログからパケットの印刷ができる

[Print] ダイアログから、選択したデータをテキストまたはポストスクリプトとして印刷するか、ファイルとして出力することができます。[Save File As] ダイアログと同じように、ある範囲のパケット番号を持つパケット、マーキングされたパケット、ディスプレイフィルタによって表示されたパケットなど、特定のパケットのみを印刷することもできます。また、3つのペインのうち選択したペインのみを印刷することもできます。オプションを選択したら [Print] ボタンをクリックしてください。

4.3 時刻の表示形式と基準時刻表示

パケット解析において、時刻は重要な要素です。ネットワークで発生している事象は時刻の要素抜きには語れず、解析の際には、ほぼすべてのキャプチャファイルで時刻の傾向やネットワークの遅延を調べることが必要になるでしょう。Wiresharkでは、時間の重要性を踏まえ、いくつかのオプションが提供されています。ここでは、時刻の表示形式と基準時刻表示を見ていきましょう。

4.3.1 時刻の表示形式

Wiresharkでは、キャプチャした各パケットにシステム時刻をもとにしたタイムスタンプが付与されます。Wiresharkはパケットがキャプチャされた時刻を示す絶対時刻を表示することも、直前のパ

ケットから相対時刻やキャプチャ開始時刻からの相対時刻で表示することもできます。

時刻の表示に関するオプションは、メインメニューの [View] にある [Time Display Format] を使って設定します (図4-7)。ここでは、時刻の表示形式のほか、時刻の表示精度についても選択が可能です。時刻の精度については、自動設定以外に、秒、ミリ秒、マイクロ秒などを手動で指定できます。本書の以降の章ではこのオプションを変更している箇所もあるので、今のうちに慣れておいてください。

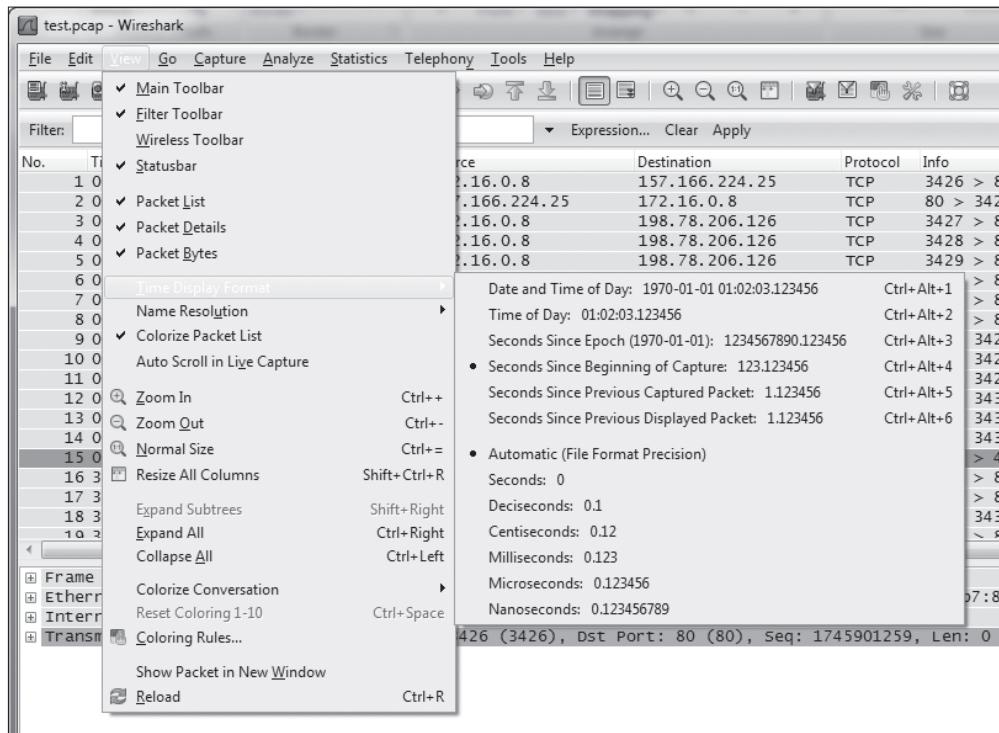


図4-7 さまざまな時刻の表示形式

4.3.2 基準時刻表示

Wiresharkでは、あるパケットがキャプチャされた時点からの相対的な時間を表示する基準時刻表示 (Packet Time Reference) の設定ができます。この機能は、キャプチャの開始時点以外のどこかで開始された一連のイベントを調べるときに特に便利です。

あるパケットからの相対的な時刻を表示するには、[Packet List] ペインから基準としたいパケットを選択し、メニューから [Edit] → [Set Time Reference] を選択します。表示を止めるには、パケットを選択し、[Edit] → [Set Time Reference] を再度選択します。このオプションはトグルになっています。

この設定を行うと、基準となったパケットの [Packet List] ペインにおける [Time] カラムは、図 4-8 のように *REF* と表示されます。

この設定は、時刻の表示形式をキャプチャ開始時点からの相対時刻にしておかないと意味がありません。それ以外の形式では無意味だけでなく、誤解を招く表示が行われてしまいます。

No.	Time	Source	Destination
4	0.118129	172.16.0.8	198.78.206.126
5	*REF*	172.16.0.8	198.78.206.126
6	0.000077	172.16.0.8	198.78.206.126
7	0.000153	172.16.0.8	198.78.206.126

図4-8 基準時刻表示の基準になったパケット

4.4 キャプチャオプションの設定

3章ではパケットキャプチャの基本中の基本について説明しました。Wiresharkでは図 4-9 に示すように、[Capture Options] ダイアログからさまざまなオプションを設定できます。ダイアログを開くに

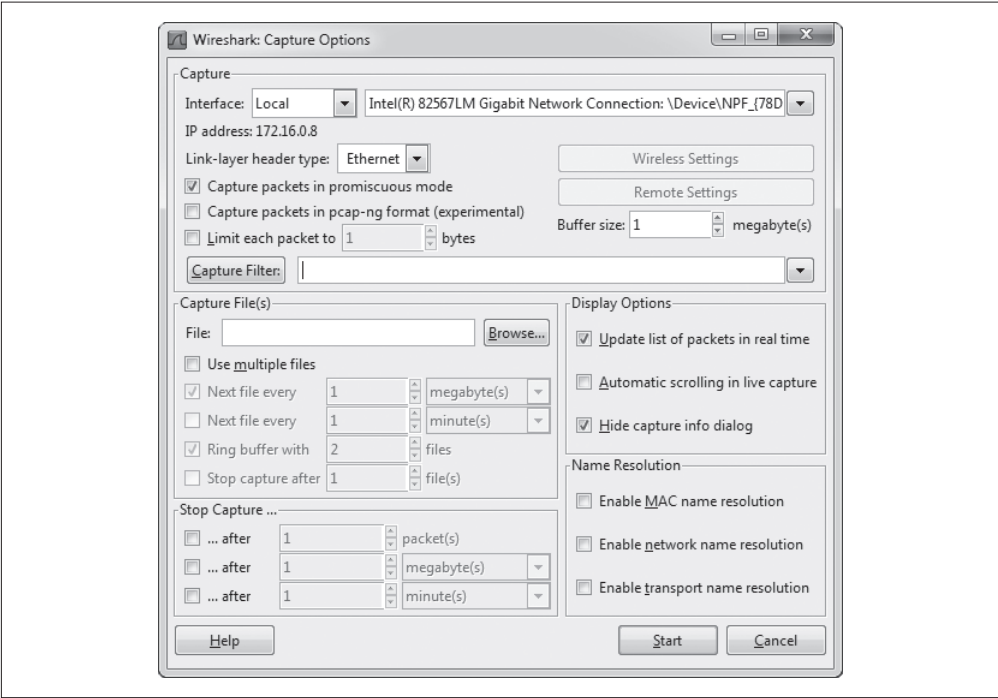


図4-9 [Capture Options] ダイアログ

は、[Capture] → [Interfaces] を選択し、パケットをキャプチャしたいインターフェイスの横にある [Options] ボタンをクリックします[†]。

[Capture Options] ダイアログにはあれこれとおまけがついていますが、いずれもパケットキャプチャを便利にするためのものです。[Capture]、[Capture Files]、[Stop Capture]、[Display Options]、[Name Resolution] といったオプションをひとつずつ見ていきましょう^{††}。

4.4.1 Captureの設定

[Capture] セクションの [Interface] ドロップダウンリストで、設定対象のネットワークインターフェイスを選択します。左側のドロップダウンリストから Local と Remote を切り替えることで、リモートホストのインターフェイスを指定することも可能です。右側のドロップダウンリストではキャプチャに使用可能なインターフェイスの一覧が表示されます。ドロップダウンメニューの下に、選択したイン

[†] 監訳注：監訳時点で最新版のバージョン 1.8.0 では、前章の図 3-5 で説明したように複数 NIC のパケットを同時にキャプチャ可能となったため、[Options] ボタンの位置は [Capture Interfaces] ダイアログ下部に変更されています。キャプチャするインターフェイスは [Capture Options] ダイアログ内で改めて選択するようになっています。

^{††} 監訳注：監訳時点で最新版のバージョン 1.8.0 では、図 3-5 で説明したとおりキャプチャ対象のインターフェイスが複数選択可能となっているため、図 4-10 のように、[Capture Options] ダイアログが大幅に変更されています。

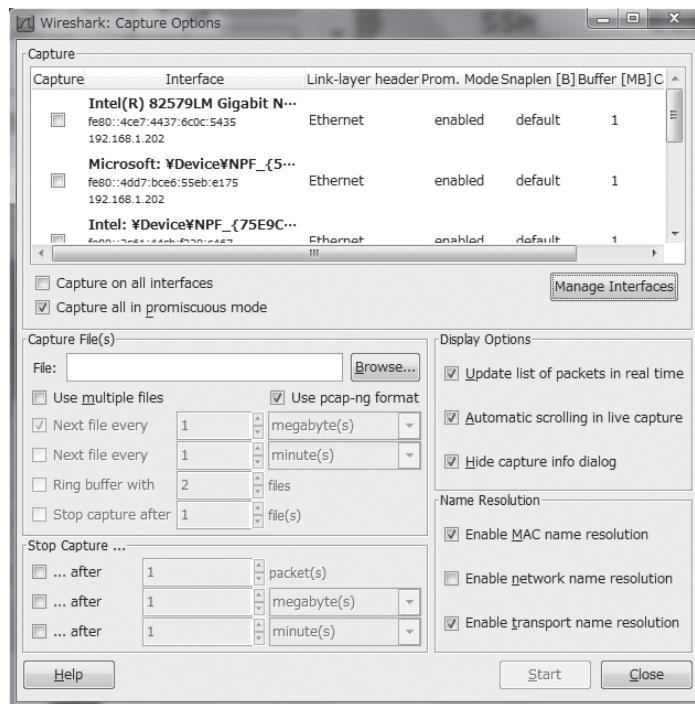


図4-10 [Capture Options] ダイアログ

(次ページに続く)

ターフェイスのIPアドレスが表示されます。

左中段の3つのチェックボックスで、プロミスキャスモードの有効無効を切り替えたり（デフォルトでは有効になっています）、現時点では実験段階にある pcap-ng フォーマットでパケットをキャプチャしたり、各パケットのサイズの上限をバイト単位で指定したりすることができます。

[Capture] セクションの右中段のボタンでは、無線やリモートの設定が可能です（ただし、これらが使用可能な場合のみ）。

その下にある [Buffer size] オプションは、Windowsが動作しているシステムでのみ指定可能なオプションで、キャプチャしたパケットをディスクに書き込む前に、一時保存するカーネルバッファのサイズを指定します（大量のパケットが破棄されているといった状況にならない限り、この値を変更する必要はないでしょう）。

4.4.2 Capture File設定

[Capture File(s)] セクションで設定を行うことで、パケットをまずキャプチャしてからファイルに保存するのではなく、キャプチャしたパケットを自動的にファイルに保存することが可能となり、パケットを保存する手間を省くことができます。保存の際は、単一ファイル、ファイルセット、指定した数のファイルで循環を行うリングバッファ形式が選択できます。このオプションを使うには、[File] テキストボックスにファイルのフルパス名を入力します。

大量のパケットをキャプチャする場合や、長期にわたってキャプチャを行う場合には、ファイルセットが特に便利です。ファイルセットとは、指定された条件で分割された一連のファイルを意味します。

（承前）リモートホストのインターフェイスの設定は、[Capture] セクションの右下にある [Manage Interfaces] ボタンを押すと表示されるメニューで行います。

プロミスキャスモードの設定、各パケットのサイズ上限、[Buffer size] オプションに相当する設定は、各インターフェイスをダブルクリックすると表示される [Edit Interface Settings] ダイアログでインターフェイスごとに行います（図4-11）。プロミスキャスモードの有効化、無効化をすべてのインターフェイスで一度に切り替える場合は、[Capture] セクションの左下にあるチェックボックスから設定することもできます。

pcap-ng フォーマットの使用は [Capture File(s)] セクションの [Use pcap-ng format]（図4-10）で制御します。

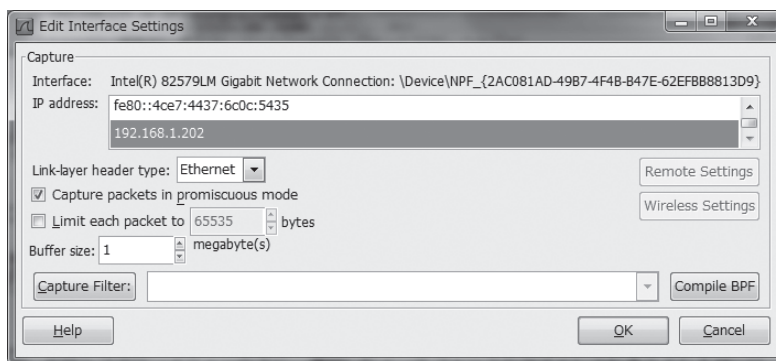


図4-11 [Edit Interface Settings] ダイアログ

ファイルセットとして保存するには、[Use Multiple Files] オプションを選択します。

Wiresharkでは、ファイルサイズと時間に基づくさまざまなトリガーで、ファイルセットの保存タイミングを制御することができます。これらのトリガーを有効にするには、[Next File Every] オプション（上がファイルサイズに基づくトリガー、下が時間に基づくトリガー）のチェックボックスを選択し、トリガーの値と単位を指定します。キャプチャしたパケットのサイズが1MBに達するごとに新しいファイルを作成するトリガーや、図4-12のようにキャプチャ時間が1分経過するごとに新しいファイルを作成するトリガーなどを作ることができます。

Name	Date modified
Capture_00001_20091115155100	11/15/2011 3:51 PM
Capture_00002_20091115155200	11/15/2011 3:52 PM
Capture_00003_20091115155300	11/15/2011 3:53 PM
Capture_00004_20091115155400	11/15/2011 3:54 PM
Capture_00005_20091115155500	11/15/2011 3:56 PM
Capture_00006_20091115155600	11/15/2011 3:56 PM
Capture_00007_20091115155700	11/15/2011 3:57 PM
Capture_00008_20091115155800	11/15/2011 3:58 PM
Capture_00009_20091115155900	11/15/2011 3:59 PM
Capture_00010_20091115160000	11/15/2011 4:00 PM

図4-12 Wiresharkにより1分間隔で作成されたファイルセット

オプションを組み合わせることも可能です。たとえば先ほどのトリガーを両方指定すると、データが1MBキャプチャされるか、あるいは1分経過するか、どちらかの条件が満たされると新たなファイルが作成されます。

[Ring Buffer With] は、ファイルセット作成の際にリングバッファを使うオプションです。これはWiresharkが複数のファイルに書き込んでいく際に、いわゆるFIFO方式を用いる指定を行います。情報技術では「リングバッファ」にはさまざまな意味がありますが、ここではファイルセットの最後のファイルがいっぱいになった時点でさらにデータを保存する必要が発生した場合、最初のファイルを上書きするようなファイルセットを意味します。このオプションを設定した場合、作成するファイルの最大数を指定します。仮に1時間おきに新しいファイルを作成するよう設定し、リングバッファを「6」に設定したとしましょう。6番目のファイルが作成されるとリングバッファが一巡し、7番目のファイルを作成する代わりに、最初のファイルを上書きします。この設定をすると、新しいデータは書き込まれますが、ハードディスク内のデータファイルの数が6個以上になることはありません。

[Stop Capture After] オプションを使うと、指定した数のファイルが作成された時点で、キャプチャを停止するよう設定できます。

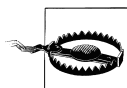
4.4.3 Stop Capture設定

[Stop Capture] セクションでは、トリガーの条件が満たされた時点でキャプチャを停止する設定が行えます。ファイルセットを用いる場合、ファイルサイズや時間の間隔以外に、パケット数によるトリ

ガーも作成できます。これらのオプションは、先ほど説明したオプションとの併用が可能です。

4.4.4 Display Options設定

[Display Options] セクションでは、キャプチャしたパケットの表示方法を制御できます。[Update List of Packets in Real Time] は見てのとりのオプションで、[Automatic Scrolling in Live Capture] オプションと組み合わせて使用することができます。両方を有効にすると、直近でキャプチャしたパケットから降順で、キャプチャしたすべてのパケットが表示されます。



[Update List of Packets in Real Time] と [Automatic Scrolling in Live Capture] を一緒に使用すると、それほど大量でないデータをキャプチャする場合でも、CPUに相当の負荷がかかります。リアルタイムでパケットを確認する必要がない限り、このオプションは両方とも使わないほうがよいでしょう。

[Hide Capture Info Dialog] オプションを使うと、キャプチャ中は、キャプチャしたパケットの数とパーセントをプロトコルごとに表示する小さなウィンドウだけが表示されるようになります。

4.4.5 Name Resolution設定

[Name Resolution] セクションでは、キャプチャ内のMAC（第2層）、ネットワーク層（第3層）、トランスポート（第4層）で自動的な名前解決の実施を制御できます。Wiresharkの名前解決については、その欠点も含め、5章で詳しく説明します。

4.5 フィルタを使う

フィルタを使うと、解析対象のパケットを的確に指定できます。簡単に言うと、フィルタは、どのパケットを対象に含むか除外するかを条件を定義する構文です。表示したくないパケットがあれば、フィルタを作成して除外できます。あるパケット以外を見たくないという場合は、見たいパケットだけを表示するフィルタを作成すればよいのです。

Wiresharkには、大きく2種類のフィルタがあります。

- キャプチャフィルタは、パケットをキャプチャしている際に適用されるもので、指定された構文に基づき、特定のパケットのみをキャプチャするものです。
- ディスプレイフィルタは、キャプチャ済のパケットに適用されるもので、指定された構文に基づき、不要なパケットを非表示にしたり、必要なパケットのみを表示したりするものです。

まずはキャプチャフィルタを見てみましょう。

4.5.1 キャプチャフィルタ

キャプチャフィルタは、パケットをキャプチャしている際に適用されます。キャプチャフィルタを使

う主な理由はパフォーマンスにあります。トラフィックの特定の範囲が解析不要であるとわかっている場合、キャプチャフィルタでフィルタを行えば、こうしたパケットのキャプチャに使う分のCPUが節約できます。

独自のキャプチャフィルタを作成できる機能は、大量のデータを扱う場合に役立ちます。必要なパケットのみに限定して調査を進めることで、解析作業を高速化できるからです。

キャプチャフィルタの使用例として、たとえばさまざまなサービスを提供しているサーバのトラフィックをキャプチャする場合があります。たとえば、262番ポートを使用するサービスを提供しているサーバのトラブルシューティングを考えてみましょう。解析対象のサーバがさまざまなポートでサービスを提供している場合、262番ポートのトラフィックのみを見つけて解析するだけでも一苦労ですが、キャプチャフィルタを使えば262番ポートのパケットのみをキャプチャできます。この章の最初に説明したように、[Capture Options] ダイアログから、以下のようにしてキャプチャフィルタを作成できます。

1. [Capture] → [Interfaces] を選択したら、パケットのキャプチャに使いたいインターフェイスの右にある [Options] ボタンをクリックし、[Capture Options] ダイアログを開きます[†]。
2. パケットのキャプチャに使いたいインターフェイスを選択します^{††}。
3. [Capture Filter] ボタンの横に構文を入力することで、キャプチャフィルタが適用されます。今回は262番ポートで送受信するトラフィックだけをキャプチャしたいので、図4-13のように「port 262」と入力します（構文については次の項で詳しく説明します）。
4. フィルタを作成したら、[Start] ボタンをクリックしてキャプチャを始めます[‡]。

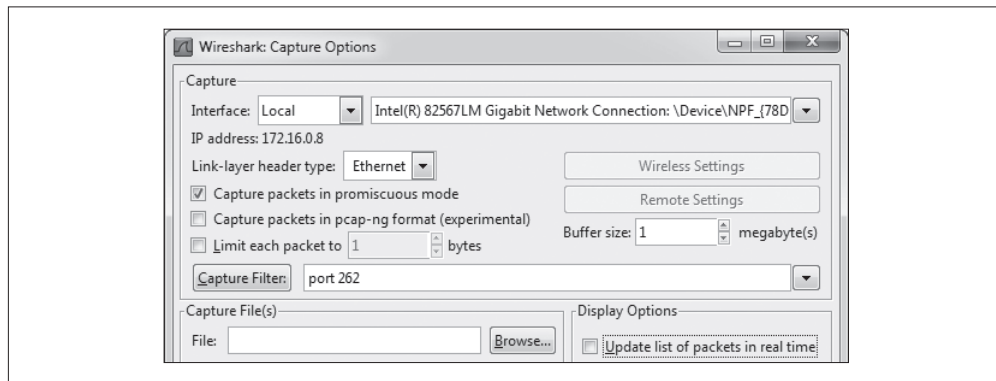


図4-13 [Capture Options] ダイアログでキャプチャフィルタを作成する

† 監訳注：監訳時点で最新版のバージョン1.8.0では、ダイアログ下部の [Options] ボタンをクリックして、[Capture Options] ダイアログを開きます。

†† 監訳注：監訳時点で最新版のバージョン1.8.0では、インターフェイスをダブルクリックして図4-11で示した [Edit Interface Settings] ダイアログを開きます。

‡ 監訳注：監訳時点で最新版のバージョン1.8.0では、[OK] ボタンをクリックしてから [Start] ボタンをクリックします。

ある程度のパケットをキャプチャしてから確認すると、262番ポートで送受信するトラフィックのみが見えているはずです。これで必要なデータだけを効率よく解析できます。

4.5.1.1 キャプチャ／BPF構文

キャプチャフィルタはWinPcapが解析するもので、Berkeley Packet Filter (BPF) 構文で記述します。さまざまなパケット解析ツールでこの構文が使われているのは、大半のパケット解析ツールがBPF構文を理解するlibpcap/WinPcapライブラリを使っているからです。ネットワークをパケットレベルで深く解析する際にBPF構文の知識は不可欠です。

BPF構文を使って作成したフィルタを「expression (式)」と呼び、それぞれの式は1つ以上の「primitive (プリミティブ)」で構成されています。プリミティブは(表4-2のように) 1つ以上の「qualifier (修飾子)」と、そのあとに続く図4-14で示したようなID文字列または数値のセットから構成されます。

表4-2 BPFの修飾子

修飾子の種類	説明	例
Type	ID文字列や数値の意味	host, net, port
Dir	ID文字列や数値の転送方向	src, dst
Proto	特定のプロトコル	ether, ip, tcp, udp, http, ftp

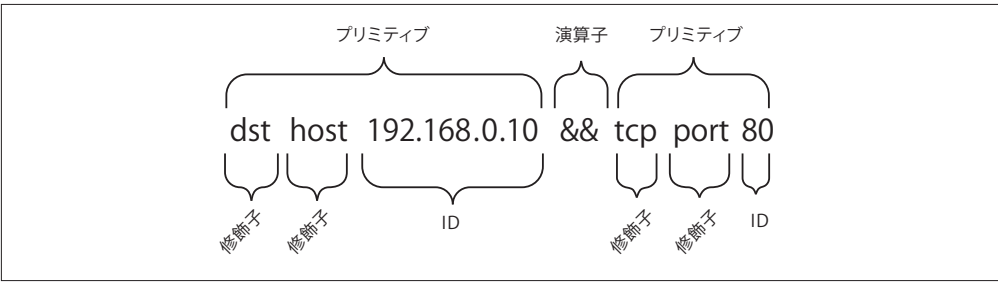


図4-14 キャプチャフィルタの一例

実際に式を作ってみましょう。src修飾子と192.168.0.10というID文字列を組み合わせることでプリミティブを作成します。このプリミティブのみだと、192.168.0.10というIPアドレスからのトラフィックのみをキャプチャするという式になります。

論理演算子を使ってプリミティブを組み合わせ、より高度な式を作成することが可能です。使える論理演算子は以下の3つです。

- 論理積演算子 AND (&&)
- 論理和演算子 OR (||)
- 否定演算子 NOT (!)

一例を挙げると、次の式は192.168.0.10という送信元IPアドレスで、80番ポートで送受信されるトラフィックのみをキャプチャします

```
src 192.168.0.10 && port 80
```

4.5.1.2 ホスト名とアドレスによるフィルタ

フィルタの多くは、通常特定のネットワーク機器、機器群を指定するものです。状況によって、機器のMACアドレス、IPv4アドレス、IPv6アドレス、DNSホスト名などに基づいてフィルタを行います。

たとえば、ネットワーク上のあるサーバとやり取りしている、とあるホストのトラフィックが気にかかるとしましょう。サーバ側で、そのホストのIPv4アドレスを用いるすべてのトラフィックをキャプチャするフィルタを、host修飾子を使って作成します。

```
host 172.16.16.149
```

IPv6ネットワークの場合は、host修飾子を使ってIPv6アドレスによるフィルタを行います。

```
host 2001:db8:85a3::8a2e:370:7334
```

ホスト名でフィルタを行うことも可能です。

```
host testserver2
```

ホストのIPアドレスが変更される可能性を危惧するのであれば、etherというプロトコル修飾子によって、MACアドレスでフィルタを行うことも可能です。

```
ether host 00-1a-a0-52-e2-a0
```

転送方向を示す修飾子は、これらの設定と組み合わせることで指定した機器を送信元もしくは宛先とするトラフィックをキャプチャする場合によく使われます。たとえば、特定ホストが送信元のトラフィックのみをキャプチャするなら、src修飾子を付加します。

```
src host 172.16.16.149
```

172.16.16.149のサーバが挙動不審な場合に、そのサーバを宛先とするデータのみをキャプチャするなら、dst修飾子を付加します。

```
dst host 172.16.16.149
```

Type修飾子(host、net、port)をプリミティブで使わない場合は、host修飾子を指定したものとみなされます。したがって前の例ではhost修飾子を省くことも可能です。

```
dst 172.16.16.149
```

4.5.1.3 ポートとプロトコルによるフィルタ

ホストに基づくフィルタ以外に、各パケットが用いるポートでフィルタを行うこともできます。ポー

トによるフィルタは、サービスポートがわかっているサービスやアプリケーションに基づくフィルタを行う際に使用できます。たとえば、8080 番ポートで通信されるトラフィックのみをキャプチャする簡単なフィルタは以下ようになります。

```
port 8080
```

8080 番ポート以外で通信されるすべてのトラフィックをキャプチャする場合は次のようになります。

```
!port 8080
```

ポートによるフィルタと転送方向を示す修飾子を組み合わせることも可能です。たとえば、標準的な 80 番の HTTP ポートで待ち受けている Web サーバを宛先とするトラフィックのみをキャプチャする場合、dst 修飾子を使います。

```
dst port 80
```

4.5.1.4 プロトコルによるフィルタ

プロトコルによるフィルタを使うことで、特定のプロトコルでフィルタを行うことができます。これは、ポートによる指定ができない、アプリケーション層以外のプロトコルを指定する際に用います。たとえば、ICMP トラフィックだけを参照したい場合にも、このフィルタを使用できます。

```
icmp
```

IPv6 トラフィック以外のすべてのトラフィックを見たい場合は、この技が使えます。

```
!ip6
```

4.5.1.5 プロトコルフィールドによるフィルタ

BPF 構文の底力のひとつが、プロトコルヘッダの各バイトを調べて、そのデータに基づく特殊なフィルタを作成できる機能です。ここで説明するこの高度なフィルタを使うと、パケットの指定の位置から指定のバイト数を確認することができます。

たとえば、ICMP ヘッダのタイプフィールドでフィルタを行いたいとしましょう。タイプフィールドはパケットの一番先頭にあり、オフセット値が 0 となっています。パケット内で調べたい位置を識別するには、プロトコル修飾子に続いてオフセット値を [] 記号 (square bracket) で囲んで指定します。今回の例では icmp[0] となります。これにより 1 バイトの整数値が返却されるので、比較に用いることができます。たとえば、到達不能 (Destination Unreachable) メッセージ (タイプ 3) の ICMP パケットのみを取得するなら、フィルタ式で次のように等値演算子を使います。

```
icmp[0] == 3
```

エコー要求 (タイプ 8) もしくはエコー応答 (タイプ 0) の ICMP パケットを調べたいという場合は、2 つのプリミティブと OR 演算子を使います。

```
icmp[0] == 8 || icmp[0] == 0
```

これらのフィルタはうまく機能しますが、フィルタに使えるのは、パケットヘッダ内のたった1バイトだけです。幸いなことに、[] 記号内のオフセット値のあとにバイト長をコロン(:)で区切って付加することで、返却されるデータ長を指定することができます。

たとえば、ICMPのタイプ3、コード1（ホスト到達不能）パケットをキャプチャするフィルタを作成したいとします。これはパケットヘッダのオフセット0から始まる1バイトのフィールド2つになります。これを識別するためには、パケットヘッダのオフセット0から始まる2バイトのデータを確認し、16進数0301（タイプ3、コード1）と比較するフィルタを作成します。

```
icmp[0:2] == 0x0301
```

RSTフラグがセットされたTCPパケットだけをキャプチャしたいということがよくあります。TCPについては6章で詳しく説明しますので、ここではTCPパケットのフラグがオフセット13にあるということだけ理解してください。これは1バイトのフラグフィールドですが、このバイト内の各ビットによってフラグが識別されるというフィールドです。TCPパケットでは複数のフラグを同時に設定できるので、tcp[13]という表現だけでは効果的にフィルタを行うことができません。複数の値でRSTビットがセットされている可能性があるからです。調査対象のバイト内での位置を指定するためには、このプリミティブに&記号を付加する必要があります。RSTフラグはこのバイト内で4という値なので、4番目のビットがセットされていればフラグが設定されていることになります。これを示すフィルタは次のようになります。

```
tcp[13] & 4 == 4
```

8という値のビットのPSHフラグがセットされているパケットを参照したい場合は、代わりにその位置を指定します[†]。

```
tcp[13] & 8 == 8
```

4.5.1.6 キャプチャフィルタ式の例

現状にあったフィルタを作成できるかどうかで、解析の成否が決まるといっても過言ではないでしょう。表4-3は筆者がよく使うキャプチャフィルタの一例です。

[†] 監訳注：著者はビット位置とその値のどちらで説明するかについて若干混乱しているように見えます。要は、たとえばPSHフラグはビット3がセットされることで有効化されるため、「(ビット3が1) → 2³ → 8」、つまり8と論理積を行えば、ビット3がセットされているかどうかを確認できるということです。

表4-3 よく使うキャプチャフィルタ

フィルタ	説明
tcp[13] & 32 ==32	URG フラグがセットされたTCP パケット
tcp[13] & 16 ==16	ACK フラグがセットされたTCP パケット
tcp[13] & 8 == 8	PSH フラグがセットされたTCP パケット
tcp[13] & 4 == 4	RST フラグがセットされたTCP パケット
tcp[13] & 2 == 2	SYN フラグがセットされたTCP パケット
tcp[13] & 1 == 1	FIN フラグがセットされたTCP パケット
tcp[13] ==18	TCP SYN-ACK パケット
ether host00:00:00:00:00:00 (実際のMACアドレスに置き換える)	指定したMACアドレスで送受信されるトラフィック
!ether host 00:00:00:00:00:00 (実際のMACアドレスに置き換える)	指定したMACアドレス以外で送受信されるトラフィック
broadcast	ブロードキャストトラフィックのみ
icmp	ICMPトラフィック
icmp[0:2] == 0x0301	ICMP ホスト到達不能
ip	IPv4トラフィックのみ
ip6	IPv6トラフィックのみ
udp	UDPトラフィックのみ

4.5.2 ディスプレイフィルタ

ディスプレイフィルタは、キャプチャファイルに適用されるフィルタで、フィルタにマッチするパケットのみを表示させるものです。ディスプレイフィルタは [Packet List] ペインの上部にある [Filter] テキストボックスに設定します。

ディスプレイフィルタを使う機会はキャプチャフィルタより多いでしょう。これは、実際のキャプチャファイルのデータを損なうことなく、特定のパケットのフィルタを行うことができるからです。フィルタの式を消去するだけで、元々のキャプチャファイルを必要に応じて再表示することができます。

ディスプレイフィルタは、キャプチャファイルから、無意味なブロードキャストパケットを消去する際にも役立ちます。たとえば [Packet List] ペインからARPブロードキャストが解析したいトラブルと関係ないので消去したいといった場合です。とはいえ、こうしたARPブロードキャストパケットはのちほど解析に必要なかもしれないので、削除するよりも、ディスプレイフィルタで一時的に表示させないようにするほうがよいのです。

ARPパケットを非表示にするには、[Packet List] ペインの上部にある [Filter] テキストボックスにフォーカスした状態で図4-15のように「!arp」と入力し、[Packet List] ペインからすべてのARPパケットを消去します。フィルタを削除するには、[Clear] ボタンをクリックします。



図4-15 [Packet List] ペイン上部の [Filter] テキストボックスでディスプレイフィルタを作成する

4.5.2.1 [Filter Expression] ダイアログ (簡単な作成方法)

図4-16の [Filter Expression] ダイアログは、Wireshark 初心者がキャプチャフィルタやディスプレイフィルタを簡単に作成できるようにしてくれる機能です。ダイアログを表示するには、[Capture Options] で [Capture Filter] ボタンをクリックし、次に [Expression] ボタンをクリックします[†]。

ダイアログの左側には、使用可能なプロトコルフィールドの一覧が表示されており、ここから使用可能なフィルタ要素を指定できるようになっています。フィルタを作成するには、以下の手順に従ってください。

1. プロトコル名の左にある [+] をクリックして、各プロトコルで利用可能なフィルタ要素を参照します。利用したいフィルタ要素をクリックしてください。
2. 選択したフィルタ要素と、その値の評価方法を指定してください。評価方法は、等しい (=)、大なり (>)、小なり (<) などの演算子です。
3. 値を指定して、フィルタ式を作成します。Wireshark が提供する定義済の値から選択するか、自身で値を設定してください。
4. フィルタの作成が完了したら [OK] ボタンをクリックしてください。作成したフィルタがテキストで表示されます。

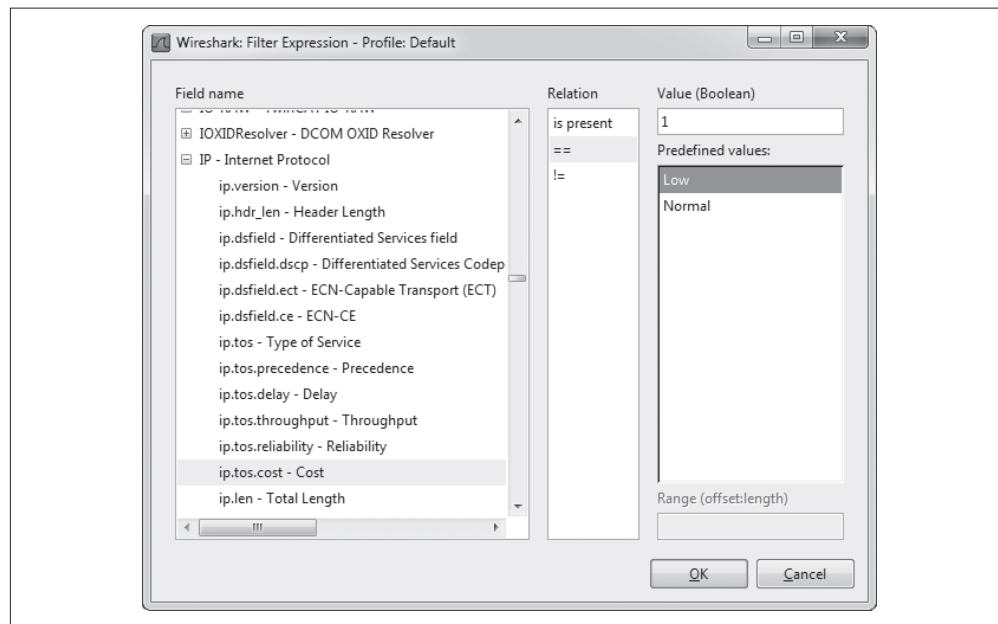


図4-16 [Filter Expression] ダイアログを使うとフィルタを簡単に作成できる

[†] 監訳注：おそらくこの記述は筆者の勘違いで、実際はメインウィンドウで [Expression] ボタンをクリックするか、メインメニューの [Analyze] から [Display Filter] を選択し、ダイアログにある [Expression] ボタンをクリックするといった必要があります。

[Filter Expression] ダイアログは初心者には非常に便利な機能ですが、フィルタの使用方法が理解できれば、手動でフィルタを作成するほうが効率がよいでしょう。ディスプレイフィルタは非常に強力ですが、構文は簡単です。

4.5.2.2 フィルタ式の文法（難しい作成方法）

キャプチャフィルタやディスプレイフィルタは、特定のプロトコルに基づいたフィルタを作成する際に使うことが多いでしょう。たとえばTCPのトラブルシューティングの場合、TCPのトラフィック以外は見る必要がないので、TCPのみを表示するフィルタを作成すればよいのです。

この課題を別の側面から見てみましょう。TCPに関連したトラブルシューティングを行う道筋を考えてみます。pingを多用して、ICMPのトラフィックが大量に発生した場合に、!icmpというフィルタを使えば、ICMPのトラフィックを消去することができます。

比較演算子を使えば、値を比較することができます。たとえばTCP/IPネットワークのトラブルシューティングの場合、特定のIPアドレスを参照するすべてのパケットを参照することがよくあります。比較演算子「==」を使えば、192.168.0.1というIPアドレスを含むパケットのみを表示するフィルタが作成できます。

```
ip.addr==192.168.0.1
```

今度は長さが128バイト以下のパケットのみを表示する場合を考えてみましょう。この場合は「<=」という比較演算子をフィルタ式で使用すればよいのです。

```
frame.len <=128
```

Wiresharkで使用可能な比較演算子は表4-4のとおりです。

表4-4 Wiresharkのフィルタとして使用できる比較演算子

演算子	説明
==	等しい
!=	等しくない
>	大なり
<	小なり
>=	以上
<=	以下

論理演算子を使えば、複数のフィルタ式を1つの文にすることができます。論理演算子を使いこなすことができれば、フィルタの効率が飛躍的に増加します。たとえば、2つのIPアドレスを含むパケットを表示したい場合を考えてみましょう。この場合は、「or」演算子を使って次のようにどちらかのIPアドレスを含むパケットを表示する式を作ればよいのです。

```
ip.addr==192.168.0.1 or ip.addr==192.168.0.2
```

Wiresharkで使用可能な論理演算子は表4-5のとおりです。

表4-5 Wiresharkのフィルタとして使用できる論理演算子

演算子	説明
and	論理積
or	論理和
xor	排他的論理和
not	否定

4.5.2.3 ディスプレイフィルタの例

フィルタの概念は難しくありませんが、実際にフィルタを作成する際には、どんなキーワードや演算子を使ったらよいか悩むところでしょう。表4-6に筆者がよくに使うディスプレイフィルタのいくつかを示します。一覧についてはWiresharkのディスプレイフィルタのリファレンス <http://www.wireshark.org/docs/dfref/> を参照してください。

表4-6 よく使用されるディスプレイフィルタ

フィルタ	説明
!tcp.port==3389	RDPトラフィックを消去する
tcp.flags.syn==1	SYNフラグがセットされたTCPパケット
tcp.flags.rst==1	RSTフラグがセットされたTCPパケット
!arp	ARPトラフィックを消去する
http	HTTPトラフィック
tcp.port==23 tcp.port 21	平文の管理用トラフィック (Telnet、FTP)
smtp pop imap	平文の電子メールトラフィック (SMTP、POP、IMAP)

4.5.3 フィルタの保存

キャプチャフィルタやディスプレイフィルタを山のように作っていると、頻繁に使うフィルタがあることに気づくことでしょう。幸い同じフィルタを何度も作成する必要はありません。Wiresharkには、フィルタを保存する機能があるのです。独自に作成したキャプチャフィルタを保存するには、以下の手順に従ってください。

- 1. メニューから [Capture] → [Capture Filters] を選択し、[Capture Filter] ダイアログを開いてください。
- 2. ダイアログの左側にある [New] ボタンをクリックし、新たなフィルタを作成します。
- 3. [Filter Name] ボックスにフィルタ名を入力します。
- 4. [Filter String] ボックスにフィルタ式を入力します。
- 5. フィルタ式を入力したら、[Save] ボタンをクリックして保存します。

独自に作成したディスプレイフィルタを保存するには、以下の手順に従ってください。

- 1. メニューから [Analyze] → [Display Filters] を選択するか、[Packet List] ペインの上部にある [Filter] ボタンをクリックして、図4-17の [Display Filter] ダイアログを開きます。

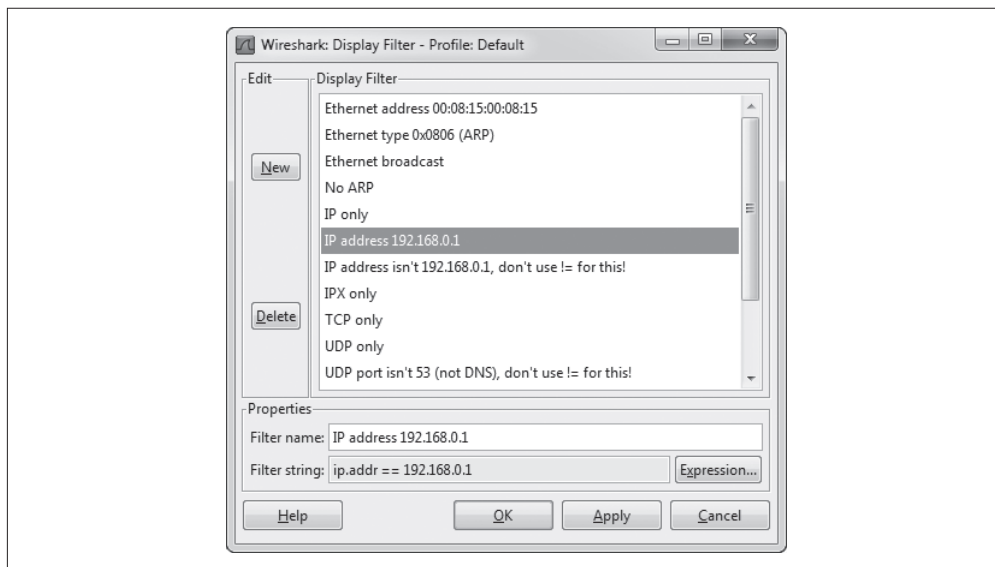


図4-17 [Display Filter] ダイアログからフィルタを保存できる

2. ダイアログの左側にある [New] ボタンをクリックし、新しいフィルタを作成します。
3. [Filter Name] ボックスにフィルタ名を入力します。
4. [Filter String] ボックスにフィルタ式を入力します。
5. フィルタ式を入力したら、[Save] ボタンをクリックして保存します。

Wiresharkには、定義済みのフィルタがいくつかありますが、これらはフィルタがどのようなものかを示す好例です。独自のフィルタを作成する際に、(Wiresharkのヘルプページと併せて) これらを活用できます。本書の例でもフィルタは何度も使われています。

5章

Wiresharkの高度な機能

Wiresharkの基本的な機能を習得したところで、次のステップとして解析とグラフ機能を習得しましょう。この章では、エンドポイントと「Conversation」ダイアログ、名前解決の詳細、プロトコルの分析、ストリームのフォロー、IO グラフなどを含む強力な機能の一端を見ていきます。

5.1 ネットワークのエンドポイントと対話

ネットワーク通信が行われるときは、最低でも2つの機器間でデータがやり取りされている必要があります。エンドポイントとは、ネットワーク上にあるデータを送受信する機器を示します。たとえばTCP/IP通信には、2つのエンドポイントがあります。192.168.1.25と192.168.1.30といった、データを送受信するホストのIPアドレスです。

仮に第2層で、2つの物理NICのMACアドレス間で通信が行われている場合を考えてみます。データを送受信するNICのアドレスが00:ff:ac:ce:0b:deと00:ff:ac:e0:dc:0fの場合、このアドレスがエンドポイントとなります（図5-1）。

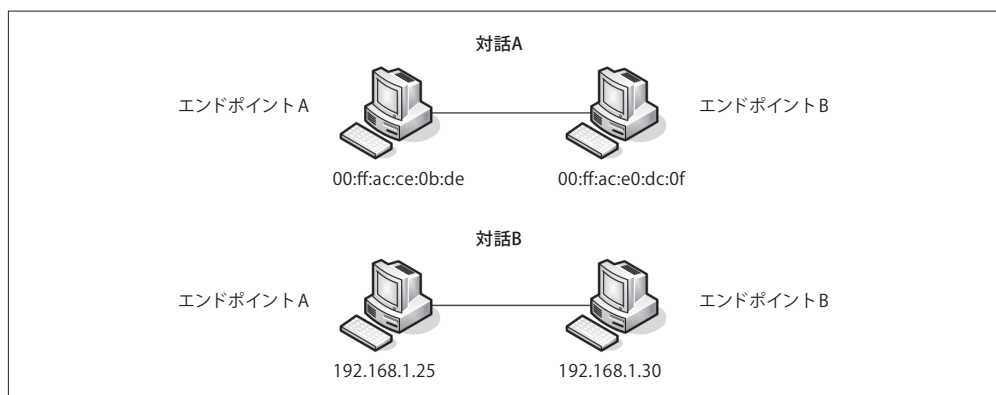


図5-1 ネットワークのエンドポイント

ネットワーク上の「対話」は、人の会話のように2台のホスト（エンドポイント）間で行われます。たとえば「やあ、元気?」「元気よ! あなたは?」「とても元気さ!」というジムとサリーの会話を、192.168.1.5のホストと192.168.0.8のホストの対話に置き換えると、「SNY」「SYN/ACK」「ACK」となります（TCP/IP通信についての詳細は6章で学びます）。

5.1.1 エンドポイントを参照する

トラフィックを解析する際に、ネットワーク上の特定のエンドポイントにトラブルを絞り込める場合も多いでしょう。メニューから [Statistics] → [Endpoints] を選択すると Wireshark の [Endpoints] ダイアログが表示され、各エンドポイントのアドレスや、送受信したパケット数、バイト数といった有用な情報を参照することができます（図5-2）。ダイアログ上部にある各タブでは、現在のキャプチャファイルにあるすべての認識可能なエンドポイントが表示されます。特定のプロトコルについてのエンドポイントを参照したい場合は、タブをクリックしてください。[Endpoints] ダイアログ内の [Name resolution] チェックボックスをオンにすると、名前解決が有効になります。

[Endpoints] ダイアログは、[Packet List] ペインで特定のパケットのみを表示するためのフィルタとして使用することもできます。エンドポイントを右クリックすると、選択したエンドポイントのみを含む、もしくは除外するトラフィックを表示するフィルタを作成するといったオプションがいくつか表示されます。また、選択したエンドポイントに色分けルールを直接適用することも可能です（色分けルールについては3章で説明しています）。

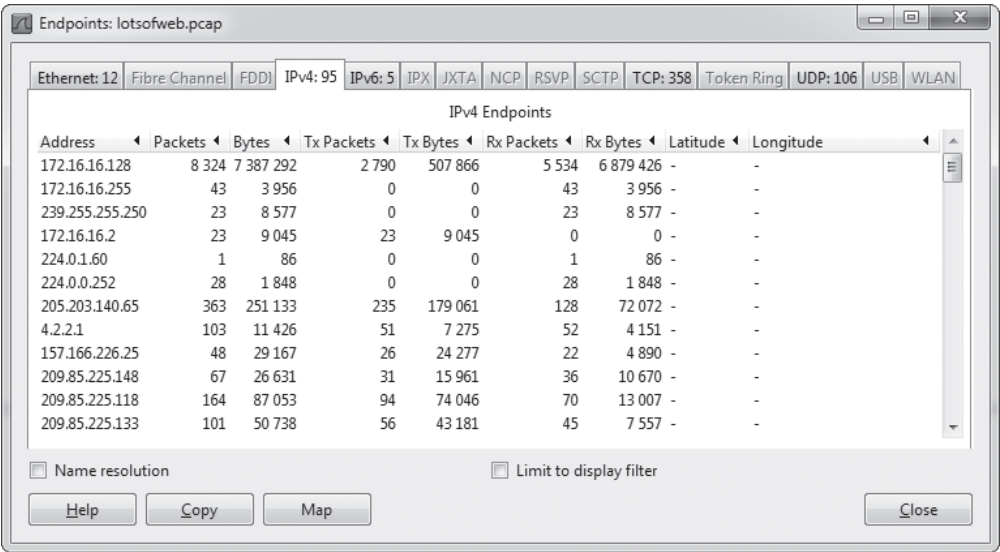


図5-2 [Endpoints] ダイアログからキャプチャファイル中の各エンドポイントを表示する

5.1.2 ネットワークの対話を見る

メニューから [Statistics] → [Conversations] を選択すると、図5-3のような [Conversations] ウィンドウが表示され、「Address A」と「Address B」という形式で対話が行われているエンドポイントのアドレスと、各ホストが送受信したパケット数、バイト数が表示されます。

対話はプロトコル別に表示されており、ウィンドウ上部にあるタブで切り替えることができます。対話を右クリックすると、エンドポイントAからのトラフィックのみを表示、エンドポイントBから受信したトラフィックのみを表示、エンドポイントAとB間のトラフィックのみを表示といった、フィルタを作成することもできます。

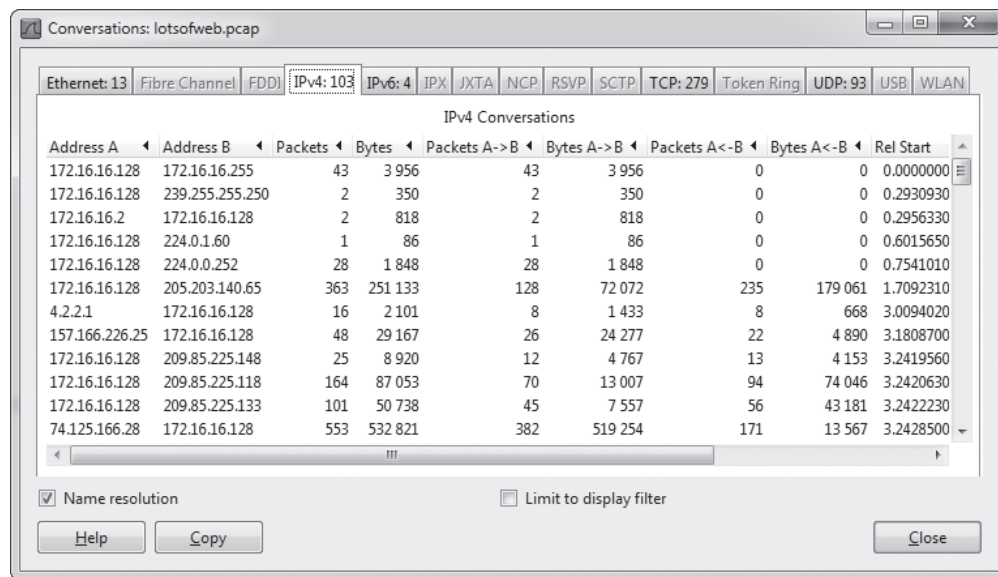


図5-3 [Conversations] ダイアログでそれぞれの対話を表示する

5.1.3 [Endpoints] と [Conversations] ダイアログを用いたトラブルシューティング

lotsofweb.pcap

トラブルシューティングにおいて、[Endpoints] ダイアログと [Conversations] ダイアログは非常に重要です。特にネットワーク上で大量のトラフィックの発生源を突き止めたり、通信量が多すぎるサーバを確認したりといった際には不可欠です。

たとえば、lotsofweb.pcap というファイルを開くと、いくつかのクライアントがインターネットをブラウジングしていることを示す、大量の HTTP トラフィックが目に入ります。[Endpoints] ウィンドウを開けば、現在目にしていてトラフィックが何かはすぐにわかるはずです。

IPv4 タブを見ると (図5-4)、バイト順にソートして最初に表示されるアドレスはローカルの 172.16.16.128 であり、これがネットワーク上でもっともおしゃべりな (通信量が多すぎる) 機器で

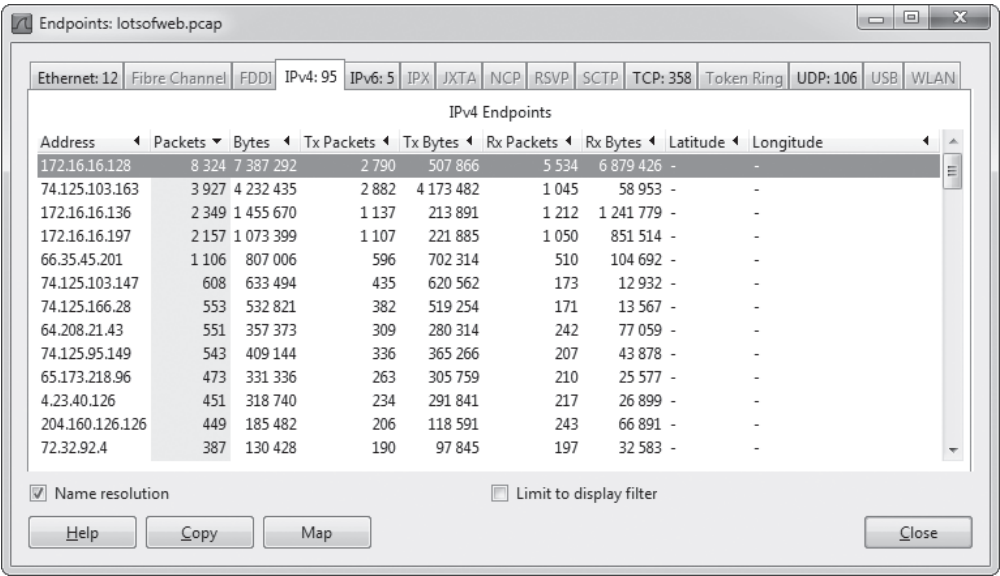


図5-4 [Endpoints] ダイアログでどのホストの通信量が一番多いかがわかる

あることがわかります。2番目のアドレスである74.125.103.163はローカルアドレスではないので、この段階では、あるクライアントがこのIPアドレスに大量のデータを送信しているのか、あるいは複数のクライアントからそれなりの量のデータが送信されているのかはわかりません。WHOIS検索 (<http://whois.arin.net/ui/>) をかければこのIPアドレスがGoogleのものであるとわかり、パケットを調べればこれがYouTubeのトラフィックであるとわかるでしょう。



IPアドレスの割り当ては、地域によって異なる組織（レジストリ）が管理します。上記の例ではAmerican Registry for Internet Numbers（ARIN）を使っていますが、ARINは米国および周辺国におけるIPアドレス割り当てを担当しています。基本的にIPアドレスを管理している組織のWebサイトで、IPのWHOIS検索を実行してください。地域がよくわからないときに、不適切なレジストリで検索を行うと、適切な地域が表示されます。アドレス登録を行っているレジストリには、ほかにAfriNIC（アフリカ）、RIPE（欧州）、APNIC（アジア太平洋）などがあります。

この情報だけで、一番通信を行っているエンドポイントが一番対話を行っていると考えても大丈夫でしょうか。今度は[Conversations]ウィンドウでIPv4タブを開き、バイト順のソートで確認してみましょう。そうするとトラフィックが動画のダウンロードと関連していることがわかります。アドレスA（74.125.103.163）から送信されているバイト数が、アドレスB（172.16.16.128）から送信されているバイト数よりもかなり多いからです（図5-5）。

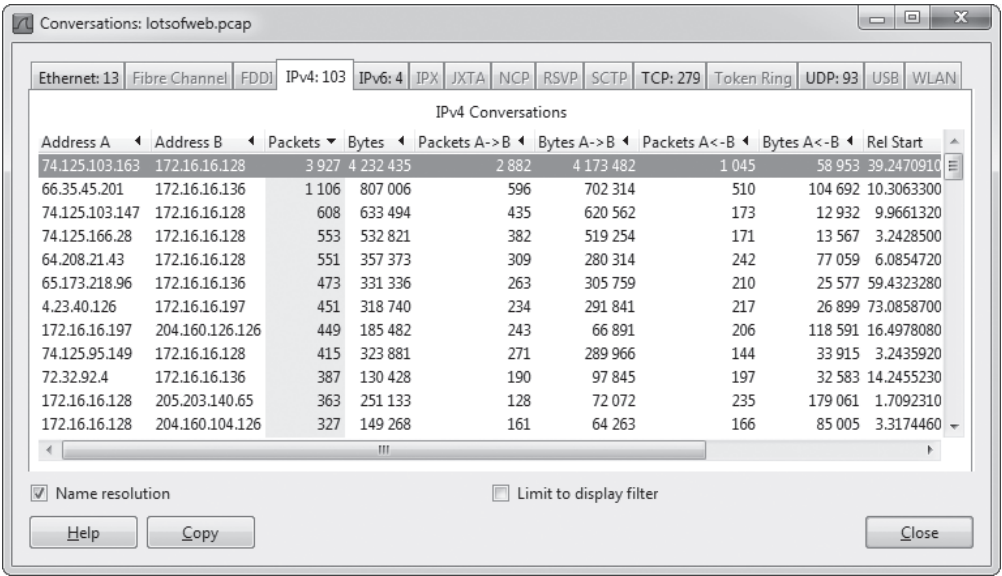


図5-5 [Conversations] ダイアログで、通信量が一番多いエンドポイント同士が通信していることがわかる

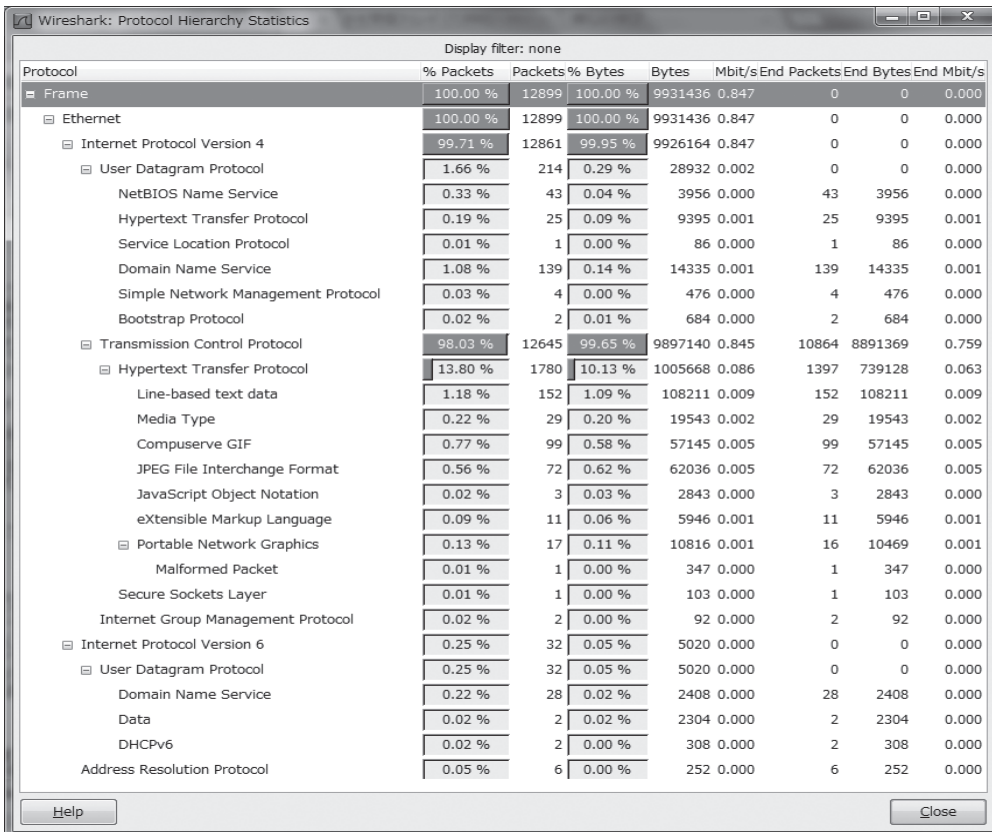
[Endpoints] および [Conversations] ダイアログを現場でどのように使うかは、後ほど説明します。

5.2 プロトコル階層統計

lotsofweb.pcap

巨大なキャプチャファイルを解析するときに、たとえばキャプチャしたパケットのうち何パーセントがTCPで、何パーセントがIP、何パーセントがDHCPかなど、各プロトコルがどのような配分になっているかを把握することが必要な場合があります。その際、パケットを1つ1つ数える必要はありません。[Protocol Hierarchy Statistics] ウィンドウを見ればよいのです。このウィンドウはネットワークのベンチマークを把握する際にとても役立ちます。たとえば普段はARPのトラフィックが全体の10%なのに、それが50%になっていたら、何か問題が起きていると予測できます。

lotsofweb.pcap ファイルを開いたら、メニューから [Statistics] → [Protocol Hierarchy] を選択して、[Protocol Hierarchy Statistics] ダイアログを開いてみましょう (図5-6)。パーセント値の合計は100%ちょうどにならない場合があります。多くのパケットにはさまざまな層でいくつかのプロトコルが含まれているため、パケット数の合計とプロトコルのパーセント値の合計は異なる場合があります。とはいえ、キャプチャファイルにおけるプロトコルの割合としては、かなり正確な値が確認できます。



Wireshark: Protocol Hierarchy Statistics

Display filter: none

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	12899	100.00 %	9931436	0.847	0	0	0.000
Ethernet	100.00 %	12899	100.00 %	9931436	0.847	0	0	0.000
Internet Protocol Version 4	99.71 %	12861	99.95 %	9926164	0.847	0	0	0.000
User Datagram Protocol	1.66 %	214	0.29 %	28932	0.002	0	0	0.000
NetBIOS Name Service	0.33 %	43	0.04 %	3956	0.000	43	3956	0.000
Hypertext Transfer Protocol	0.19 %	25	0.09 %	9395	0.001	25	9395	0.001
Service Location Protocol	0.01 %	1	0.00 %	86	0.000	1	86	0.000
Domain Name Service	1.08 %	139	0.14 %	14335	0.001	139	14335	0.001
Simple Network Management Protocol	0.03 %	4	0.00 %	476	0.000	4	476	0.000
Bootstrap Protocol	0.02 %	2	0.01 %	684	0.000	2	684	0.000
Transmission Control Protocol	98.03 %	12645	99.65 %	9897140	0.845	10864	8891369	0.759
Hypertext Transfer Protocol	13.80 %	1780	10.13 %	1005668	0.086	1397	739128	0.063
Line-based text data	1.18 %	152	1.09 %	108211	0.009	152	108211	0.009
Media Type	0.22 %	29	0.20 %	19543	0.002	29	19543	0.002
CompuServe GIF	0.77 %	99	0.58 %	57145	0.005	99	57145	0.005
JPEG File Interchange Format	0.56 %	72	0.62 %	62036	0.005	72	62036	0.005
JavaScript Object Notation	0.02 %	3	0.03 %	2843	0.000	3	2843	0.000
eXtensible Markup Language	0.09 %	11	0.06 %	5946	0.001	11	5946	0.001
Portable Network Graphics	0.13 %	17	0.11 %	10816	0.001	16	10469	0.001
Malformed Packet	0.01 %	1	0.00 %	347	0.000	1	347	0.000
Secure Sockets Layer	0.01 %	1	0.00 %	103	0.000	1	103	0.000
Internet Group Management Protocol	0.02 %	2	0.00 %	92	0.000	2	92	0.000
Internet Protocol Version 6	0.25 %	32	0.05 %	5020	0.000	0	0	0.000
User Datagram Protocol	0.25 %	32	0.05 %	5020	0.000	0	0	0.000
Domain Name Service	0.22 %	28	0.02 %	2408	0.000	28	2408	0.000
Data	0.02 %	2	0.02 %	2304	0.000	2	2304	0.000
DHCPv6	0.02 %	2	0.00 %	308	0.000	2	308	0.000
Address Resolution Protocol	0.05 %	6	0.00 %	252	0.000	6	252	0.000

図5-6 [Protocol Hierarchy Statistics] ダイアログでは各プロトコルの割合が表示される (バージョン1.8.0)

トラフィックを調査する際に最初に参照することの多いダイアログのひとつが [Protocol Hierarchy Statistics] ダイアログです。このダイアログを見れば、今ネットワークで発生している事象の全体像がよくわかるからです。トラフィックに精通するようになると、使われているプロトコルの割合を見るだけで、ネットワーク上のユーザーや機器の概況がわかるようになります。筆者自身、あるネットワークセグメントのトラフィックを見れば、たとえばICMPやSNMPなどの管理プロトコルが存在すればIT部門、SMTPトラフィックの量が多ければ受注処理部門、「World of Warcraft[†]」のトラフィックがある一帯は厄介なインターン用の区画といった具合に、そのネットワークセグメントがどこの部門のものかを即時に判断できることもしばしばです。

5.3 名前解決

ネットワークのデータは、00:16:CE:6E:8B:24という物理アドレスのように、覚えるには長すぎる英

[†] 監訳注：オンラインゲームのひとつ。

数字のアドレス体系を用いて通信されています。名前解決とは、あるプロトコルが既知のアドレスを別のアドレスに変換するために用いる処理のことです。たとえば00:16:CE:6E:8B:24というMACアドレスの機器は、DNSとARPプロトコルによってMarketing-2.domain.comという名前で参照が可能となったりします。暗号のようなアドレスを読みやすいアドレスに変換することによって、機器を識別しやすいようにするわけです。

5.3.1 名前解決を有効にする

名前解決を有効にするには、メニューから [Capture] → [Options] を選択して [Capture Options] ダイアログを表示してください。図5-7のように、Wiresharkには3種類の名前解決が可能です。

MAC層の名前解決

ARPプロトコルを使って、00:09:5B:01:02:03といった第2層のMACアドレスを10.100.12.1といった第3層のアドレスに変換します。変換できない場合、Wiresharkはプログラムの置かれたディレクトリにあるethersファイルを使って変換を試みます。それにも失敗すると、Netgear_01:02:03のようにMACアドレスの先頭3バイトをIEEEが定めたメーカー名に変換します。

ネットワーク層の名前解決

IPアドレス192.168.1.50といった第3層のアドレスを「MarketingPC1.domain.com」といった読みやすいDNS名に変換します。

トランスポート層の名前解決

ポート番号を名前に変換します。たとえば80番ポートを「http」という名前に変換します。

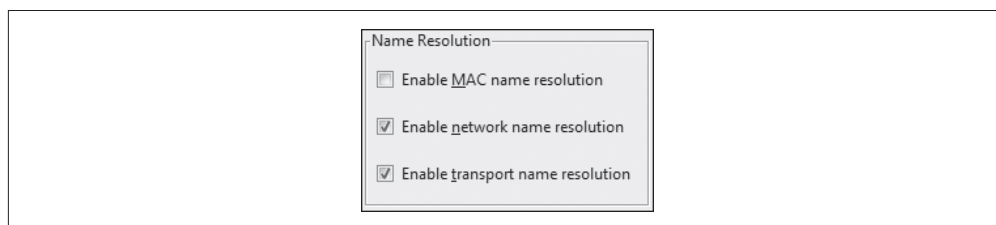


図5-7 [Capture Options] ダイアログで名前解決を有効にする

名前解決機能の力を借りてキャプチャファイルを読みやすくすることで、解析に要する時間を劇的に節約できる場合もあります。たとえばDNSの名前解決を使えば、パケットの送信元として特定したホストの名前を一瞬で確認することができます。

5.3.2 名前解決の欠点

名前解決はよいことづくしのように見えますが、以下のような欠点があります。

- クエリ先のネームサーバが名前を解決できないといった理由で、名前解決に失敗することがあります。
- 名前解決した情報はキャプチャファイルに保存されないため、キャプチャファイルを開くたびに名前解決が行われます。そのためキャプチャファイルを開いたときにネームサーバに接続できなければ名前解決できません。
- DNSに依存することになるため、余分なパケットが発生します。DNS名の名前解決トラフィックが発生するため、キャプチャファイルが見にくくなるかもしれません。経験上、トラブルを解析しているときには自分自身のトラフィックは見たくないものです。
- 名前解決のために余分な処理が発生します。巨大なキャプチャファイルを扱っていてメモリが不足しているときには、システムリソースを有効活用するために名前解決は行わないほうがよいでしょう。

5.4 プロトコル分析機構

Wiresharkでは、プロトコル分析機構 (protocol dissector) により、プロトコルをさまざまな要素に分解して解析可能な状態に整形しています。たとえば、ICMPのプロトコル分析機構は、Wiresharkがネットワークから取得した生データをICMPパケットとして整形して表示します。

分析機構は、ネットワークを流れる生のパケットとWiresharkとの間の翻訳機のようなものです。Wiresharkがプロトコルをサポートするには、そのプロトコルの分析機構がWiresharkに内蔵されている必要があります (あるいはC言語かPythonを使って自分で記述する必要があります)。Wiresharkは各パケットを分析する際に、いくつかの分析機構を並行して用います。使用する分析機構はプログラムのロジックによる推測で決定されます。

5.4.1 分析機構の変更

wrongdissector.pcap

残念ながら、Wiresharkがいつも正しい分析機構を選択するとは限りません。これは特に (ネットワーク管理者によってセキュリティ対策として設定されたり、従業員によってアクセス制限を迂回するために行われたりする) 非標準のポートの使用といった、一般的でない設定が行われている際に言えることです。幸いなことに、Wiresharkが選択した分析機構を変更することが可能です。

wrongdissector.pcapを開いてみましょう。このファイルには2台のホスト間での大量のSSL通信が記録されています。SSLはSecure Socket Layerプロトコルの略で、コンピュータ間での暗号化された安全な通信のために使用されます。普通の場合であれば、SSLは暗号化されているため、Wiresharkでトラフィックを参照してもあまり有益な情報は得られません。しかしながら何かがおかしいのは明らかです。パケットをクリックして [Packet Bytes] ペインを参照し、これらのパケットの内容をよく見ると、平文のトラフィックがあることにすぐに気づきます。4番目のパケットを見ると、FileZillaというFTPサーバのアプリケーションに関する文字列が存在していることがわかります。それに続くパケットでは、ユーザー名とパスワードに関するリクエストとレスポンスがはっきりと表示されています。

これが本当にSSLトラフィックであれば、パケットに含まれているデータは一切読めないはずで、送信されているユーザー名やパスワードも一切表示されないはずです(図5-8)。ここに示されている情報を見る限り、これはおそらくSSLトラフィックではなく、FTPトラフィックであると仮定して問題ないでしょう。このFTPトラフィックがHTTPS (HTTP over SSL) の標準である443番ポートを使っているために、このような事態が発生してしまったのです。

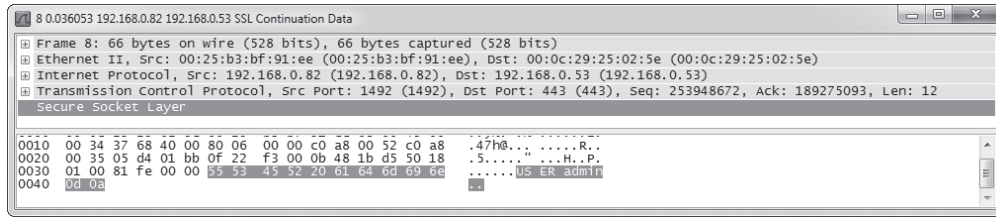


図5-8 平文のユーザー名とパスワード……これはSSLではなくFTPのようだ!

この問題を解決するには、FTPの分析機構を使うよう Wireshark に指示する必要があります。以下の手順に従って設定してください。

1. SSLパケットのひとつを右クリックして [Decode As] を選択します。どのプロトコルの分析機構を使うかを選択するダイアログが表示されます。
2. [Transport] タブでドロップダウンメニューから [destination(443)] を選択し[†]、[FTP] を選択します。これで、TCPの宛先ポートが443番のトラフィックでFTPの分析機構が利用されます(図5-9)。

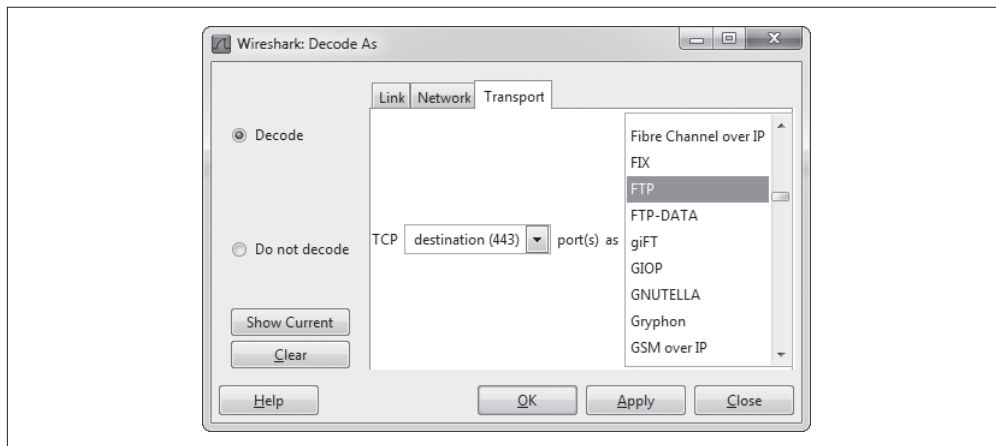
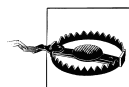


図5-9 [Decode As] ダイアログで分析機構を指定する

[†] 監訳注：手順1でサーバからクライアントへ向かうパケットを選択した場合は、ドロップダウンメニューから [source(443)] を選択します。

3. [OK] ボタンをクリックすれば、キャプチャファイルに即座に変更が適用されます。

データが適切に解析されており、逐一バイト列を確認しなくても、[Packet List] ペインを見るだけで、パケットを解析できるはずです。



分析機構の変更はキャプチャファイルには保存されず、Wiresharkを終了すると失われます。キャプチャファイルを開くたびに、分析機構を変更する必要があります。

同じキャプチャファイルで分析機構の変更を複数行ってもかまいません。キャプチャファイルで分析機構の変更をいくつも行ってしまうと、覚えておくのが大変になりますが、Wiresharkが代わりに覚えておいてくれます。[Decode As] ダイアログの[Show Current] ボタンをクリックすれば、今までに行った変更の一覧が表示されます(図5-10)。
[Clear] ボタンを押すことで、それらの変更を消去することも可能です[†]。

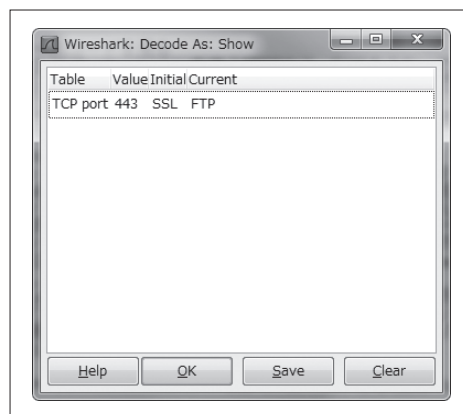


図5-10 [Show Current] ボタンを押すと、それまでに行った変更の一覧が表示される (バージョン1.8.0)

5.4.2 分析機構のソースコードを見る

オープンソースアプリケーションを使う醍醐味は、なぜそうなっているのかがわからなくなったとき、ソースコードを見てその原因を明確に確認できることです。これは、特定のプロトコルが間違っ

て分析されている理由を確認するときに特に便利です。

プロトコル分析機構のソースコードを確認するには、WiresharkのWebサイトで、[Develop] のリンクの上にマウスを持っていき、[Browse the Code] をクリックします。Wiresharkのsubversionリポジトリへと移動し、現バージョンのソースコードだけでなく、以前のバージョンのソースコードも見られます。[releases] フォルダをクリックすると、すべての公式なWireshark (およびEthereal) リリースの一覧が、古い順に表示されます。確認したいリリースを選択すると、epan/dissectors フォルダにプロトコル分析機構が見つかるはずです。各分析機構は「packets-プロトコル名.c」という名称になっています。

ファイルの中身はちょっと複雑ですが、いずれも標準的なテンプレートを採用しており、豊富なコメ

[†] 監訳注：本書監訳時点で最新版のバージョン1.8.0で確認したところ、図5-10のように[Save]というボタンが追加されていました。将来的に分析機構の変更のインポート、エクスポートを実現する構想だと思われますが、本書監訳時点ではこのボタンは機能していないようで、押しても何も動作しませんでした。

ントがついていることに気づくでしょう。C言語に詳しくなくても、分析機構の基本的な機能は理解できます。Wiresharkで参照できる内容を深く理解したいならば、単純なプロトコルでよいので、最低限分析機構をきちんと理解しておくことをお勧めします。

5.5 TCPストリームの表示

http_google.pcap

Wiresharkのとても便利な機能のひとつとして、TCPストリームを読みやすい形式に整形してくれる機能があります。クライアントからサーバに送信されたデータを大量の小さいデータとして見る代わりに、[Follow TCP Stream] 機能がデータをまとめて見やすくしてくれます。これは、HTTPやFTPなどの平文のアプリケーション層プロトコルを参照する際に特に有益です（こうした一般的なプロトコルの動作については次章で詳しく見ていきます）。

簡単なHTTPトランザクションを考えてみましょう。http_google.pcap ファイルを開き、ファイル内のTCPまたはHTTPパケットのどれかをクリックし、ファイルを右クリックして、[Follow TCP Stream] を選択します。するとTCPストリームが新しいダイアログに表示されます（図5-11）。

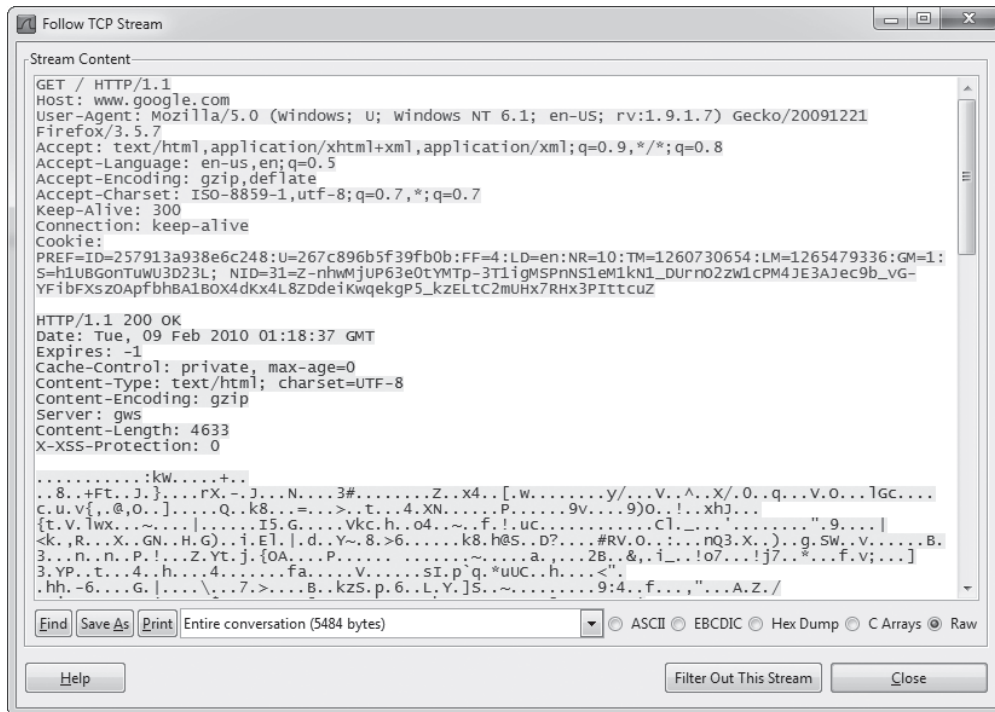


図5-11 読みやすい形式で通信を表示する [Follow TCP Stream] ダイアログ

このダイアログ内のテキストは2色に色分けされています。赤は送信元から宛先へのトラフィック、青は反対方向、つまり宛先から送信元へのトラフィックを示しています。どちらから発信されたかで

色分けされているのです。この例では、クライアントがWebサーバに対してコネクションを開始しているため、赤で表示されています。

このTCPストリームでは、2台のホスト間での通信のほとんどを見ることができます。通信はWebのルートディレクトリ (/) へのGETリクエストで始まり、サーバはリクエストを受け取ったことをHTTP/1.1 200 OKという形式で返しています。以降では、クライアントがファイルをリクエストし、サーバがそれに対して応答するといった、同様のパターンがストリーム上で繰り返されています。これはユーザーが実際にGoogleホームページをブラウズしているところですが、ユーザーが見ているものを、内側から見ているのです。

TCPストリームは、画面上で生のデータとして見るだけでなく、テキスト内を検索したり、テキストファイルとして保存または印刷したりすることができます。また文字列をASCII、EBCDIC、16進数、C言語の配列に変換することも可能です。これらのオプションは[Follow TCP Stream] ダイアログの下部にあります。

TCPストリームの表示は、特定のプロトコルを扱う際に非常に役立つでしょう。

5.6 パケット長

download-slow.pcap

単一の、もしくはいくつかのパケット群のサイズからさまざまなことがわかります。一般的な状況であれば、イーサネットの最大フレームサイズは1518バイトです。この数値からイーサネット、IP、TCPのヘッダを差し引いた1460バイトを第7層のプロトコルヘッダおよびデータのやり取りに使うことができます。これを頭に入れたうえで、パケット長の分布から、キャプチャしたトラフィックについて推測してみましょう。download-slow.pcap ファイルを開き、メニューから[Statistics] → [Packet Lengths] を選択して、[Create Stat] ボタンをクリックすると、図5-12のようなダイアログが表示されます。

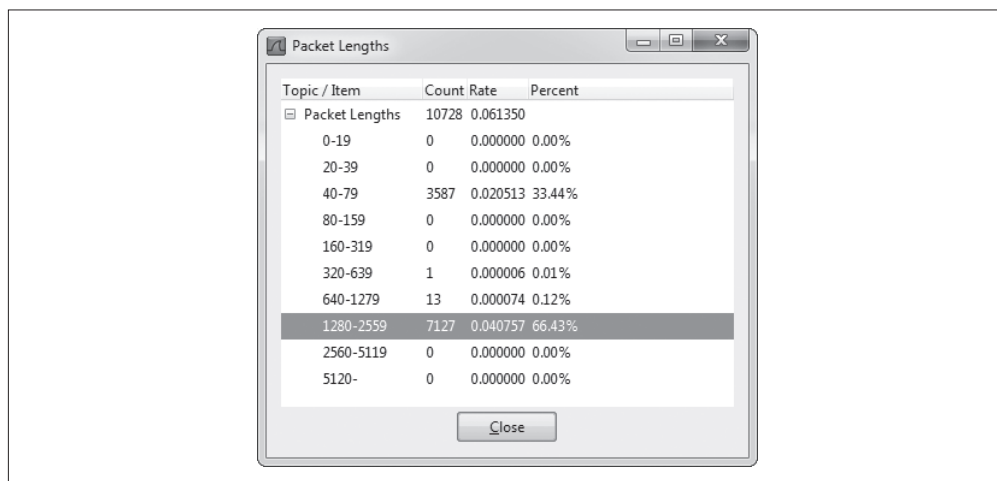


図5-12 キャプチャファイルのトラフィックの推測に役立つ [Packet Lengths] ダイアログ

ここではサイズが1280バイトから2559バイトのパケットの統計を示している部分をハイライトしています。サイズの大きなパケットは通常データ転送を、小さなパケットはプロトコルのシーケンス制御を示しています。ここでは、大きなパケットの比率がかなり大きくなっています(66.43%)。そのためファイルの中のパケットを見なくても、キャプチャファイルには1つ以上のデータ転送が含まれていることがわかります。つまりこれはHTTPダウンロードかFTPアップロード、あるいはその他のホスト間でデータ転送が行われる通信だということになります。

残りのパケットの大半(33.44%)は、サイズが40バイトから79バイトです。このサイズのパケットは、通常データを含まないTCP制御パケットです。一般的なプロトコルヘッダのサイズを考えてみましょう。イーサネットヘッダは14バイト(プラス4バイトのCRC)、IPヘッダは最少で20バイト、そしてデータやオプションのないTCPヘッダも20バイトです。つまり標準的なTCP制御パケット(SYN、ACK、RST、FINなど)は54バイト程度であり、この範囲に収まることになります。もちろんIPやTCPオプションが付加されていればサイズは増えます。

パケット長を調べると、キャプチャの全体像がよくわかります。サイズの大きなパケットがたくさんあれば、データが転送されていると考えてよいでしょう。またパケットの大半が小さいサイズなら、データ転送があまり行われていない、プロトコル制御命令で構成されていると考えられます。これは確実な規則ではありませんが、さらに細かい分析を行う前に、こうした仮説を立てておくとうまくいく場合があります。

5.7 グラフ表示

グラフは分析の基本であり、データの概要を把握する最適な方法のひとつです。Wiresharkではキャプチャしたデータの把握を助けるグラフ機能がいくつかありますが、そのひとつがIOグラフ機能です。

5.7.1 IOグラフを見る

`download-fast.pcap` `download-slow.pcap`

Wiresharkでは、[IO Graphs] ウィンドウで送受信されているデータのスループットをグラフ化することができます。ここではデータのスループットにあるスパイクや小康状態を確認したり、各プロトコルのパフォーマンス遅延を確認したり、複数のデータストリームを同時に比較したりすることが可能です。

IOグラフの例として、インターネットからのファイルのダウンロードを見てみましょう。`download-fast.pcap`を開いて、TCPパケットのどれかをクリックしてハイライトし、[Statistics] → [IO Graphs]を選択します。

[IO Graphs] ダイアログでは、キャプチャファイル中のデータの流れを、グラフとして見るができます。図5-13の例では、ダウンロードにより平均500パケット/秒の転送が継続的に続き、最後に減少していることがグラフからわかります。

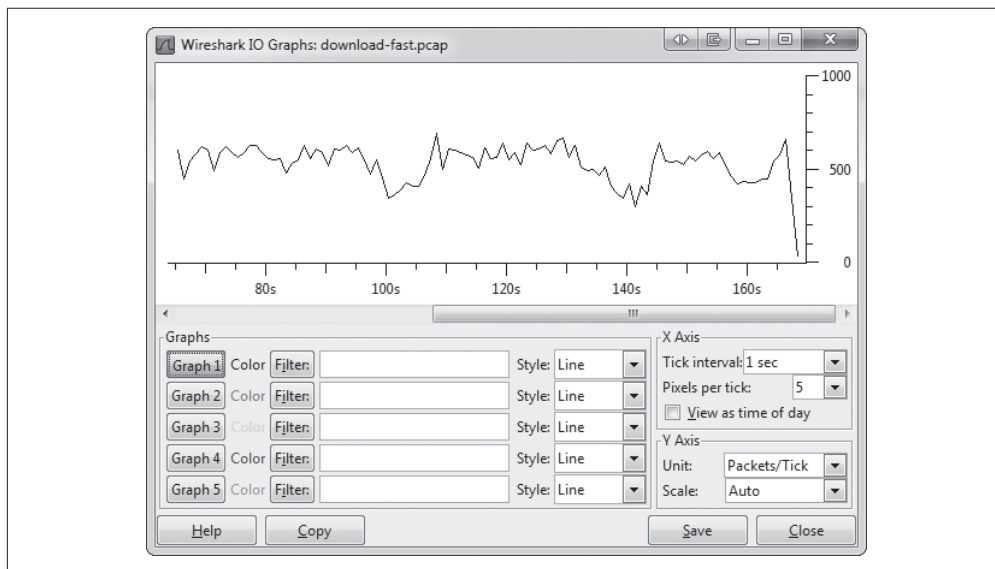


図5-13 安定した高速ダウンロードを示すIOグラフ

これを速度の遅いダウンロードのものと比較してみましょう。現在のファイルを開いたまま、Wiresharkの別のインスタンスを開いて、download-slow.pcapを開きます。このダウンロードをIOグラフにしてみると、図5-14のように違いがはっきりとわかります。

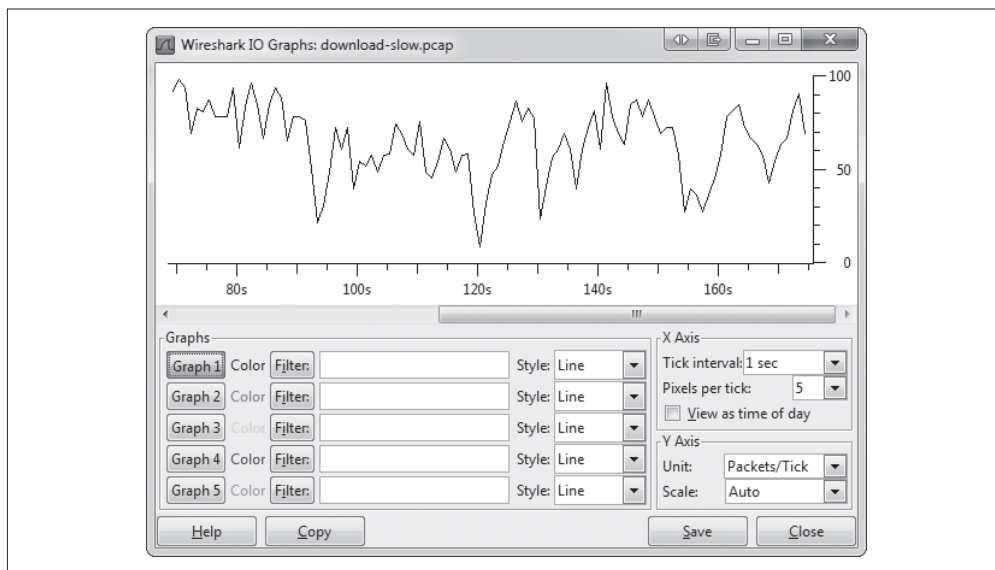


図5-14 安定していない遅いダウンロードを示すIOグラフ

このダウンロードのデータ転送レートは0から100パケット/秒で、安定からは程遠く、1秒当たりのパケット数がほぼ0になっているときさえあります。2つのダウンロードのグラフを並べてみると、その不安定さは一目瞭然です(図5-15)。

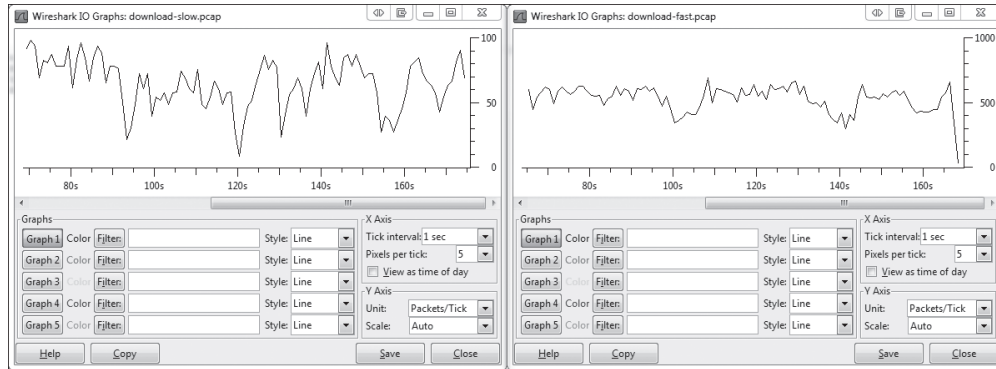


図5-15 複数のIOグラフを並べると違いの識別に役立つ

ダイアログの下部にいくつか設定オプションが存在します。最大5つまでのフィルタを作成でき(6章と7章で説明しますが、表示やキャプチャでも同じ構文が使えます)、フィルタの色分けができます。たとえば、ARPを赤、DHCPを青で表示するフィルタを作成して、赤と青の折れ線グラフで表示すれば、2つのプロトコルのスループットを簡単に見分けることが可能です。

5.7.2 ラウンドトリップタイムグラフ

download-fast.pcap

Wiresharkには、キャプチャファイルのラウンドトリップタイムを表示するグラフ機能も備わっています。ラウンドトリップタイム(RTT)とは、パケットの受信が確認されるまでにかかる時間を指します。これは、パケットが宛先に届き、その返答が自分に戻ってくるのにかった時間です。RTTの分析は、通信の遅延が発生したポイントやボトルネックを見つけて、遅延の有無を確認するためによく行われます。

この機能を使ってみましょう。download-fast.pcap ファイルを開いて、TCPパケットのいずれかを選んでから、メニューから [Statistics] → [TCP Stream Graph] → [Round Trip Time Graph] を選択すると、図5-16のようなRTTグラフが表示されます。

グラフのそれぞれの点が各パケットのRTTを表しています。デフォルトではシーケンス番号によってソートされた値が表示されます。グラフ内の点をクリックすると、[Packet List] ペインの該当パケットに直接移動できます。

RTTグラフを見ると、高速ダウンロードのRTT値はほぼ0.05秒以下で、やや遅い場合でも0.10秒から0.25秒程度であることがわかります。許容範囲を超えている値もわずかにあるものの、RTT値のほとんどは問題がないので、ファイルのダウンロードとしては問題ないRTTとみなせるでしょう。

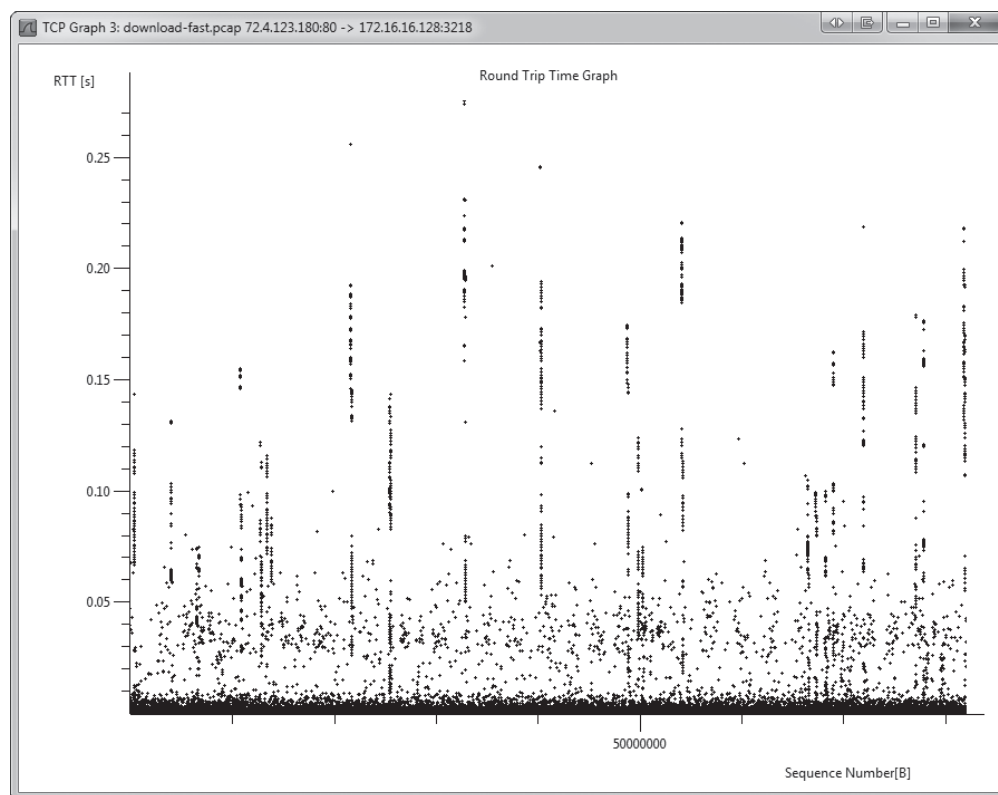


図5-16 このダウンロードのRTTグラフはほぼ安定しており、値の分散はわずしか見られない

5.7.3 フローグラフ

http_google.pcap

コネクションを視覚化して時間経過に伴うデータの流れを示すものとして、フローグラフ機能は非常に便利です。フローグラフは、基本的にホスト間のコネクションを示すカラムビューとなっており、視覚的にトラフィックを把握できるように、トラフィックが整理されています。

フローグラフを作成するには、http_google.pcap ファイルを開き、[Statistics] → [Flow Graph] を選択します。処理対象のパケットとフローのタイプについてのオプションを選択する小さなダイアログが表示されますが、このサンプルではデフォルトのままで問題ないので、[OK] ボタンをクリックしてフローグラフを作成します (図5-17)。

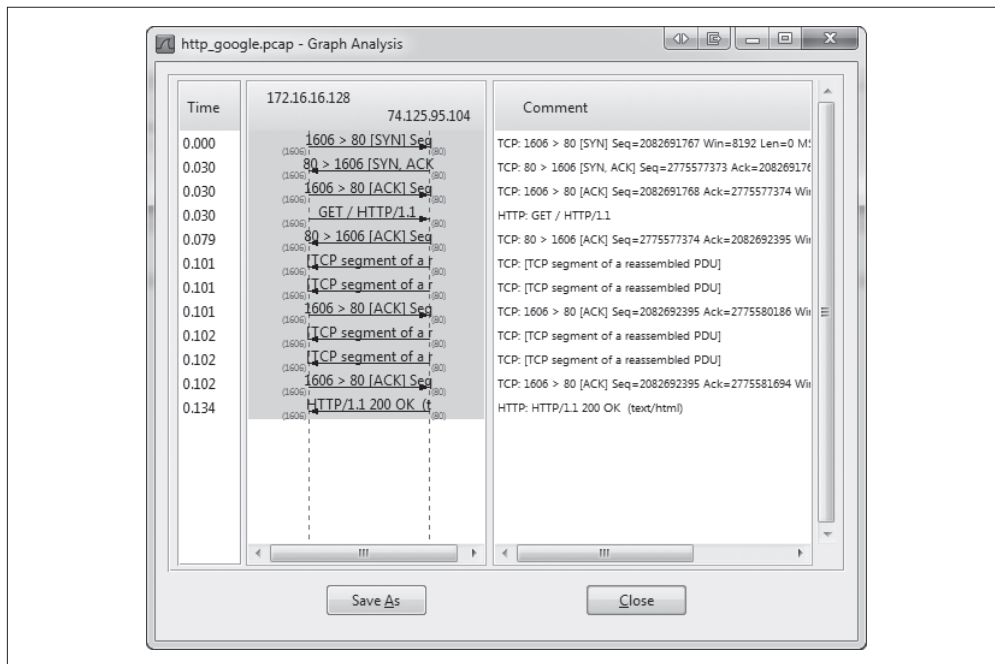


図5-17 TCPフローグラフで接続を視覚化できる

5.8 エキスパート情報

download-slow.pcap

Wiresharkの各プロトコル分析機構では、そのプロトコルのパケットが特定の状態となった際に警告を挙げるために用いられる「エキスパート情報」を設定することができます。状態は以下に4つに分けることができます。

- **Chat** 通信の基本情報
- **Note** 通常の通信の一部であると考えられる異常なパケット
- **Warning** 通常の通信の一部であるとは考えられない異常なパケット
- **Error** パケットもしくはそれを解析した解析機構でエラーが発生したもの

サンプルを見てみましょう。download-slow.pcap ファイルを開き、[Analyze] → [Expert Info Composite] を選択して[†]、[Expert Infos] ダイアログを表示します (図5-18)。

[†] 監訳注：バージョン1.8.0では [Expert Info] を選択します。

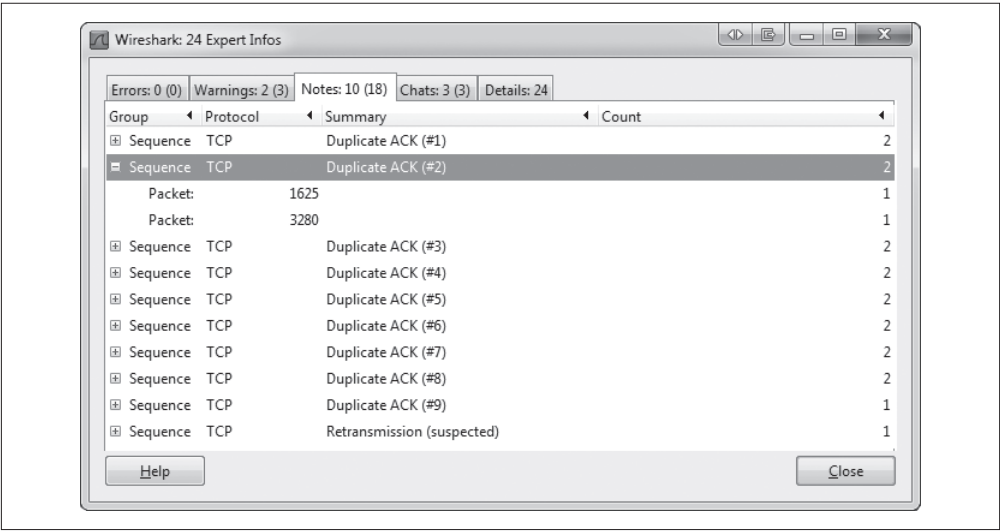


図5-18 [Expert Infos] ダイアログはプロトコル解析機構によるエキスパートシステムからの情報を表示する

ダイアログには分類された状態ごとにタブがあり、Errorはゼロ、Warningが3、Noteが18、Chatが3あることがわかります。タブに示されたカッコの外の数字は重複しないメッセージの件数、カッコ内の数字は重複するものも含めたメッセージの件数を示しています。

このキャプチャファイルのメッセージがすべてTCP関連となっているのは、執筆時点では、TCP以外のプロトコルにエキスパート情報システムが導入されていなかったためです。執筆時点ではTCP関連で14個のエキスパート情報メッセージが存在し、トラブルシューティングの際に非常に有用です。これらのメッセージは、以下に示す条件を満たした際に、個々のパケットにフラグを立ててくれます。

Chatメッセージ

Window Update

TCP受信ウィンドウのサイズが変更されたことを送信者に通知するために、受信者によって送信される

Noteメッセージ

TCP Retransmission

パケット消失の結果として発生する。ACKが重複して受信された、またはパケットの再送タイマーがタイムアウトとなった場合に発生する

Duplicate ACK

ホストが次に期待するシーケンス番号を受け取らなかった場合、最後に受信したデータの重複ACKを生成する

Zero Window Probe

ゼロウィンドウパケットが送信されたあとに、TCP受信ウィンドウの状態を監視するのに使われる（9章で説明）

Keep Alive ACK

キープアライブパケットに応答が送信された

Zero Window Probe ACK

Zero Window Probeパケットに応答が送信された

Window is Full

受信者のTCP受信ウィンドウがいっぱいの状態であることを、送信元のホストに知らせる際に使われる

Warningメッセージ

Previous Segment Lost

パケット消失を意味する。データストリームにおいて期待するシーケンス番号がスキップされた場合に発生する

ACKed Lost Packet

ACKパケットによって送達確認されたはずのパケットが到着していない場合に発生する

Keep Alive

コネクションのキープアライブパケットが確認された

Zero Window

TCP受信ウィンドウのサイズが一定値に達し、Zero Window通知が送られて、送信者にデータ送信を停止するよう要求された場合に発生する

Out-of-Order

シーケンス番号により、受信したパケットのシーケンスの乱れが検知された

Fast Retransmission

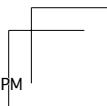
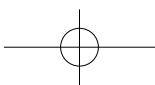
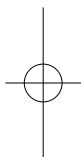
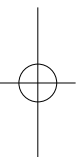
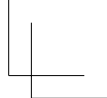
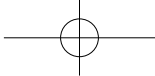
重複ACKから20ミリ秒以内に発生した再送信

Errorメッセージ

特になし

6章でTCPを学習し、9章で遅いネットワークのトラブルシューティングについて学べば、これらのメッセージの意味が明確になるでしょう。

本章で説明したいいくつかの機能は、滅多に起こらない状況でしか使われないように見えるかもしれませんが、想像以上に利用することになると思います。ここで説明したダイアログとオプションに慣れておいてください。以降の章で何度も触れることになります。



6章

知っておきたい下位層プロトコル

遅延に関するトラブルをトラブルシューティングするとき、うまく機能してしないアプリケーションを確認するとき、あるいは異常なトラフィックを発見するためにセキュリティ上の脅威に注力するときなど、まずは正常時のトラフィックの理解が不可欠です。ここからの数章では、正常時のネットワークトラフィックについてパケットレベルで見えていきます。働き者のTCP、UDP、IPといった最低限知っておきたいプロトコル、またHTTP、DHCP、DNSなどのよく使われるアプリケーション層プロトコルを見ていきましょう。それぞれのプロトコルのセクションでは関連するキャプチャファイルを最低ひとつは紹介しているので、ダウンロードして直接いじってみることができます。本章では特に、OSI参照モデルの第1層から第4層に該当する下位層プロトコルに焦点を当てています。

これらの章は、間違いなく本書においてもっとも重要な章です。ここを読み飛ばすのは、コーンブレッドなしに日曜日の夕食を作るようなものです。各プロトコルの機能について、すでにきちんと理解している場合でも、最低限さっと目を通して、それぞれのパケットの構造を見直しておきましょう。

6.1 ARP (Address Resolution Protocol)

ネットワーク通信では、論理アドレスおよび物理アドレスの両方が使われます。論理アドレスを用いることで、ネットワーク越しに直接接続されていない機器間での通信が可能になります。物理アドレスは、単一のネットワークセグメント上でスイッチを通じて直接つながっている機器間の通信に使われます。たいていの場合、通信を行うには、この両方のアドレスが連携して動作する必要があります。

ネットワーク上のある機器と通信したいとしましょう。その機器はサーバかもしれませんし、ファイルを共有したいワークステーションかもしれません。通信を始めようとしているアプリケーションは、すでにリモートのホストのIPアドレスを（7章で説明するDNSによって）認識しており、送信したいパケットの第3層から第7層を構築するのに必要な情報をすべて持っているものとします。ここでさらに必要な情報は、宛先ホストのMACアドレス（物理アドレス）を含む、第2層（データリンク層）のデータだけです。

MACアドレスが必要なのは、ネットワーク上の機器をつないでいるスイッチが、そのポートに接続しているすべての機器のMACアドレスを格納する**CAM**（Content Addressable Memory）テーブルを使っているためです。スイッチは、特定のMACアドレスに対するトラフィックを受信すると、このテーブルを使ってどのポートにトラフィックを送信するかを判断します。宛先のMACアドレスが不明な場合、送信元の機器は、まず自身のキャッシュにアドレスがないかを調べます。キャッシュに存在しない場合は、ネットワークに対してさらなる通信を行うことで解決することが必要になります。

TCP/IPネットワーク（IPv4）において、IPアドレスからMACアドレスを求めるのに使われるプロトコルを**ARP**（Address Resolution Protocol）と呼び、これはRFC 826で定義されています。ARPは、ARPリクエストとARPレスポンスの2種類のパケットしか使いません（図6-1）。

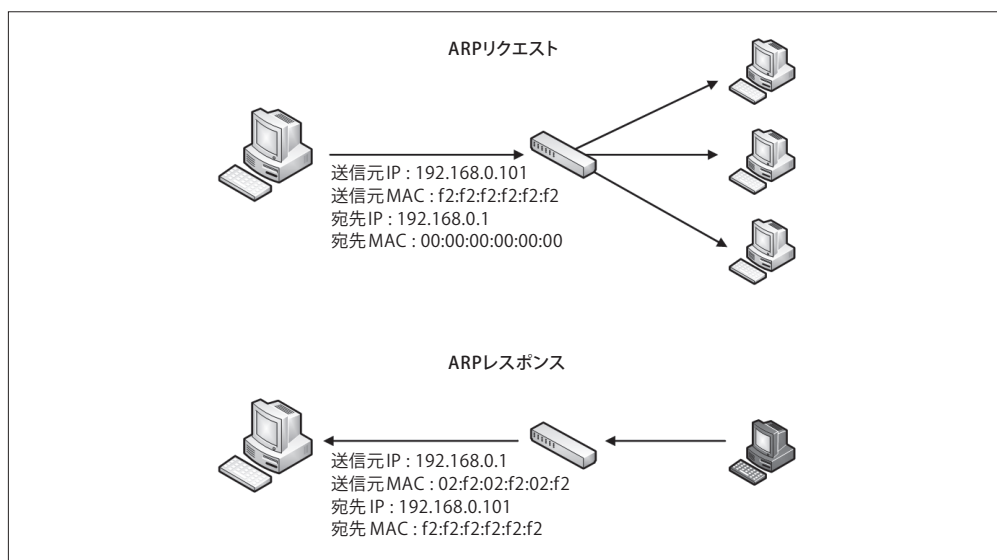


図6-1 ARP処理



RFC (Request for Comments) とは、プロトコルの標準的な実装について定義している公的なドキュメントのことです。RFCのWebサイト<http://www.rfc-editor.org>で、RFCを検索することができます。

送信元のホストが、「皆さんこんにちは、私のIPアドレスはXX.XX.XX.XXで、MACアドレスはXX:XX:XX:XX:XX:XXです。IPアドレスがXX.XX.XX.XXの人に送信するものがあるのですが、MACアドレスがわかりません。このIPアドレスを持っている方、MACアドレスを返信してくれませんか?」というARPリクエストを送ったとしましょう。

このパケットはネットワークセグメント上のすべての機器に送られますが、このIPアドレスを持たない機器はパケットを単に廃棄します。このIPアドレスを持つ機器だけが、「送信元さん、あなた

が探しているIPアドレスXX.XX.XX.XXを持っているのは私です。私のMACアドレスはXX:XX:XX:XX:XX:XXです」という答えのARPレスポンスを送ります。

この処理が完了すると、送信元はこの機器のMACおよびIPアドレスの情報でキャッシュを更新し、データ送信を開始できる状態になります。



コマンドプロンプトで「arp -a」と入力すると、WindowsのARPテーブルを参照することができます。

この処理が実際に行われているところを見れば、どう機能するかを理解する助けになると思います。が、実例を見る前に、まずはARPパケットヘッダについて説明します。

6.1.1 ARPヘッダ

ARPヘッダには、図6-2に示しているようなフィールドが含まれています。

ARP (アドレス解決プロトコル)		
ビットオフセット	0-7	8-15
0	ハードウェアタイプ	
16	プロトコルタイプ	
32	ハードウェアアドレス長	プロトコルアドレス長
48	オペレーション	
64	送信元ハードウェアアドレス (先頭の16ビット)	
80	送信元ハードウェアアドレス (第2の16ビット)	
96	送信元ハードウェアアドレス (第3の16ビット)	
112	送信元プロトコルアドレス (先頭の16ビット)	
128	送信元プロトコルアドレス (第2の16ビット)	
144	宛先ハードウェアアドレス (先頭の16ビット)	
160	宛先ハードウェアアドレス (第2の16ビット)	
176	宛先ハードウェアアドレス (第3の16ビット)	
192	宛先プロトコルアドレス (先頭の16ビット)	
208	宛先プロトコルアドレス (第2の16ビット)	

図6-2 ARPパケットの構造

ハードウェアタイプ

第2層のタイプ。たいていの場合、イーサネット（タイプ1）です。

プロトコルタイプ

ARPリクエストが使われる上位層のプロトコル。

ハードウェアアドレス長

ハードウェアアドレスの長さ（オクテット／バイト）。イーサネットの場合は6。

プロトコルアドレス長

指定されたプロトコルタイプの論理アドレスの長さ（オクテット／バイト）。

オペレーション

ARPパケットの機能。リクエストの場合は1、レスポンスの場合は2。

送信元ハードウェアアドレス

送信元のハードウェアアドレス。

送信元プロトコルアドレス

送信元の上位層のプロトコルアドレス。

宛先ハードウェアアドレス

宛先のハードウェアアドレス（ARPリクエストの場合は0）。

宛先プロトコルアドレス

宛先の上位層のプロトコルアドレス。

arp_resolution.pcap ファイルを開き、この処理を見てみましょう。個々のパケットをひとつずつ見ていきます。

6.1.2 パケット1：ARPリクエスト

arp_resolution.pcap

最初のパケットはARPリクエストです（図6-3）。[Packet Details] ペインでイーサネットヘッダを調べれば、このパケットが本当にブロードキャストパケットであることを確認できます。宛先のアドレスはff:ff:ff:ff:ff:ff^①です。これはイーサネットのブロードキャストアドレスなので、この宛先に送信されたパケットは、現在のネットワークセグメント上にあるすべての機器にブロードキャストされます。イーサネットへの中にある送信元アドレスはMACアドレスとして認識されます^②。

構造から見て、これは確かにIPを使っているイーサネットネットワーク上のARPリクエストであることがわかります。送信元のIPアドレス（192.168.0.114）とMACアドレス（00:16:ce:6e:8b:24）^③が、宛先のIPアドレス（192.168.0.1）^④とともに表示されています。宛先のMACアドレス（入手しようとしている情報）は不明なので、宛先のMACアドレスは00:00:00:00:00:00 となっています^⑤。

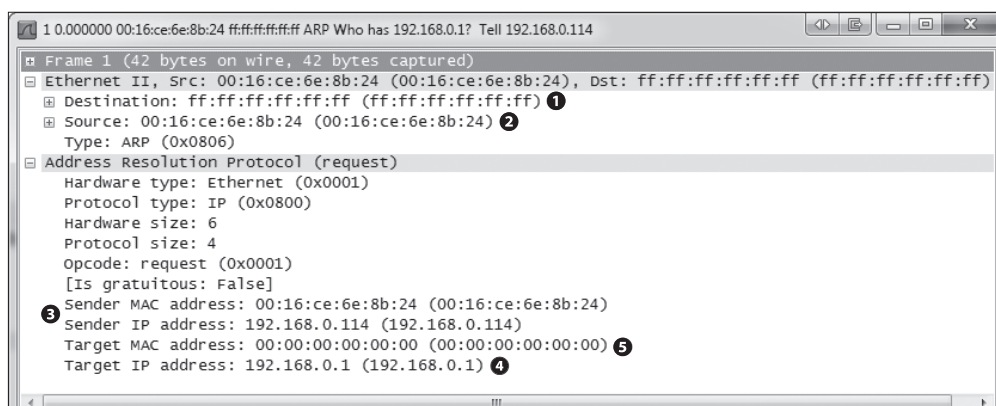


図6-3 ARPリクエストパケット

6.1.3 パケット2：ARPレスポンス

最初のリクエストに対するレスポンス (図6-4) では、イーサネットヘッダに、最初のパケットの送信元MACアドレスが宛先アドレスとして格納されています。

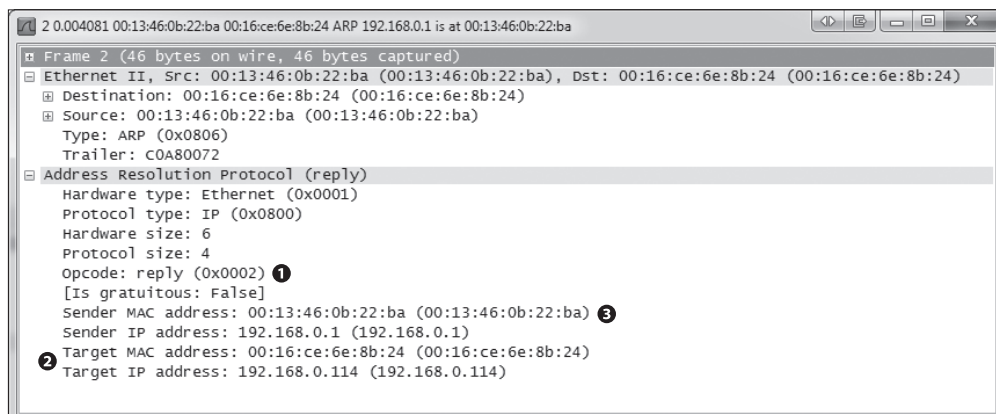


図6-4 ARPレスポンスパケット

このARPレスポンスのヘッダはARPリクエストのヘッダとよく似ていますが、いくつかの違いがあります。

- パケットのオペレーションコード (opcode) が0x0002となり①、リクエストではなくレスポンスであることを示しています。
- アドレス情報が入れ替わっており、送信元MACアドレスとIPアドレスが、宛先のMACアドレスとIPアドレスになっています②。
- 何よりも重要なこととして、IPアドレス192.168.0.1のホストのMACアドレス (00:13:46:0b:22:

ba) ③が格納されています。

6.1.4 gratuitous ARP

arp_gratuitous.pcap

何かが「勝手に (gratuitously)」行われるというのは、否定的な意味合いを持つことがよくあります。しかしながら gratuitous ARP (勝手な ARP) は意味のあるものです。

IPアドレスは変更される場合がよくあります。IPアドレスが変わると、キャッシュに記憶されたIPアドレスとMACアドレスとのマッピング情報が不正となります。これによる通信のトラブルを防ぐため、gratuitous ARPパケットが送信され、受け取った機器のキャッシュを新しいマッピング情報で強制的に更新します (図6-5)。

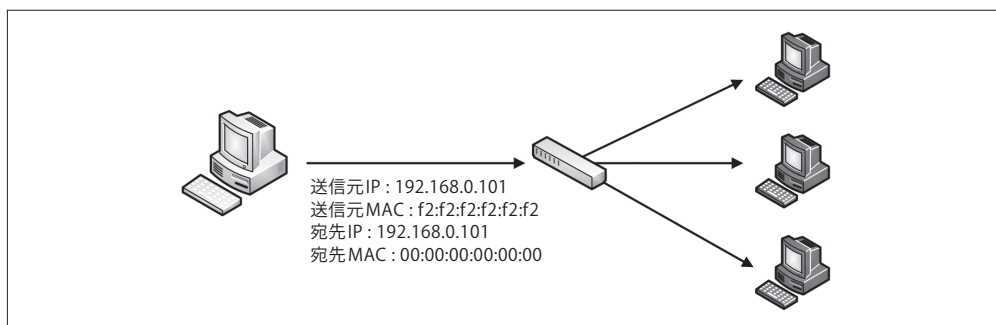


図6-5 gratuitous ARPの処理

gratuitous ARPパケットが使用される場面はいくつかあります。一番よく見られるのは、IPアドレスの変更です。キャプチャファイル arp_gratuitous.pcap を開き、実際に見てみてください。gratuitous ARPを構成するパケットは1つだけなので、ファイルに含まれているパケットも1つです (図6-6)。

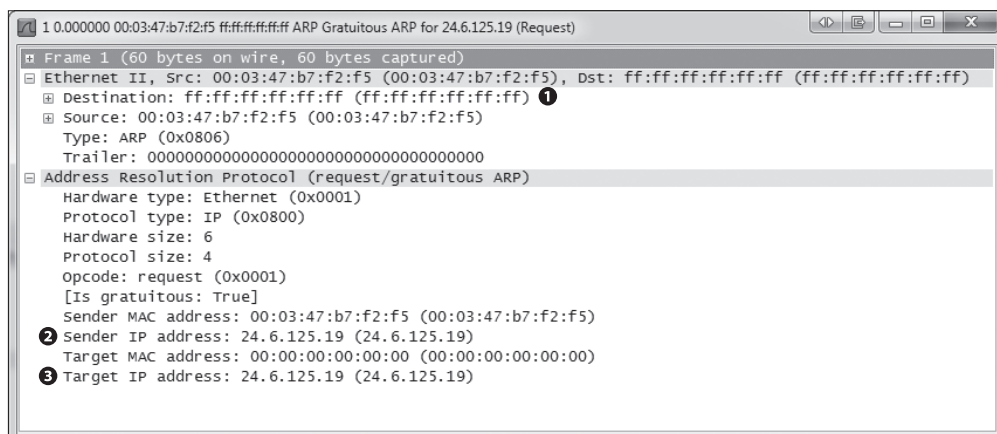


図6-6 gratuitous ARPパケット

イーサネットヘッダを調べると、このパケットはブロードキャストとして送信されているため、すべての機器が受信することがわかります❶。gratuitous ARPのヘッダはARPリクエストのヘッダと似ていますが、送信元IPアドレス❷と宛先IPアドレス❸が同一である点が異なります。gratuitous ARPパケットを受け取った機器は、ARPテーブルを新しいIPアドレスとMACアドレスとのマッピング情報で更新します。このARPパケットは一方的に送りつけられるものですが、受け取った側は、結果としてAPRキャッシュを更新するため、「gratuitous (勝手)」ということになるのです。

gratuitous ARPパケットが使われる場面はほかにもあります。機器のIPアドレスの変更によって生成される以外に、一部のOSは起動時にgratuitous ARPを生成します。また受信トラフィックのロードバランスのためにgratuitous ARPパケットが使われる場合があります。

6.2 IP (Internet Protocol)

OSI参照モデルにおける第3層プロトコルの主な役割は、ネットワーク間で通信を行うことです。ここまで見てきたように、MACアドレスは第2層の単一ネットワーク上での通信に利用されます。似たような形で、第3層はネットワーク間の通信処理を担当しています。この通信を可能にするプロトコルはいくつかありますが、もっとも一般的なのはIP (Internet Protocol) でしょう。ここではRFC 791で定義されたIPv4を見ていきます。

IPv4の働きを理解するには、ネットワーク間のトラフィックの流れを理解する必要があります。IPv4は通信処理を担っており、エンドポイントの場所にかかわらず、その間のデータ通信を担当しています。

ハブまたはスイッチですべての機器がつながっている単純なネットワークをLAN (Local Area Network) と呼びます。2つのLANを接続する場合はルータでつなぎます。複雑なネットワークは、世界中に分布する何千台ものルータを経由する何千ものLANで構成されていることがあります。インターネット自体も、数百万に及ぶLANとルータの集合体と言えます。

6.2.1 IPアドレス

IPアドレスは32ビットのアドレスで、ネットワークに接続された機器に割り振られた識別番号です。32文字もの長さの1と0が連続した文字列を覚えるのは至難の業なので、IPアドレスはドット区切り10進数表記 (dotted-quad notation) で記述されます。

ドット区切り10進数表記では、IPアドレスを構成する1と0の数字を4分割し、10進数へと転換して、A.B.C.Dの形式で0から255までの数字を使って表現します (図6-7)。11000000 10101000 00000000 00000001というIPアドレスがあるとします。この値はまず覚えられません。しかしドット区切り10進数表記を使えば、192.168.0.1と表記することができます。

IPアドレスを4分割するには理由があります。IPアドレスは2つの部分、ネットワークアドレスとホストアドレスから成り立っています。ネットワークアドレスは機器が接続しているLANを識別するもので、ホストアドレスは機器そのものを識別するためのものです。IPアドレスのどの部分がネット

ワークアドレスで、どの部分がホストアドレスとなるかは常に同じというわけではありません。これを決めるのは、ネットワークマスク（ネットマスク）と呼ばれるアドレス情報で、サブネットマスクと呼ばれることもあります。

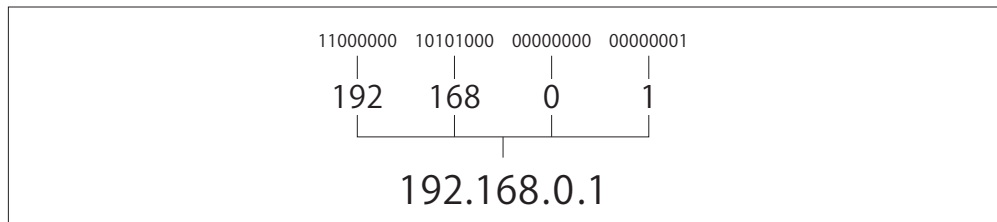


図6-7 ドット区切り10進数表記によるIPv4アドレス表記

ネットマスクは、IPアドレスのどの部分がネットワークアドレスで、どの部分がホストアドレスかを識別します。ネットマスクの数値も32ビット長であり、1にセットされた各ビットが、ネットワークアドレスのために確保されたIPアドレス部分を示します。残ったビットは0にセットされ、ホストアドレスを示します。

たとえば、2進数で00001010 00001010 00000001 00010110となるIPアドレス10.10.1.22があったとします。ネットマスクにより、IPアドレスを2つに分けることができます。ネットマスクが11111111 11111111 00000000 00000000だとします。この場合、アドレスの前半（10.10 / 00001010 00001010）はネットワークアドレスとなり、後半（.1.22 / 00000001 00010110）がネットワーク上にある個々のホストになります（図6-8）。

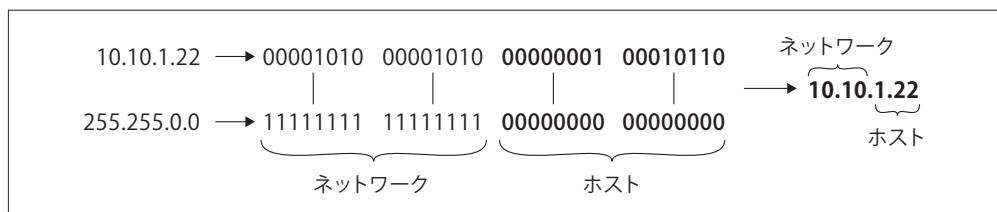


図6-8 IPアドレスのビットの配置はネットマスクで決まる

ネットマスクもドット区切り10進数表記で記述できます。たとえばネットマスク11111111 11111111 00000000 00000000は、255.255.0.0と表記できます。

一般にIPアドレスとネットマスクは、CIDR（Classless Inter-Domain Routing）表記で記述されます。この形式では、IPアドレスの表記に続き、/記号とIPアドレスのネットワーク部分を表すビット数が続きます。たとえばIPアドレス10.10.1.22、ネットマスク255.255.0.0をCIDR表記で記述すると、10.10.1.22/16となります。

6.2.2 IPv4ヘッダ

送信元と宛先のIPアドレスはIPv4パケットヘッダの重要な構成部分ですが、パケット内のIP情報はこれだけではありません。IPヘッダは、先に見たARPパケットのヘッダと比べるとかなり複雑です。IPが適切に機能するよう多くの機能が盛り込まれているのです。

IPv4ヘッダには以下のフィールドがあります (図6-9)。

バージョン

使用されているIPのバージョン。

ヘッダ長

IPヘッダの長さ。

サービスタイプ (TOS)

トラフィックに優先順位をつけるためにルータが使用する優先フラグとサービス種別フラグ。

パケット長

パケットに含まれるIPヘッダとデータの長さ。

識別子

パケットもしくは断片化した一連のパケット群を識別するために用いられる識別子。

フラグ

パケットが断片化されたパケットの一部なのかどうかを識別するのに使われます。

フラグメントオフセット

パケットが断片化されている場合、このフィールドの値を使ってパケットを正しい順序で並べて復元します。

生存時間 (TTL)

パケットの「寿命」を定義する値。ルータを通過する際のホップ数もしくは秒数で計測されます。

プロトコル

パケットのタイプを識別するのに使われます。

ヘッダチェックサム

IPヘッダが破損していないかを検証するのに用いられるエラー検出機構。

送信元IPアドレス

パケットを送信した機器のIPアドレス。

宛先IPアドレス

パケットの宛先のIPアドレス。

オプション

追加のIPオプションのために予約されている。ソースルーティングやタイムスタンプのオプション情報が含まれます。

データ

IPとともに送信される実際のデータ。

IP (インターネットプロトコル)					
ビットオフセット	0-3	4-7	8-15	16-18	19-31
0	バージョン	ヘッダ長	TOS	パケット長	
32	識別子			フラグ	フラグメントオフセット
64	TTL		プロトコル	ヘッダチェックサム	
96	送信元IPアドレス				
128	宛先IPアドレス				
160	オプション				
160または192以上	データ				

図6-9 IPv4パケット構造

6.2.3 生存時間 (TTL)

ip_ttl_source.pcap ip_ttl_dest.pcap

パケットが破棄されるまでの時間、あるいはパケットが通過できるルータの数を定義するのが、生存時間 (TTL) です。TTLはパケットが作成されるときに定義され、通常パケットがルータによって転送されるごとに、1ずつ減っていきます。たとえば、あるパケットのTTLが2の場合、最初に到達したルータはTTLを1減らして次のルータへとパケットを転送します。さらに受け取ったルータはTTLを0へと減らすため、パケットの最終的な宛先がそのネットワーク上にない場合、パケットは破棄されます (図6-10)。TTL値は定義上時間ベースであるため、輻輳しているルータの場合、TTL値が1より多く減ってしまふことがあります。しかし通常は、1台のルータを通過するごとにTTLが1だけ減ると仮定して問題ありません。

TTL値はなぜ重要なのでしょう？ パケットの生存時間について意識するのは、通常送信元から宛先までに要する時間だけです。しかしながら、何十台ものルータを経由して、インターネット越しの宛先までたどり着かなければならないパケットを考えてみましょう。その途中で設定に誤りがあるルータにぶつかれば、最終的な宛先へと到着できなくなる可能性があります。こうしたとき、ルータはいくつかのことを行いますが、その行為のためにパケットがネットワーク上を永遠にさまよう場合があるのです。

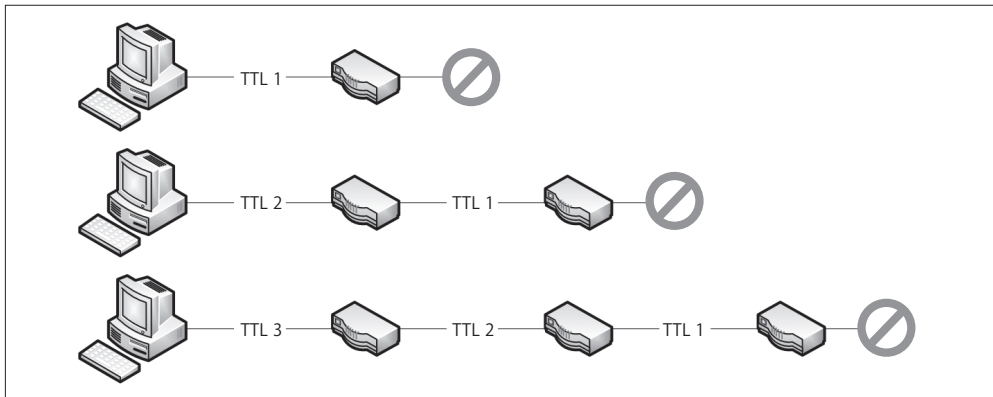


図6-10 パケットのTTLはルータを通過するごとに減少する

プログラミングの経験がまったくなくても、無限ループがさまざまな問題を引き起こし、プログラムやOS全体をクラッシュさせてしまう結果をもたらすのはおわかりでしょう。同じことがネットワーク上のパケットでも起こるのです。ルータ間を延々と行き来するパケットの数が増えれば、DoS状態に陥るまで、ネットワークの使用可能帯域は消耗します。こうした問題を回避するために、IPヘッダのTTLフィールドが作られたのです。

Wiresharkで一例を見てみましょう。ip_ttl_source.pcap ファイルには、2つのICMPパケットが含まれています。ICMP（本章でのちほど説明します）がIPを使ってパケットを送っていることは、[Packet Details] ペインのIPヘッダセクションを展開することで確認できます（図6-11）。

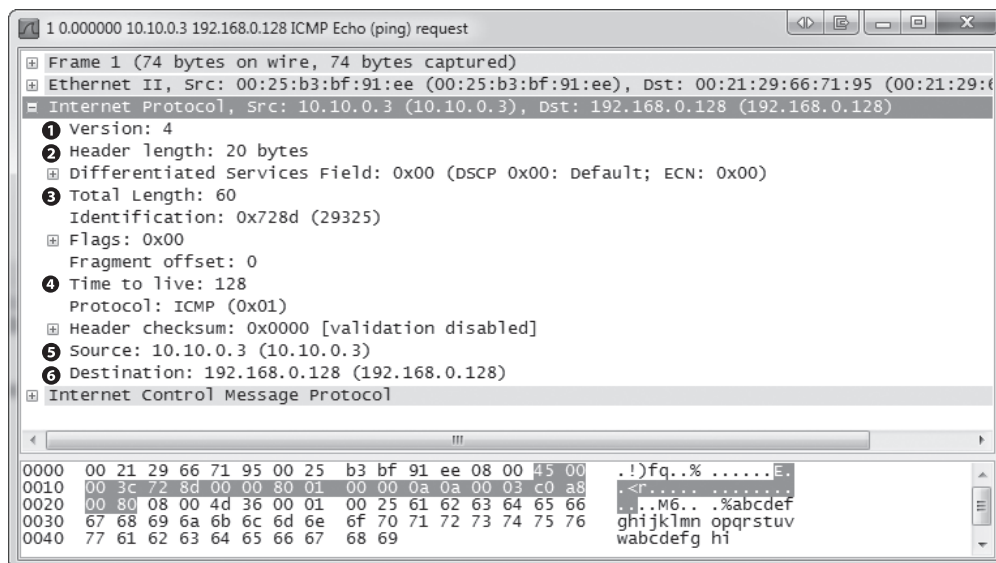


図6-11 パケットのIPヘッダ

ここでは、IPのバージョンは4❶、ヘッダ長は20バイト❷、ヘッダとペイロードを合わせたパケット長は60バイト❸、TTLフィールドの値は127❹であることがわかります。

ICMP pingの主な目的は、機器間の通信テストです。データはある機器から別の機器へリクエストとして送られ、受信した機器はそのデータをレスポンスとして送り返さなければなりません。このファイルでは、アドレス10.10.0.3の機器❺が、アドレス192.168.0.128の機器❻へとICMPリクエストを送っています。このキャプチャファイルは、送信元である10.10.0.3の機器で作成されています。

今度はip_ttl_dest.pcap ファイルを開きましょう。このファイルでは、データは宛先である192.168.0.128でキャプチャされています。先頭のパケットのIPヘッダを開き、TTL値を調べます(図6-12)。

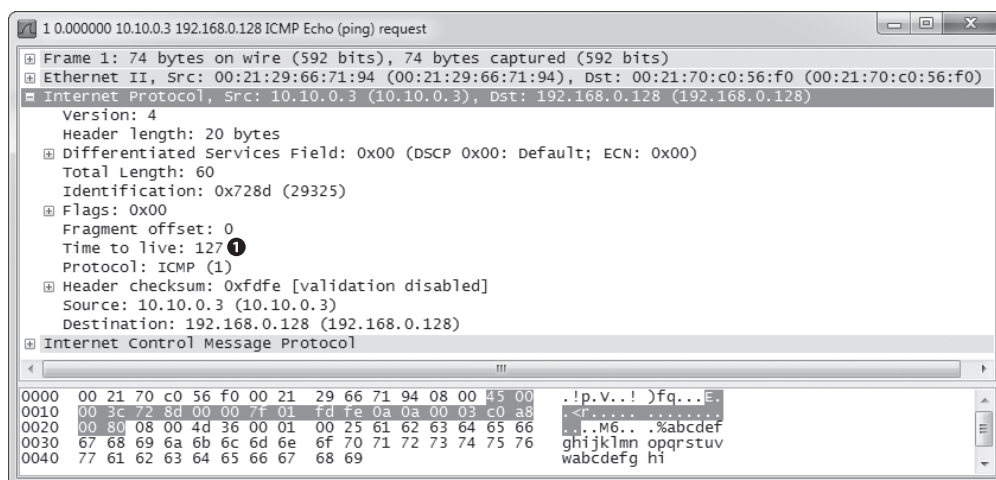


図6-12 IPヘッダからTTLが1減少しているのがわかる

このTTL値が127(1)で、最初のTTL値128よりも1少ないのがすぐにわかるはずです。ネットワークの構造を知らなくても、この2台の通信機器の間には1台のルータがあり、そのルータを通過するときにTTL値が1減ったと結論づけることができます。

6.2.4 IPフラグメンテーション (断片化)

ip_frag_source.pcap

パケットのフラグメンテーション (断片化) は、データストリームをより小さな断片 (fragment) に分割することで、多種多様なネットワーク上でも確実に送信するためのIPの機能です。

パケットの断片化は、第2層のデータリンク層プロトコルのMTU (Maximum Transmission Unit) サイズと、これらの第2層プロトコルを使っている通信機器の設定によって発生します。多くの場合、第2層で使われているプロトコルはイーサネットです。イーサネットのデフォルトのMTUは1500、つまりイーサネットネットワーク上で送信できるパケットの最大サイズは1500バイトということになります (イーサネットヘッダ分の14バイトは除く)。



MTUには標準値がありますが、機器のMTUは手動で再設定できます。MTU設定はインターフェイスごとに設定でき、ルータのインターフェイスだけでなく、WindowsやLinuxシステムでも変更できます。

通信機器はIPパケットの送信準備をする際に、パケットが送信されるネットワークインターフェイスのMTUサイズとパケットのデータサイズとを比較して、パケットを断片化すべきかどうかを判断します。データサイズがMTUより大きければ、パケットは断片化されます。パケットの断片化は次のような手順で行われます。

1. 通信機器がデータを送信するためにデータをいくつかのパケットへと断片化します。
2. 断片化された各データのサイズに合わせ、IPヘッダのパケット長フィールドを設定します。
3. 最後のパケットを除いたすべてのパケットのMF (More Fragments) フラグを1にセットします。
4. 断片化された各パケットヘッダのフラグメントオフセットフィールドを設定します。
5. パケットが送信されます。

ip_frag_source.pcapファイルは、アドレス192.168.0.128の通信機器へpingリクエストを送信しているアドレス10.10.0.3のホストで取得されたものです。[Packet List] ペインの [Info] カラムには、ICMP (ping) リクエストに続き、断片化されたIPパケットが2つあります。

まずは1つ目のパケットのIPヘッダを調べてみましょう (図6-13)。

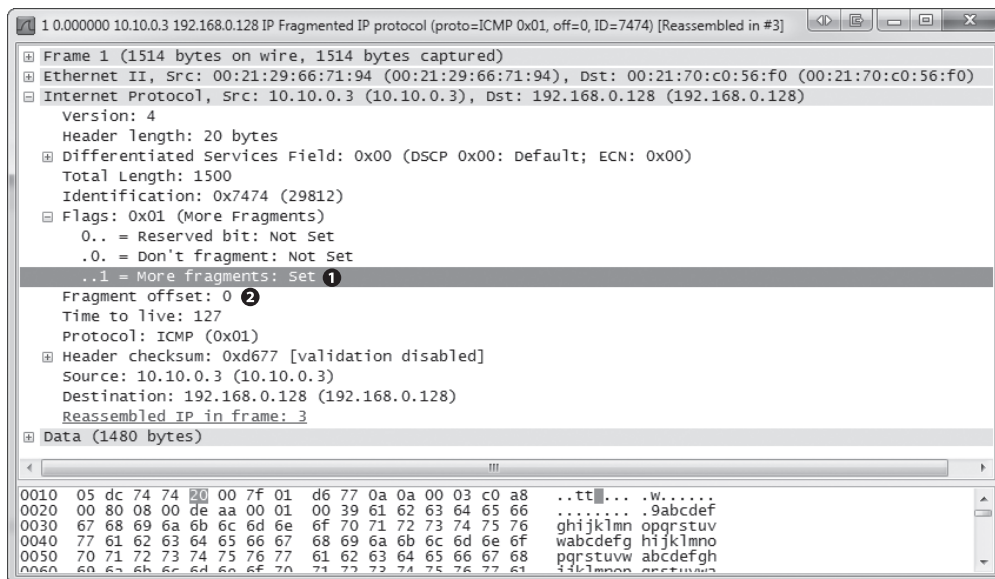


図6-13 MFとフラグメントオフセットの値が断片化されたパケットであることを示している

MFフラグとフラグメントオフセットフィールドから、このパケットは断片化されたパケットの一部であることがわかります。断片化されたパケットは、フラグメントオフセットの値が正になっているか、MFフラグがセットされています。最初のパケットではMFフラグがセットされている❶ので、受信側の機器では引き続きパケットの到着を待ち続けることになります。フラグメントオフセットが0に設定されている❷ので、このパケットが断片化されたパケットの最初のパケットであることがわかります。

2番目のパケットのIPヘッダ(図6-14)でもMFフラグがセットされていますが❶、フラグメントオフセットの値は1480です❷。これは1500バイトのMTUから、IPヘッダの20バイトを引いた数字です。

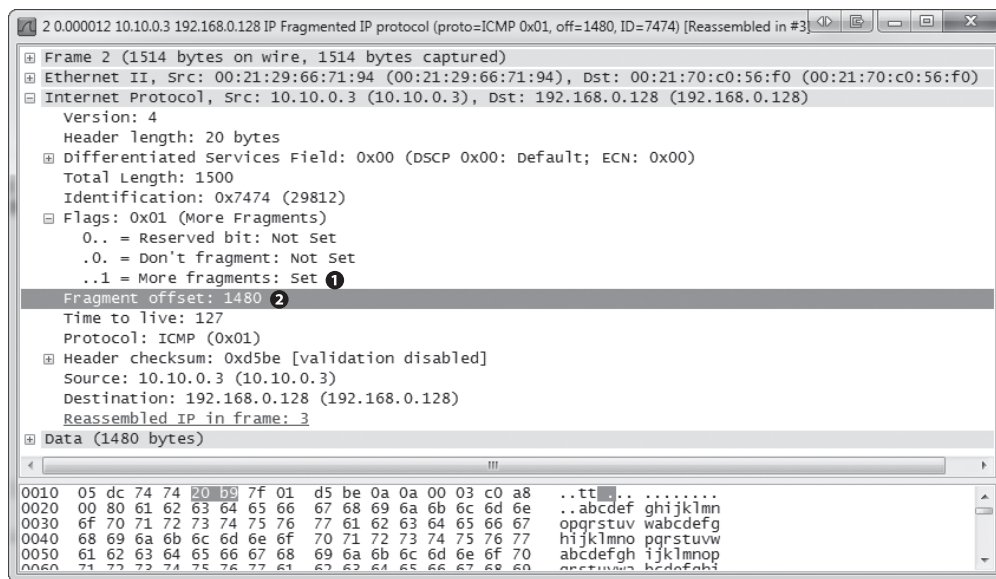


図6-14 フラグメントオフセットの値はパケットのサイズによって増える

3番目のパケット(図6-15)ではMFフラグはセットされていない❶ので、これはデータストリームの最後のパケットであることがわかります。フラグメントオフセットの値は1480 + (1500 - 20) の2960に設定されています❷。これらの断片化されたパケットは、IPヘッダの識別子フィールドの値が同一である❸ことから、いずれも同じパケットの一部であるとわかります。

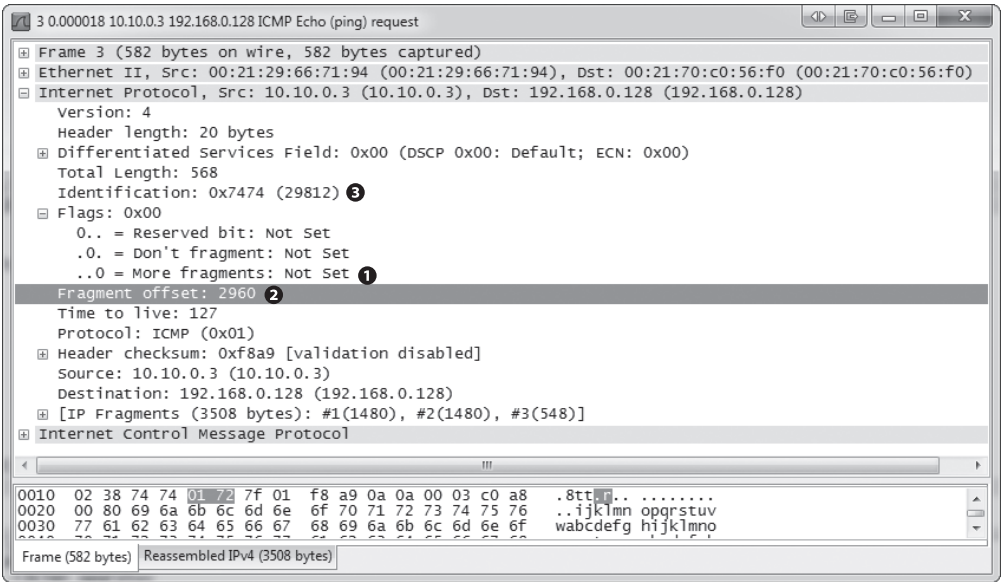


図6-15 MFがセットされていないので最後の packet であることがわかる

6.3 TCP

TCP (Transmission Control Protocol) の目的は、エンドポイント間でデータを確実に配送することです。RFC 793で定義されているTCPは、OSI参照モデルの第4層で動作します。TCPはデータの順序性とエラー訂正を行い、データが到着すべき場所に確実に届くようにします。一般的に使われているアプリケーション層プロトコルの多くが、TCPとIPによってパケットを宛先に配送しています。

6.3.1 TCPヘッダ

TCPヘッダの複雑さを見れば、どれほど多くの機能を提供しているかがわかります。図6-16に示しているように、TCPヘッダのフィールドは次のようになっています。

送信元ポート番号

パケットの送信に用いられるポート番号。

宛先ポート番号

パケットが送信される先のポート番号。

シーケンス番号

TCPセグメントを識別するための番号。このフィールドはデータストリームの欠損を防ぐために使われます。

確認応答番号 (ACK 番号)

通信の相手機器から受け取る番号で、次のパケットに含まれることが期待されるシーケンス番号。

フラグ

送信されたTCPパケットのタイプを識別するもので、URG、ACK、PSH、RST、SYN、FINフラグがあります。

ウィンドウサイズ

バイト単位の受信側のバッファサイズ。

チェックサム

TCPヘッダとデータの内容が、到着時に変更されていないことを確認するために使われます。

緊急ポインタ

URGフラグをセットした場合、このフィールドにより、CPUがパケット内のデータをどこから読みはじめるべきかを指示します。

オプション

TCPパケットで指定できるさまざまなオプションフィールド。

TCP				
ビットオフセット	0-3	4-7	8-15	16-31
0	送信元ポート番号			宛先ポート番号
32	シーケンス番号			
64	ACK番号			
96	データオフセット	予約済	フラグ	ウィンドウサイズ
128	チェックサム			緊急ポインタ
160	オプション			

図6-16 TCPヘッダ

6.3.2 TCPポート

tcp_ports.pcap

すべてのTCP通信は送信元と宛先のポートを使って行われます。これはTCPヘッダからもわかります。ポートとは、昔の電話交換機のジャックのようなものです。電話交換機のオペレータは、ライトとプラグを監視しています。ライトが光ると、電話のかけ手とつなぎ、誰に電話をかけたいのかを尋ね、かけ手と話したい相手（受け手）とをケーブルでつなぎます。すべての通話には送信元ポート（電話のかけ手）と宛先ポート（受け手）が必要です。TCPポートもほぼ同様に機能します。

リモートサーバや機器上にある特定のアプリケーションにデータを送信するには、TCPパケットが

待ち受けているリモートサービスのポートを知っていなければなりません。設定済みのポート以外のポートにアクセスしようすると、通信に失敗してしまいます。

送信元ポートは、このやり取りの中でそれほど重要ではなく、ランダムに選ぶことができます。リモートサーバは、送信されてきた元々のパケットから、通信に使うポートを決定します (図6-17)。

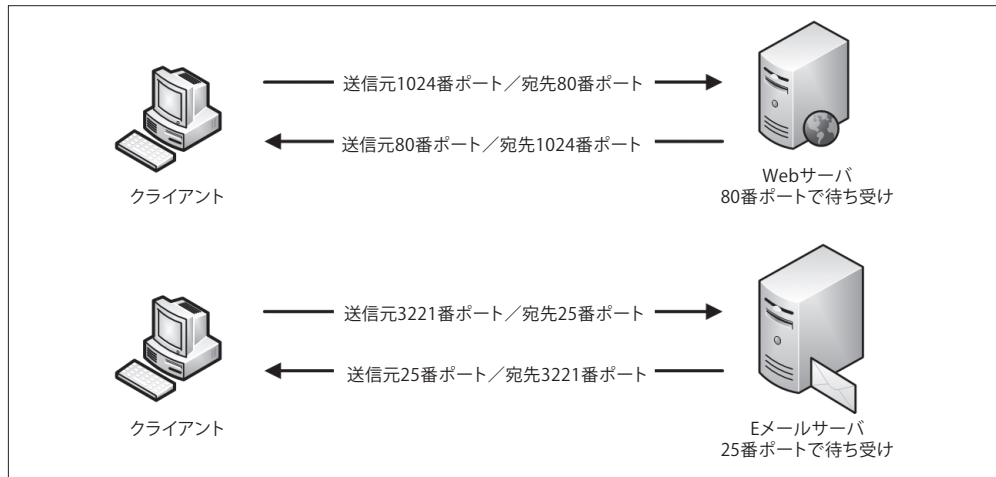


図6-17 TCPはポートを使ってデータを送信する

TCPで通信に使えるポートは65,535個あり、一般に2つのグループに分けることができます。

- 1から1023までがスタンダードポートグループ[†]です (0は予約済みなので除外します)。サービスが使っている標準のポート番号の多くが、標準ポートグループ内のポートとなっています。
- 1024から65535までがエフェメラルポートグループです (ただしOSによってはこの定義が異なります)。どんなときでも、1個のポートで通信できるのは1つのサービスだけなので、現代のOSは、ポートの重複を避けるために、送信元ポートをランダムに選択します。こうした送信元ポートは、エフェメラルポートの範囲に入ります。

tcp_ports.pcap ファイルを開いて、いくつかのTCPパケットを確認することで、パケットが使っているポート番号を識別してみましょう。このファイルには、2つのWebサイトをブラウズしているクライアントのHTTP通信が含まれています。前述したように、HTTPは通信にTCPを使用するので、これは標準的なTCPトラフィックの良いサンプルとなります。

このファイルの最初のパケットでは (図6-18)、最初の2つの値がパケットの送信元ポートと宛先ポートを表しています。このパケットは、172.16.16.128から212.58.226.142へ送信されています。送信元ポートは2826❶で、エフェメラルポートです (送信元ポートはOSがランダムに選択するということを思い出してください。なおランダムに選択された番号より大きい値が使われることもあります)。宛先

[†] 監訳注：日本ではウェルノウンポート (well-known port) と呼ばれることが多いと思います。

ポートは標準のポートである80番ポートです❷。このポートはHTTPを使用するWebサービスが利用するものです。

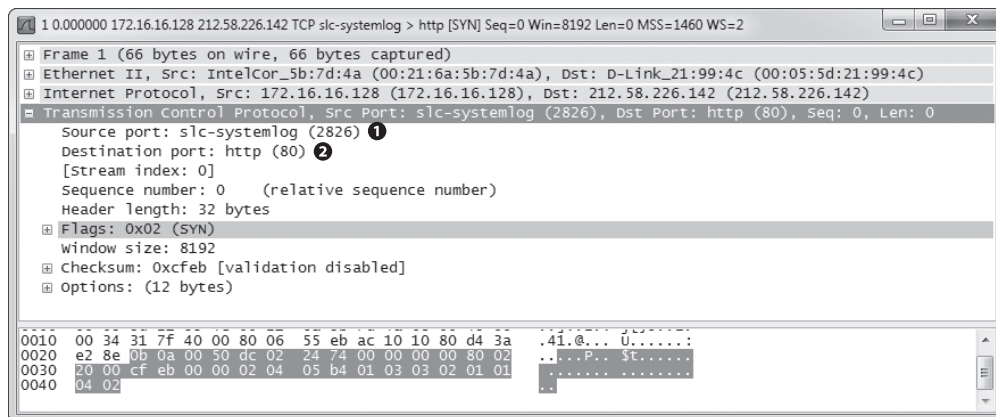


図6-18 TCPヘッダに送信元ポートと宛先ポートが含まれている

Wiresharkではこれらのポートにslc-systemlog(2826)とhttp(80)というラベルを付けている点に注意してください。Wiresharkはポート番号と、その一般的な用途の一覧を管理しています。これらのポートは主にスタンダードポートですが、多数のエフェメラルポートがサービスに割り当てられています。ポートのラベルは非常にわかりにくいので、名前解決を無効にしておきましょう。これを行うには、メニューから[Edit] → [Preferences] → [Name Resolution]を選択し、[Enable Transport Name Resolution]の横のチェックを外します。この機能は有効のままにして、Wiresharkが特定ポートを識別する方法だけを変更したい場合は、Wiresharkプログラムディレクトリにある「Services」ファイルを修正します。このファイルはIANA (Internet Assigned Numbers Authority) の一般的なポート一覧を元にしています。

2番目のパケットは、212.58.226.142から172.16.16.128へと送り返されたものです(図6-19)。IPアドレス同様、送信元と宛先ポートが入れ替わっています❶。

すべてのTCP通信が、ランダムに選ばれた送信元ポートが既知の宛先ポートと通信するという形で行われます。最初のパケットが送られると、リモートの通信機器が確立されたポートを使って送信元と通信します。

このサンプルのキャプチャファイルには、もうひとつ通信ストリームが含まれています。通信に使用されたポート番号を探してみましょう。

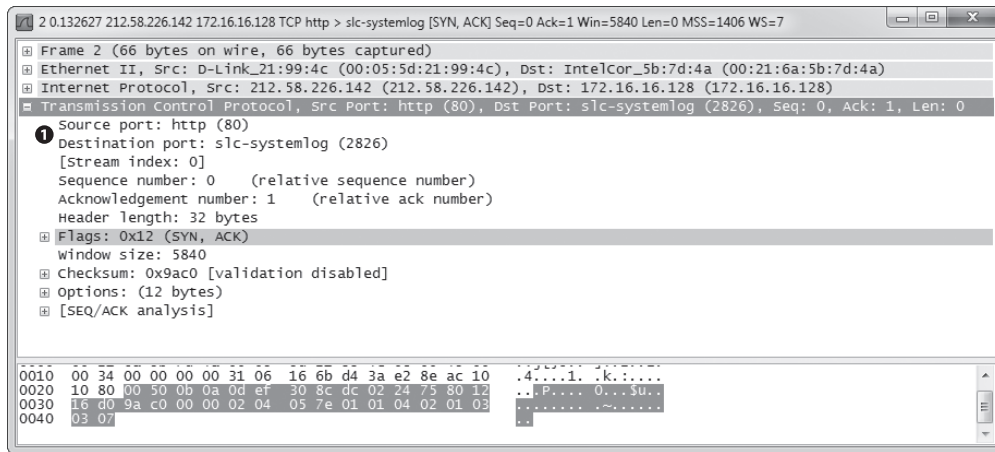


図6-19 返信では送信元と宛先のポート番号が入れ替わる



本書を読み進めるに従い、一般的なプロトコルやサービスに割り当てられたポートについての知識がさらに深まるでしょう。そのうちに使っているポートを見れば、使われているサービスと通信機器がわかるようになります。一般的なポート番号の網羅的な一覧については、<http://www.iana.org/assignments/port-numbers/>を参照してください。

6.3.3 TCPの3ウェイハンドシェイク

tcp_handshake.pcap

TCPを使用する通信は、必ず2つの機器間のハンドシェイクが始まります。ハンドシェイクの処理には、いくつかの目的があります。

- 送信元の機器が、宛先の機器と確実に通信できるようにします。
- 送信元の機器が、送信元が通信しようとしているポートで待ち受けが行われていることを確認できるようにします。
- 送信元の機器が受信者に向けて初期シーケンス番号の送信を行い、両機器間でパケットの順序性を保てるようにします。

TCPハンドシェイクには3つのステップがあります(図6-20)。最初のステップで、通信を行いたい機器(ホストA)は宛先(ホストB)にTCPパケットを送ります。この最初のパケットには、下位層のプロトコルヘッダ以外のデータは含まれていません。このパケットのTCPヘッダにはSYNフラグがセットされており、通信処理に使われる最初のシーケンス番号とMSS(Maximum Segment Size)が含まれています。ホストBはこのパケットに応答し、SYNフラグとACKフラグがセットされ、最初のシーケンス番号を含んだ同様のパケットを送り返します。最後にホストAは、ACKフラグのみがセットされた最終パケットをホストBに送ります。この処理が完了すると、通信開始に必要なすべての情報を、両ホストが持つことになります。

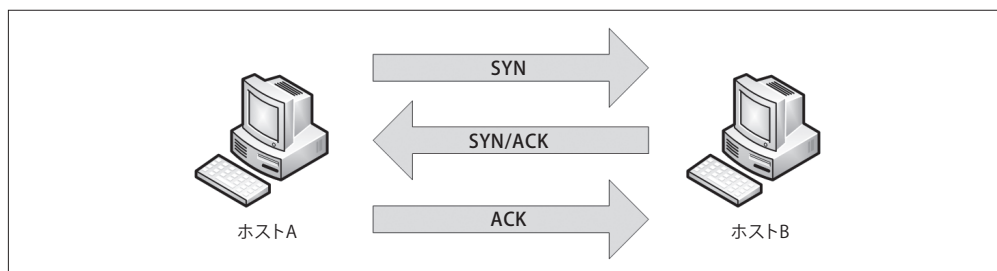


図6-20 3ウェイハンドシェイク



TCPパケットは、セットされたフラグの名前で呼ばれることがよくあります。たとえばSYNフラグがセットされたパケットなら、SYNパケットと呼びます。つまりTCPハンドシェイクプロセスで使われるパケットは、SYNパケット、SYN/ACKパケット、ACKパケットとなります。

tcp_handshake.pcapを開き、この処理を実際に見てみましょう。Wiresharkには、TCPパケットのシーケンス番号を解析しやすい番号に置き換える機能がありますが、実際のシーケンス番号を見るため、ここではこの機能を無効にしておきます。無効にするには、メニューから[Edit] → [Preferences]を選択し、[Protocols]を広げてから[TCP]を選択します。次に[Relative Sequence Numbers]と[Window Scaling][†]の横のボックスのチェックを外し、[OK]ボタンをクリックします。

最初のパケットはSYNパケットです(図6-21)。このパケットは172.16.16.128の2826番ポートから、212.58.226.142の80番ポートへと送られています。送信元されたシーケンス番号は3691127924であること❶がわかります。

2番目のパケットは、212.58.226.142からの応答となるSYN/ACKパケットです(図6-22)。このパケットには、初期シーケンス番号(233779340)❶と、ACK番号(3691127925)❷が含まれています。ACK番号は、送信元から送られてきたシーケンス番号に1を加えたものになっています。次に受け取れることを期待するシーケンス番号の指定に使われるのがこのフィールドだからです。

最後のパケットは、172.16.16.128から送られるACKパケットです(図6-23)。このパケットには、先ほどのパケットのACK番号フィールドにあったシーケンス番号3691127925❶が含まれています。

TCP通信の前には必ずハンドシェイクが行われます。中身のごちゃごちゃしたキャプチャファイルで通信の始まりを検索する場合、SYN-SYN/ACK-ACKのシーケンスが目印となります。

[†] 監訳注：監訳者が1.8.0、1.6.8、1.0.15などのバージョンで確認した限り[Window Scaling]というチェックボックスは存在しませんでした。

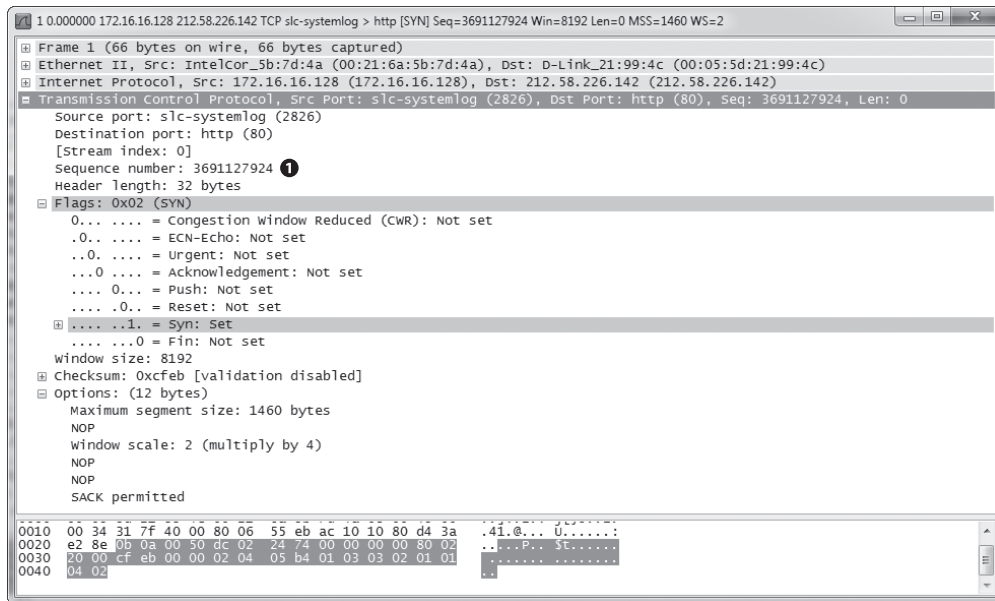


図6-21 最初のSYN/パケット

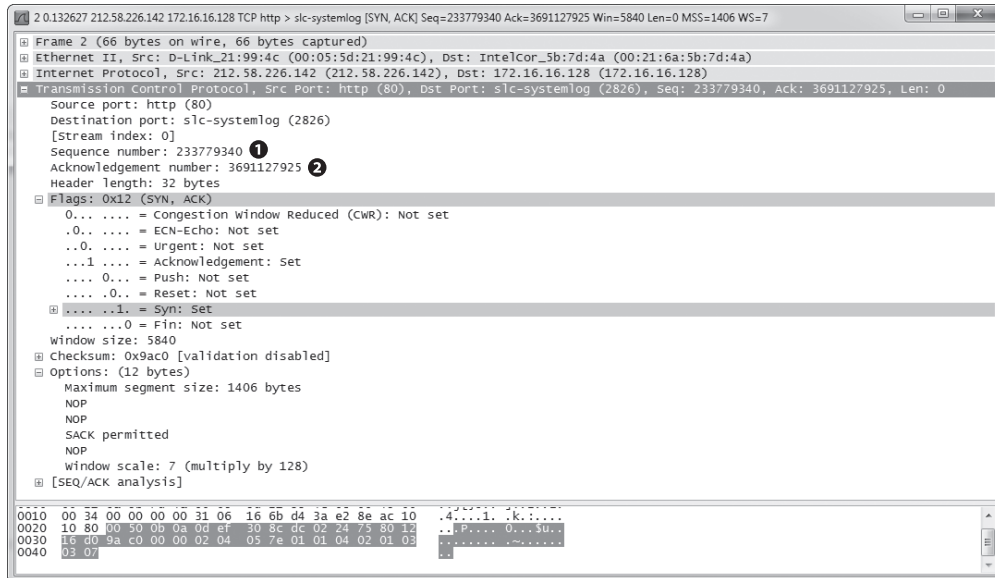


図6-22 SYN/ACK応答パケット

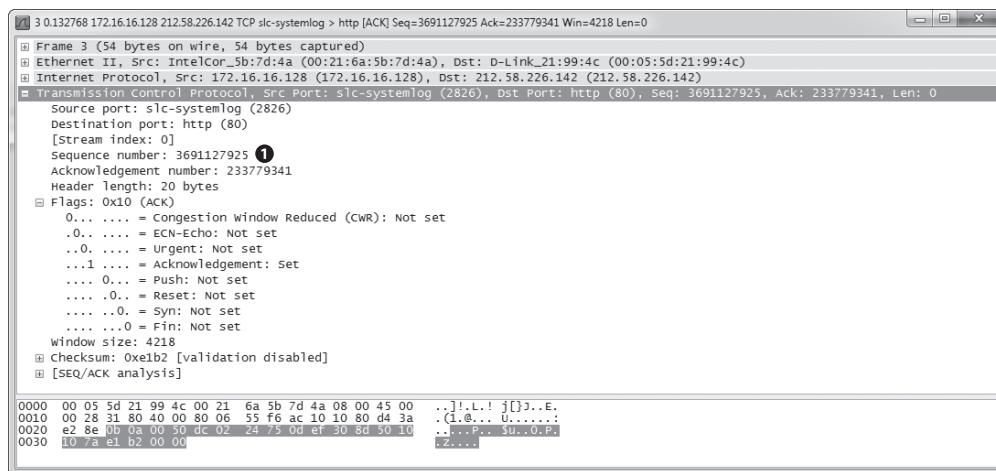


図6-23 最終のACKパケット

6.3.4 TCPのティアダウン (切断)

tcp_teardown.pcap

どんな出会いにも別れはつきものですが、TCPハンドシェイクにもティアダウンがあります。TCPのティアダウンは、通信を終えた2つの通信機器間の接続を正常に終了させるのに用いられるものです。この処理には4個のパケットがかかわり、接続の終了を示すものとしてFINフラグが使われます。

ティアダウンでは、ホストAはホストBにFINフラグとACKフラグがセットされたTCPパケットを送り、接続の終了を知らせます。ホストBはACKパケットで応答後、自分のFIN/ACKパケットを送ります。これに対しホストAはACKパケットで応答し、通信が終了します。この処理を図式化したのが図6-24です。

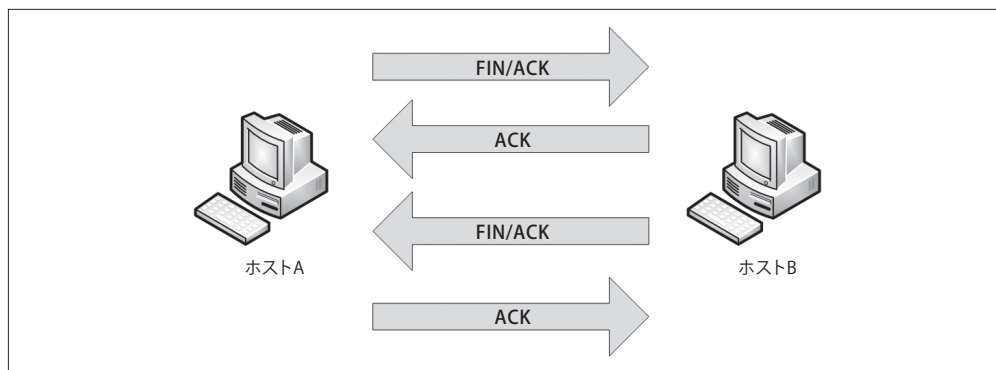


図6-24 TCPティアダウンの処理

tcp_teardown.pcap ファイルを開いて、この処理をWiresharkで見てみましょう。最初のパケット(図6-25)に着目すると、67.228.110.120の機器が、FIN/ACKフラグがセットされた①パケットを送って、ティアダウンの処理を開始しているのがわかります。

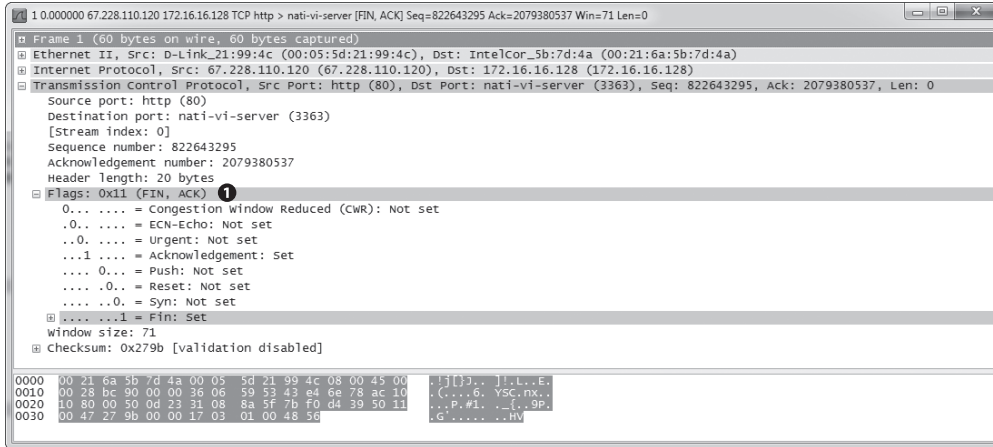


図6-25 FIN/ACKがティアダウンの処理を開始

このパケットを受け取ると、172.16.16.128は受信したことを知らせるACKパケットで応答し、FIN/ACKパケットを送ります。67.228.110.120が最後のACKを送信すると、処理が完了します。ここで2つの機器間の通信は終了するので、通信を再開するには新たにTCPハンドシェイクを行わなければなりません。

6.3.5 TCPリセット

tcp_refuseconnection.pcap

すべてのコネクションがTCPティアダウンで正常に終了するのが理想です。しかし実際には、コネクションの突然の終了が頻繁に起こります。これは、攻撃者によるポートスキャンの実行や、単なる機器の設定ミスからも起こりえます。こうした場合、RSTフラグをセットしたTCPパケットが使われます。RSTフラグはコネクションの突然の終了や、コネクションの試行の拒否を示すためのものです。

tcp_refuseconnection.pcap ファイルは、RSTパケットを含んだネットワークトラフィックの一例です。最初のパケットは、192.168.100.1と80番ポートで通信しようとしている、192.168.100.138の機器から送られたものです。この機器は、192.168.100.1がCisco ルータであり、80番ポートで待ち受けていないことを知りません。Cisco ルータの場合、Web インターフェイスを設定しない限り、80番ポートでコネクションを待ち受けるサービスは存在しません。この通信の試行に対し、192.168.100.1は192.168.100.138にパケットを送り、80番ポートでは通信が行えないと知らせます。図6-26は、通信の試行が突如終了したことを示す、2番目のパケットのTCPヘッダです。このRSTパケットにはRSTフラグとACKフラグ以外には何も含まれておらず①、これ以降は通信はありません。

RSTパケットは、このサンプルのように通信の試行の最初で送られた場合、通信の途中で送られた

場合、どちらの場合でも通信を終了させます。

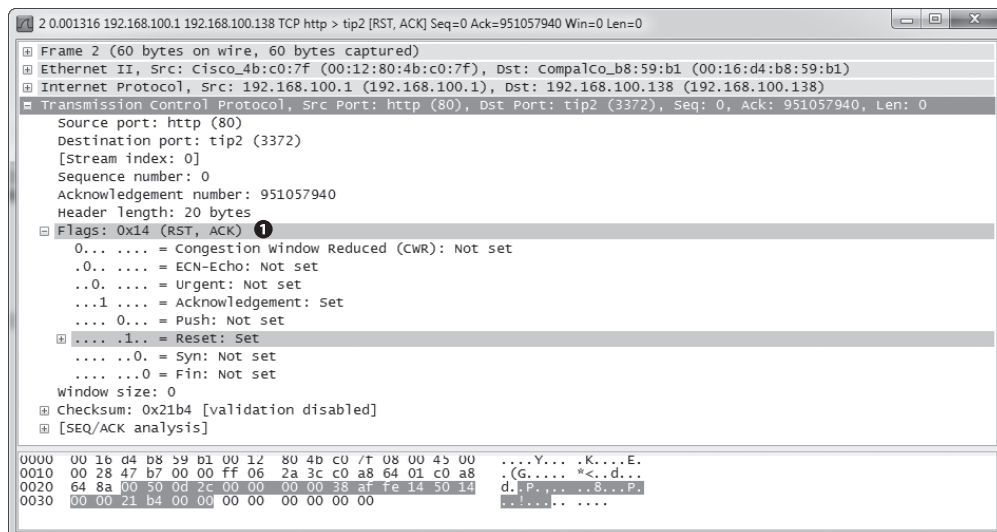


図6-26 RSTフラグとACKフラグが通信終了を知らせる

6.4 UDP

UDP (User Datagram Protocol) は、TCPのほかにネットワークでよく使われるもうひとつの第4層プロトコルです。TCPがエラー検出機能により信頼性の高いデータ転送を目的に設計されているのに対し、UDPは高速な転送を目的としています。このためUDPは、ベストエフォートでのデータ転送を行うものであり、一般的に「コネクションレスプロトコル」と呼ばれます。コネクションレスプロトコルは、TCPのハンドシェイクやティアダウンのような、手順に基づいたコネクションの確立や終了を行いません。

UDPは信頼性の高いデータ送信を提供しないコネクションレスプロトコルであるため、UDPのトラフィックは、どう考えても信頼できないように思えます。確かにそうなのですが、実際のところ、UDPを使うプロトコルは、信頼性を保証するための独自のサービスを内蔵していたり、信頼性の高い接続を実現するためにICMP機能を使ったりしています。たとえばパケットの高速送信を重視するDNSやDHCPといったアプリケーション層プロトコルは、UDPをトランスポート層プロトコルとして利用しますが、エラー検出と再送信タイマーを独自に実装しています。

6.4.1 UDPヘッダ

udp_dnsrequest.pcap

UDPヘッダは、TCPヘッダよりもはるかに小さくシンプルです。UDPヘッダのフィールドは次のようになっています (図6-27)。

送信元ポート

パケット送信に使われるポート。

宛先ポート

パケットの宛先となるポート。

パケット長

パケットの長さをバイトで示したもの。

チェックサム

UDPヘッダとデータの内容が、到着時に変更されていないことを確認するために使います。

UDP		
ビットオフセット	0-15	16-31
0	送信元ポート	宛先ポート
32	パケット長	チェックサム

図6-27 UDPヘッダ

udp_dnsrequest.pcap ファイル内のパケットは1つです。このパケットはUDPを使ったDNSリクエストです。UDPヘッダを開くと、4つのフィールドがあるのがわかります (図6-28)。

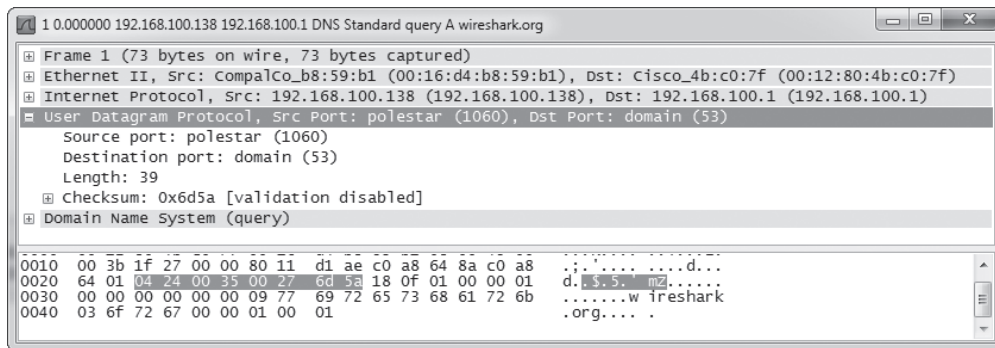


図6-28 UDPパケットの中身は非常にシンプル

UDPはデータ配信の信頼性を保証しないことを覚えておってください。したがってUDPを使うアプリケーションは、信頼性が必要な場合、別の手段を取る必要があります。

6.5 ICMP

ICMP (Internet Control Message Protocol) は、TCP/IPが動作するための補助的な役割を果た

すプロトコルで、TCP/IPネットワーク上の通信機器、サービス、経路などが使用可能かどうかに関する情報を提供します。ネットワークのトラブルシューティングに関するテクニックやツールの大半が、ICMPメッセージタイプを駆使しています。ICMPはRFC 792で定義されています。

6.5.1 ICMPヘッダ

ICMPはIPの一部であり、メッセージの送信にIPを使っています。ICMPヘッダは比較的小さく、目的に応じて変わります。ICMPヘッダのフィールドは次のとおりです (図6-29)。

タイプ

RFCの規定に基づく、ICMPメッセージのタイプや分類。

コード

RFCの規定に基づく、ICMPメッセージの下位分類。

チェックサム

受信時にICMPヘッダとデータの内容を確認するためのもの。

データ

タイプとコードのフィールドによって変わる部分。

ICMP			
ビットオフセット	0-15		16-31
0	タイプ	コード	チェックサム
32	データ		

図6-29 ICMPヘッダ

6.5.2 ICMPタイプとメッセージ

先述したように、ICMPパケット構造はその目的によって変わり、「タイプ」と「コード」の各フィールドの値によって決まります。

タイプをパケットの分類、コードを下位分類と考えるとよいかもしれません。たとえばタイプの値が3なら「宛先到達不可能 (Destination Unreachable)」を意味します。この情報だけでは問題解決には不十分ですが、コードの値が3、つまり「ポート到達不能 (Port Unreachable)」とわかれば、通信を試行しているポートに問題があると結論づけることができます。



ICMPのタイプとコードの網羅的なリストについては、<http://www.iana.org/assignments/icmp-parameters>を参照してください。

6.5.3 エコー要求とエコー応答

icmp_echo.pcap

ping コマンドに対する賛辞こそが、ICMP が名声を勝ち得た最大の要因と言ってもよいでしょう。ping は機器間の接続性の確認に用いられます。たいていの IT の専門家なら、ping に馴染んでいるはずです。

ping を使うには、コマンドプロンプトで ping <IP アドレス> と入力します。<IP アドレス> の部分は、ネットワーク上の機器の実際の IP アドレスに置き換えてください。対象の機器が起動しており、通信経路に問題がなく、ファイアウォールで通信がブロックされていなければ、ping コマンドに対する応答があるはずです。

図 6-30 では、応答を 4 回受け取ることに成功しており、そのサイズ、RTT、使用された TTL が表示されています。ping は、パケットの送信数、受信数、消失数についての概要も提供しています。通信に失敗した場合には、その理由を説明したメッセージが表示されます。

基本的に、ping コマンドは相手の機器にパケット 1 つ送って応答を待つことで、図 6-31 のようにその機器との接続性を判断します。

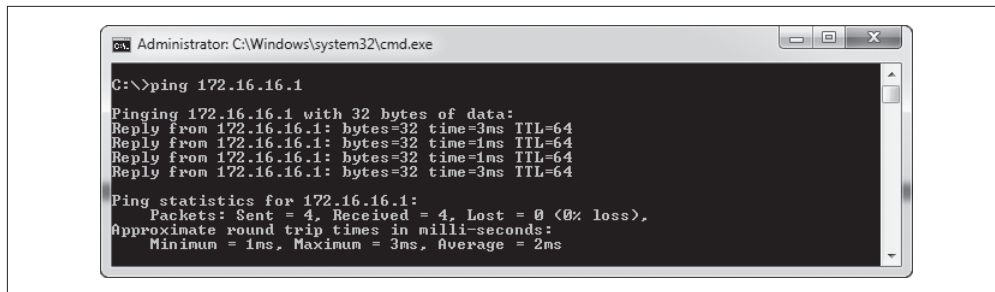


図 6-30 接続確認に用いられる ping コマンド

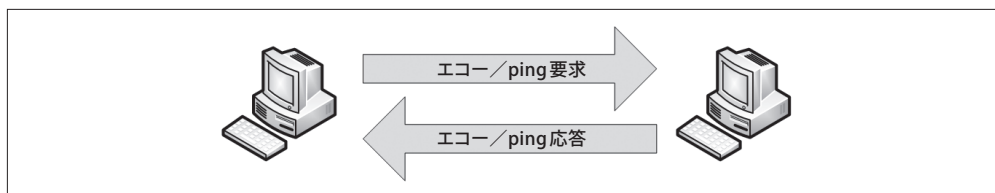


図 6-31 ping コマンドは 2 ステップのみ



ping は長きにわたり IT の必需品となっていますが、ホストベースのファイアウォールの普及により、結果の信憑性が少々怪しくなっています。今日のファイアウォールの多くが、ICMP パケットへの応答を抑止しているからです。攻撃者が ping を使って、機器への接続性を確認する行為が阻止されるため、セキュリティの観点からは素晴らしいのですが、同時にトラブルシューティングをも難しくしてしまっています。相手と通信ができるとわかっているのに、接続確認で ping を送っても応答が得られないというのは、イライラさせられるものです。

簡単なICMP通信の一例として、実際にpingコマンドがどう機能するかを見てみましょう。icmp_echo.pcapファイルのパケットは、pingを実行すると何が起こるかを示しています。

最初のパケット(図6-32)は、ホスト192.168.100.138が、192.168.100.1にパケットを送信していることを示しています❶。このパケットのICMP部分を展開し、タイプとコードのフィールドを見れば、このパケットの種類がわかります。このサンプルではパケットのタイプは8❷、コードは0❸なので、エコー要求であることを示しています(Wiresharkではタイプとコードについての説明が表示されます)。エコー要求(ping)は前半部分です。これはIPを使う単純なICMPパケットで、データはほとんど含んでいません。ICMPパケットには、タイプ、コード、チェックサムに加え、要求と応答を対にするためのシーケンス番号と、ICMPパケットの可変部分にランダムなテキスト文字列が含まれています。

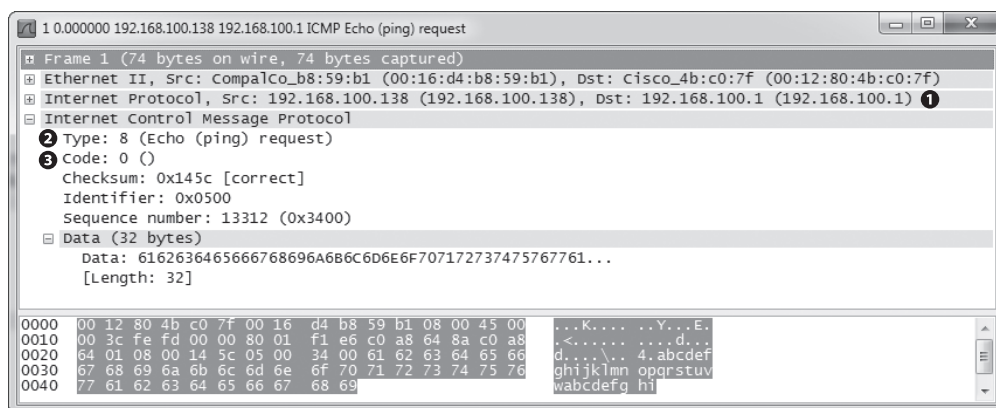


図6-32 ICMPエコー要求パケット



エコーとpingという用語はしばしばほぼ同じ意味で使われますが、pingはツールの名称であることを覚えておきましょう。pingはICMPエコー要求パケットを送るために使われます。

2番目のパケットは、要求に対する応答です(図6-33)。このパケットのICMP部分のタイプは0❶、コードは0❷で、これがエコー応答であることを示しています。シーケンス番号が最初のパケットと一致しているので❸、このエコー応答が先のパケットのエコー要求に対応するものであることがわかります。この応答パケットには、エコー要求で送られたものと同じ32バイトの文字列も含まれています❹。192.168.100.138がこのパケットを受け取ると、pingが成功を報告します(先ほどの図6-30)。

pingのデータ部分のサイズを増やすことで、パケットを断片化させて、ネットワークのさまざまなトラブルシューティングに利用できることを覚えておきましょう。断片化のサイズが小さいネットワークのトラブルシューティングの際には、これが必要となるかもしれません。

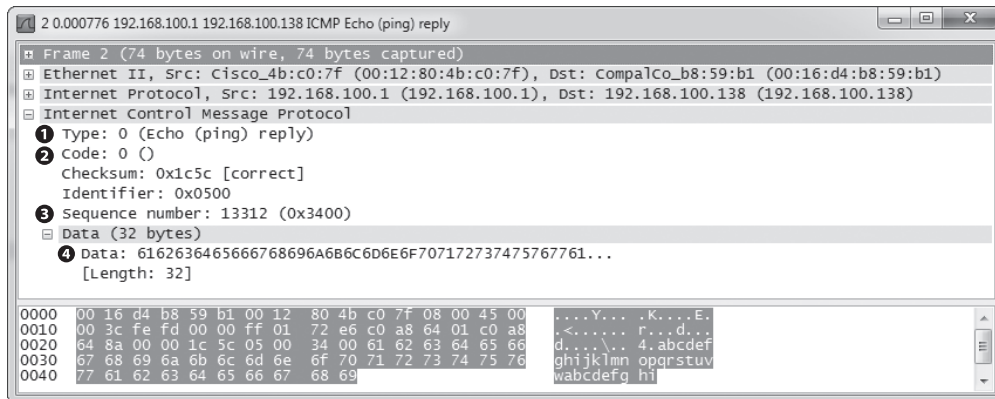


図6-33 ICMPエコー応答パケット



ICMPエコー要求に含まれるランダムなテキスト文字列は、攻撃者の興味の対象となり得ます。攻撃者は、この情報により標的とする機器のOSを突き止めることができるからです。また攻撃者が秘密の通信手段として、このフィールドに小さなデータを埋め込むことも可能です。

6.5.4 traceroute

icmp_traceroute.pcap

tracerouteは、ある機器から別の機器までの経路を調べるのに利用されます。単純なネットワークであれば、経路上にはルータがひとつだけ、あるいはひとつもない場合もあるでしょう。しかし複雑なネットワークになると、パケットは最終目的地へ到達するまでに、何十ものルータを通過しなければならないので、トラブルが発生した場合にそれを解決できるよう、パケットの経路を正確に追跡することが重要となります。

ICMPを使うことで（少しだけIPの助けを借りて）、tracerouteはパケットがたどる経路を突き止めます。icmp_traceroute.pcapファイルの最初のパケットは、先ほど見たエコー要求とかなりよく似ています（図6-34）。

ぱっと見ると、このパケットは、192.168.100.138から4.2.2.1①への、単純なエコー要求①のように見えます。パケットのICMP部分は、エコー要求パケットの形式とまったく同じです。しかしながら、このパケットのIPヘッダを展開すると、ひとつ奇異な値があることに気づくはずで、パケットのTTLが1②、つまり最初のルータに届いた時点でパケットが消失することを意味しています。宛先の4.2.2.1はインターネットアドレスなので、送信元と宛先の間には最低1台のルータが存在するはずであり、つまりこのパケットは宛先へ届かないことになります。実は、tracerouteはこのパケットを最初のルータまでしか届かなくさせることで機能を実現しているので、これはまったく問題ありません。

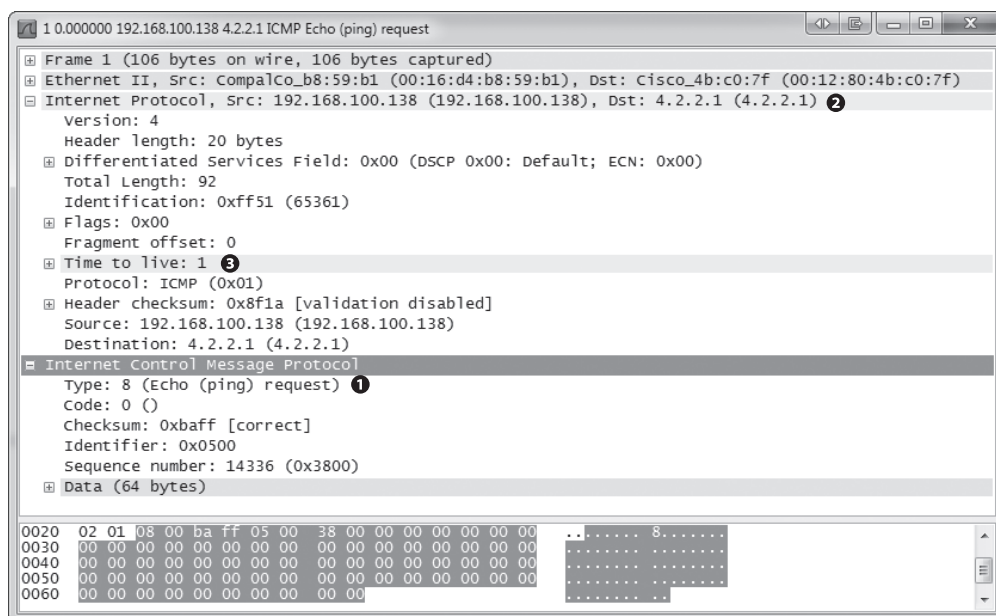


図6-34 TTL値が1のICMPエコー要求パケット

2番目のパケットは、予想どおり最初のルータからの応答です(図6-35)。パケットが192.168.100.1の機器に到達した時点で、TTLが0に減らされるため、パケットはこれ以上転送されないことになり、ルータはICMPレスポンスを返します。このパケットのタイプは11①、コードは0②で、転送の途中でTTLが時間切れになってしまったために、宛先に到達しなかったことを意味しています。

このICMPパケットはICMP部分の末尾に、IPヘッダのコピー③と最初のエコー要求で送られたICMPデータ④を含んでいるため、「ダブルヘッドパケット」と呼ばれる場合があります[†]。この情報はトラブルシューティングの際に非常に役立ちます。

TTL値1でパケットを再送信するという処理が、7番目のパケットまで2回繰り返されます。7番目のパケットでも最初のパケットと同様のことが行われますが、今回はIPヘッダのTTL値が2に設定されているので、パケットは2番目のルータまで到達することになります。予想どおり、12.180.241.1のルータから、同じように宛先到達不能のメッセージとTTL切れのメッセージが送られてきます。最終的な宛先である4.2.2.1に到達するまで、TTL値を1ずつ増やして処理が続きます。

まとめると、tracertの処理とは、経路上にある各ルータとやり取りし、宛先までの経路図を作成するというものです。図6-36がこの経路図です。

[†] 監訳注：監訳者の周囲では、この呼称は聞いたことがありませんでした。

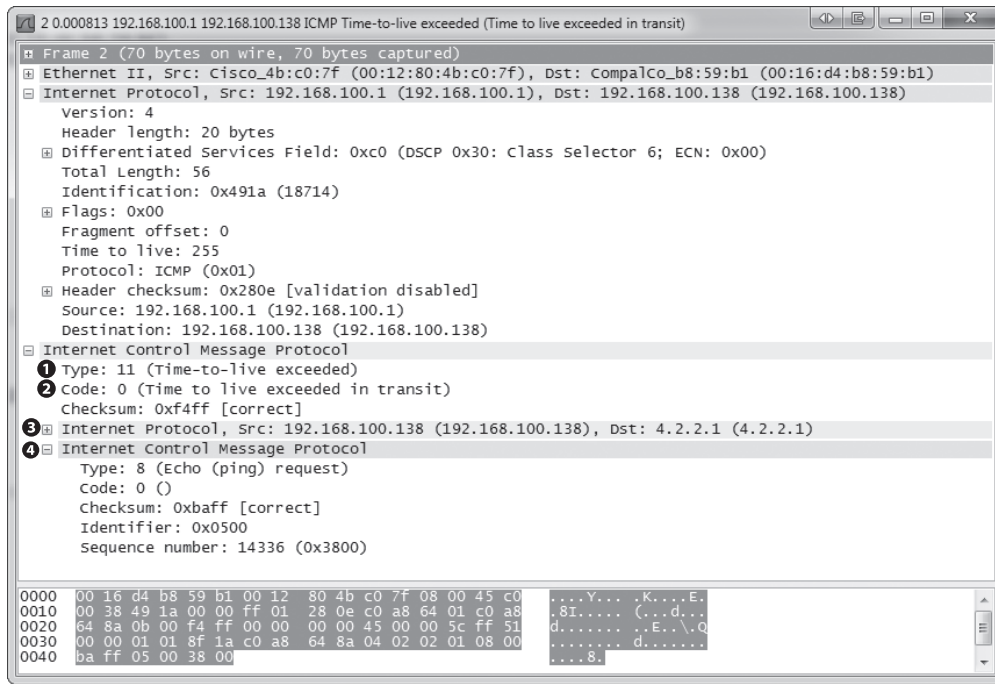


図6-35 経路の最初のルータからのICMPレスポンス

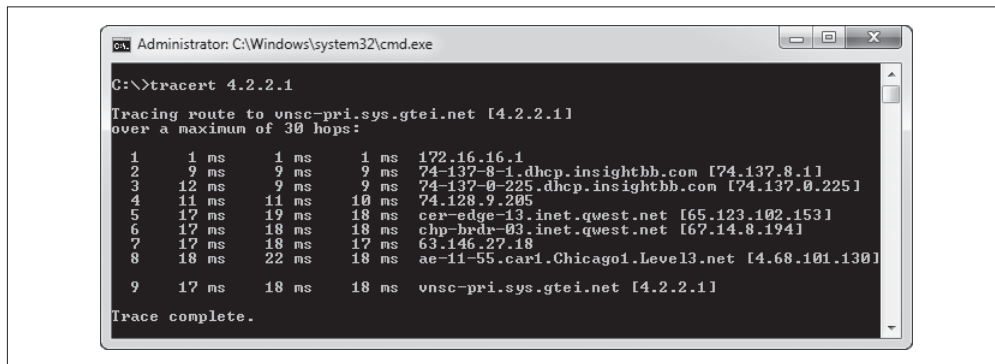


図6-36 tracerouteからの出力例



ここではICMPのみを使うWindowsのtracertについて説明しています[†]。Linuxのtracerouteはもう少し多才で、経路情報の取得に他のプロトコルを利用することができます。

[†] 監訳注：Windowsでは、歴史的経緯でコマンド名がtracertとなっています。

本書を通じて見ていきますが、ICMPには多種多様な機能があります。シナリオでパケット解析を行っていく中でICMPをさらに頻繁に用いることになります。

本章では、パケット解析のプロセスで調査することになるプロトコルの中で最重要のもの的一端を紹介しました。IP、TCP、UDP、ICMPはネットワーク通信の基礎であり、日々の作業において必須のものです。次章では、アプリケーション層で知っておきたいプロトコルについて見ていきます。

7章

知っておきたい上位層プロトコル

この章では、引き続き個々のプロトコルの機能と、これらのプロトコルがWiresharkでどのように見えるかについて見ていきます。ここではもっとも一般的な上位層（第7層）プロトコルであるDHCP、DNS、HTTPの3つについて説明します。

7.1 DHCP

昔のネットワークでは、ネットワーク上で通信を行いたければ、機器に手作業でアドレスを割り当てる必要がありました。しかしネットワークが拡大するにつれ、この手作業の処理はたちまち面倒なものへとなっていきます。この問題を解決するために開発されたのが、ネットワークに接続した機器に自動的にアドレスを割り当てる **BOOTP** (Bootstrap Protocol) です。のちにBOOTPは、より洗練された **DHCP** (Dynamic Host Configuration Protocol) によって置き換えられました。

DHCPは、機器が自動的にIPアドレス（およびDNSサーバやルータなどその他の重要なネットワーク情報のアドレス）を取得できるようにする、アプリケーション層プロトコルです。今日のDHCPサーバのほとんどは、ネットワークのデフォルトゲートウェイやDNSサーバのアドレスといった情報もクライアントに提供しています。

7.1.1 DHCPパケット構造

DHCPパケットは、相当量の情報をクライアントに提供します。図7-1のように、DHCPパケット内には以下のフィールドがあります。

オペコード (OpCode)

パケットがDHCP要求かDHCP応答かを示します。

ハードウェアタイプ

ハードウェアアドレスのタイプ (10MB イーサネット、IEEE 802、ATM など)。

ハードウェアアドレス長

ハードウェアアドレスのアドレス長。

ホップ数

DHCPサーバの発見を補助するリレーエージェントが使用します。

トランザクションID

要求と応答を対にするために使われるランダムな数字。

経過秒数

クライアントがDHCPサーバにアドレスを要求してからの秒数。

フラグ

DHCPクライアントが受け取るトラフィックのタイプ（ユニキャスト、ブロードキャストなど）。

クライアントIPアドレス

クライアントのIPアドレス（IPアドレスフィールドから得られたもの）。

割り当てIPアドレス

DHCPサーバから提供されたIPアドレス（最終的にはクライアントのIPアドレスフィールド値となります）。

サーバIPアドレス

DHCPサーバのIPアドレス。

DHCP				
ビットオフセット	0-15		16-31	
0	オペコード	ハードウェアタイプ	ハードウェアアドレス長	ホップ数
32	トランザクション ID			
64	経過秒数		フラグ	
96	クライアント IP アドレス			
128	割り当て IP アドレス			
160	サーバ IP アドレス			
196	ゲートウェイ IP アドレス			
228 超	クライアントハードウェアアドレス (16バイト)			
	サーバホスト名 (64バイト)			
	ブートファイル (128バイト)			
	オプション			

図7-1 DHCPパケット構造

ゲートウェイIPアドレス

ネットワークのデフォルトゲートウェイのIPアドレス。

クライアントハードウェアアドレス

クライアントのMACアドレス。

サーバホスト名

サーバのホスト名 (オプション)。

ブートファイル

DHCPが使うブートファイル (オプション)。

オプション

パケットの構造を拡張してDHCPの機能を拡張するために使用します。

7.1.2 DHCP更新処理**dhcp_nlease_renewal.pcap**

DHCPの主な目的は、更新処理でクライアントにアドレスを割り当てることです。**DHCP更新処理**は、dhcp_nlease_renewal.pcapファイルを見るとわかるように、特定のクライアントとDHCPサーバ間で発生します。DHCP更新処理は、DISCOVER、OFFER、REQUEST、ACKNOWLEDGMENTの4種類のDHCPパケットを使うため、DORA処理とも呼ばれています (図7-2)。それぞれのDORAパケットを見てみましょう。

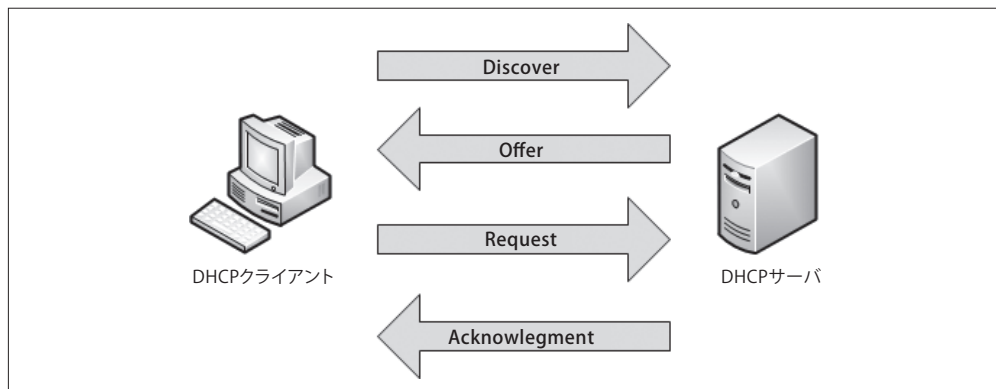


図7-2 DHCP DORA処理

7.1.2.1 DISCOVERパケット

キャプチャファイルでわかるように、最初のパケットは0.0.0.0の68番ポートから255.255.255.255の67番ポートへと送られます。なぜ0.0.0.0なのかというと、このクライアントにはまだIPアドレスがないからです。パケットが255.255.255.255に送られるのは、このアドレスがネットワークに依存しないブロードキャストアドレスであり、パケットがネットワーク上のすべての機器に確実に届くようにし

てくれるからです。機器はDHCPサーバのアドレスを知らないので、最初のパケットは、待ち受けているDHCPサーバを見つけるために送られます。

[Packet Details] ペインを見ると、まずDHCPがトランスポート層プロトコルとしてUDPを使っていることに気づきます。DHCPはクライアントが要求した情報を受信するまでの速度を非常に重視しているためです。またDHCPは信頼性を維持する機構を持っているため、UDPが最適なのです。[Packet Details] ペインで最初のパケットのDHCPセクションを調べれば、DISCOVER処理の詳細を見ることができます(図7-3)。



Wiresharkは、現在でもDHCPをBOOTPとして扱っているため、[Packet Details] ペインでは、DHCPセクションの代わりにBootstrap Protocolセクションが確認できます。ただし、本書では、これをパケットのDHCPセクションとして扱います。

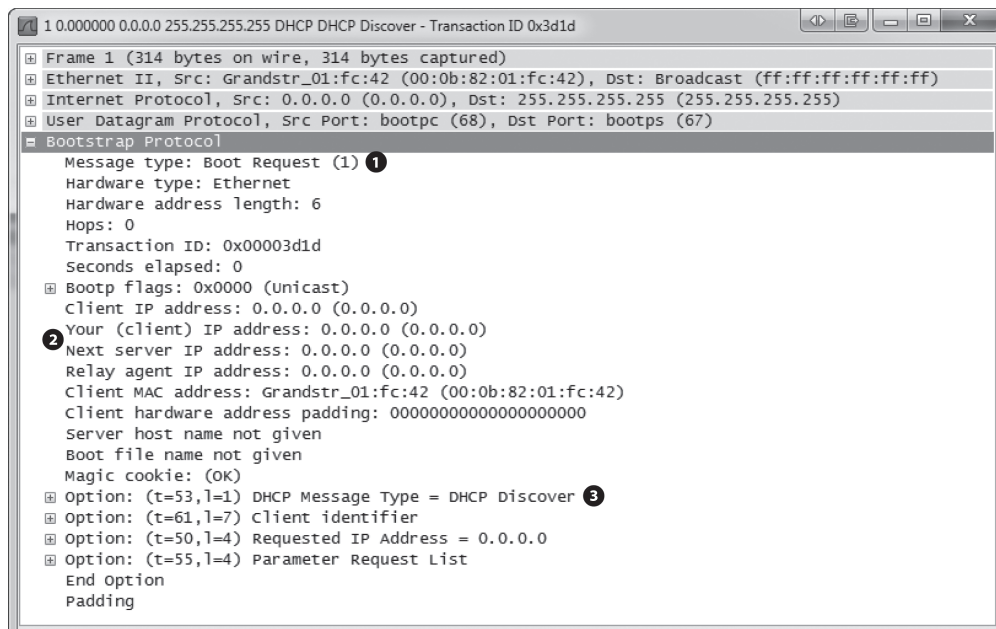


図7-3 DHCPDISCOVERパケット

このパケットがリクエストであるのは、Message Type フィールドが1であることからわかります①。リクエストパケットのほとんどのフィールドは、前のセクションのDHCPフィールドの一覧に基づき、(IPアドレスフィールド②のように) 空白か、見ればすぐわかる設定になっています。このパケットの要点は4つのオプションフィールドにあります。

DHCP Message Type

これはオプションタイプ「53 (t=53)」であり、長さが1、値も1となっています❸。これらの値はDHCPDISCOVERパケットであることを示しています。

Client Identifier

IPアドレスを要求しているクライアントについての追加情報を提供します。

Requested IP Address

クライアントが割り当てを要求するIPアドレスを提示します（通常は以前使ったIPアドレス）。

Parameter Request List

クライアントがDHCPサーバから受け取りたい設定項目（重要なネットワーク機器のIPアドレス）の一覧。

7.1.2.2 DHCPOFFERパケット

2番目のパケットのIPヘッダには正規のIPアドレスの一覧が含まれており、このパケットが192.168.0.1から192.168.0.10へ送られていることを示しています（図7-4）。クライアントにはまだ192.168.0.10のアドレスが割り当てられていないので、サーバはまずARPから提供されたハードウェアアドレスを使ってクライアントとの通信を試行します。通信できなかった場合は、ブロードキャストを行います。

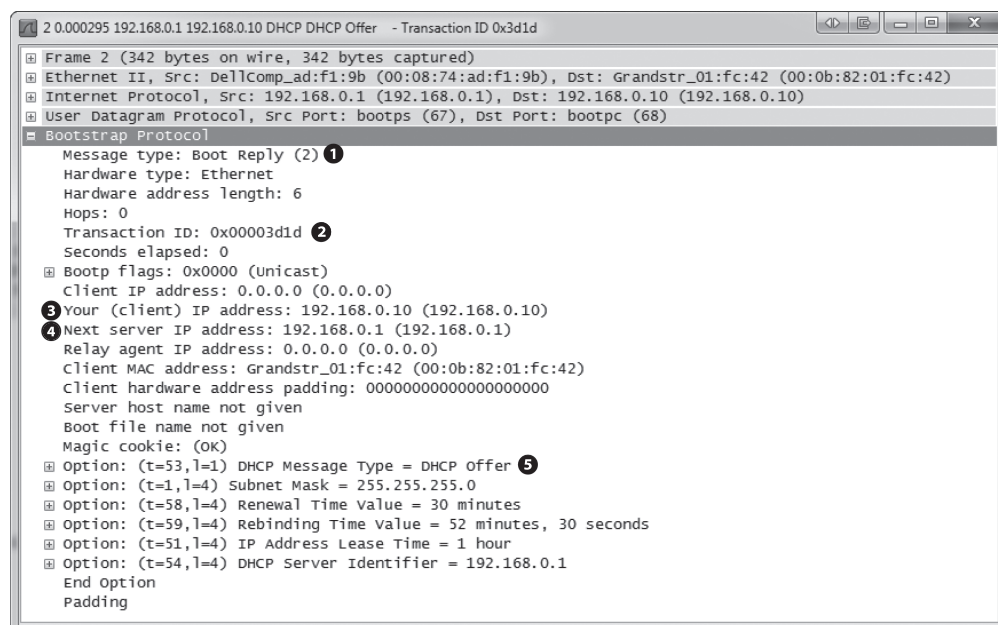


図7-4 DHCPOFFERパケット

2番目のパケットのDHCP部分は「OFFERパケット」と呼ばれ、Message Typeが応答であることを示しています❶。パケットには先ほどのパケットと同じトランザクションIDが含まれているので❷、この応答が要求に対するものであるとわかります。

OFFERパケットは、DHCPサーバがクライアントにサービスを提供するために送るものです。パケットには、DHCPサーバ自身の情報と、クライアントに提供したいアドレスの情報とが含まれています。図7-4では、割り当て（クライアント）IPアドレスのフィールドで、192.168.0.10がクライアントに提供されています❸。Next Server IP Address フィールドにある192.168.0.1という値❹は、DHCPサーバとデフォルトゲートウェイが同じIPアドレスを共有していることを示しています。

最初のオプションはパケットがDHCP Offerとして認識されていることを示しています❺。その後続くオプションにより、サーバからクライアントのIPアドレス以外の追加情報が提供されていることがわかります。提供されているのは次のような情報です。

- サブネットマスクが255.255.255.0
- 更新時間は30分
- 再バインディング (REBINDING) 時間は52分30秒
- IPアドレスリース期間は1時間
- DHCPサーバ識別子は192.168.0.1

7.1.2.3 DHCPREQUESTパケット

クライアントはDHCPサーバからの通知を受け取ると、図7-5のようにDHCPREQUESTパケットを送ります。

まだIPアドレスの取得プロセスが完了していないので、3番目のパケットもIPアドレス0.0.0.0から送ります❶が、パケットは、宛先のDHCPサーバを知っています。

Message Type フィールドは、パケットが要求パケットであることを示しています❷。キャプチャファイル内のパケットはすべて一連の更新処理を構成していますが、新たな要求/応答トランザクションとなるため、新しいトランザクションIDが付与されています❸。IPアドレス情報がすべて空白になっているという点で、DHCPDISCOVERパケットと似ています。

Option フィールドから❹、これがDHCPREQUESTパケットであることがわかります。Requested IP Addressが空白でなくなり、DHCP Server Identifier フィールドにもアドレスが入っています。

7.1.2.4 DHCPACKパケット

処理の最終段階としてDHCPサーバは、図7-6のように要求されたIPアドレスをクライアントにDHCPACKパケットで送り、この情報をデータベースに記録します。

これでクライアントにIPアドレスが割り当てられ、通信を始めることが可能になりました。

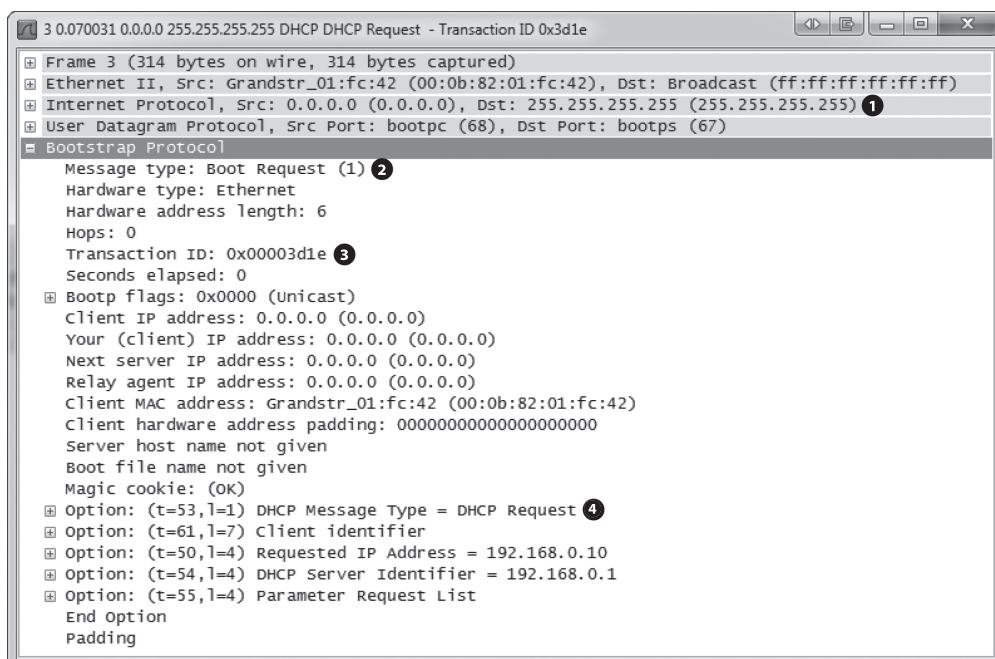


図7-5 DHCPREQUEST/パケット

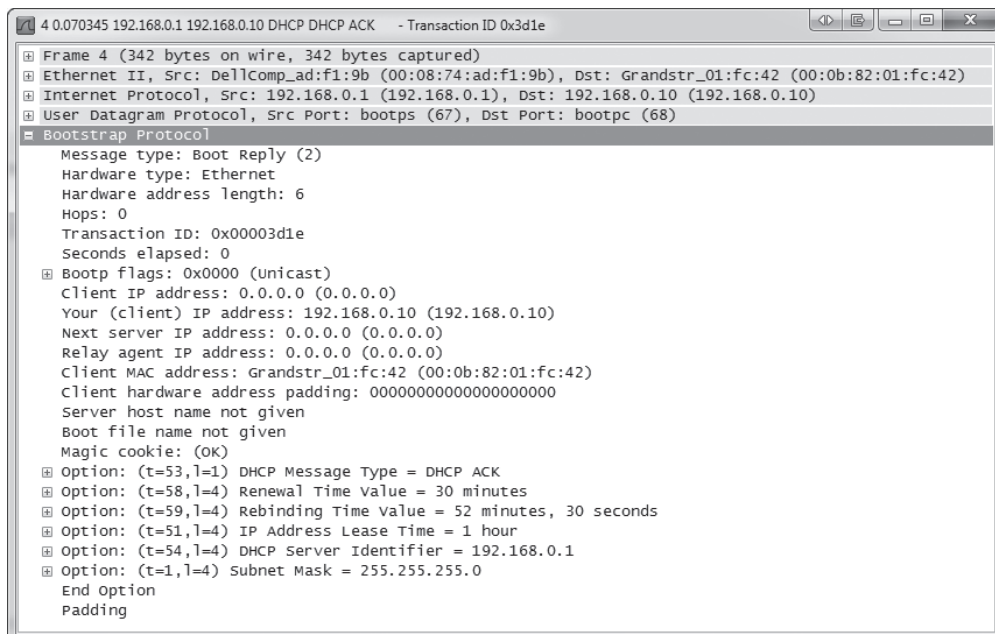


図7-6 DHCPACK/パケット

7.1.3 DHCPのリース更新

dhcp_inlease_renewal.pcap

DHCPサーバが機器にIPアドレスを割り当てることを、クライアントにIPアドレスをリース（貸し出し）するといいます。つまりクライアントは一定期間だけこのIPアドレスを使用でき、引き続き使用したい場合はリースの更新が必要になります。先述したDORAプロセスは、クライアントが初めてIPアドレスを取得する場合や、リース期限が切れている場合に発生し、いずれの場合も機器のリース期限が切れているものとして扱われます。

IPアドレスをリース中のクライアントが再起動すると、IPアドレスを再度要求するためにDORAプロセスの簡略版を実行しなければなりません。このプロセスをリース更新と呼びます。

リース更新では、DHCPDISCOVERYパケットとDHCPOFFERパケットは不要です。リース期限切れ更新でのDORA処理と比較すると、リース更新ではREQUESTとACKパケットのステップ以外必要ありません。リース更新のサンプルのキャプチャファイルがdhcp_inlease_renewal.pcapにあります。

7.1.4 DHCPオプションとメッセージタイプ

DHCPの柔軟性は、そのオプションにあります。ご覧のとおり、そのサイズも内容も多種多様にわたっています。パケット全体の大きさは、使用するオプションの組み合わせによって変わってきます。DHCPオプションの網羅的な一覧については、<http://www.iana.org/assignments/bootp-dhcp-parameters/>を参照してください。

すべてのDHCPパケットに必須となる唯一のオプションが、Message Type オプション（オプション53）です。このオプションは、パケットに含まれる情報を、DHCPクライアントまたはサーバがどのように処理するかを識別します。表7-1のように、Message Type オプションは8種類あります。

表7-1 DHCP Message Typeオプション

タイプ番号	メッセージタイプ	説明
1	DISCOVER	クライアントが使用可能なDHCPサーバを見つけるためのメッセージ
2	OFFER	サーバからクライアントへ送られるDISCOVERパケットへの応答メッセージ
3	REQUEST	クライアントからサーバへ送られる設定情報要求メッセージ
4	DECLINE	クライアントからサーバへ送られる設定情報無効通知メッセージ
5	ACK	サーバからクライアントへ送られる設定情報の提供メッセージ
6	NAK	サーバからクライアントへ送られる設定情報の提供拒否メッセージ
7	RELEASE	クライアントからサーバへ送られるリースの中断メッセージ。これにより設定情報が解放される
8	INFORM	クライアントがすでにIPアドレスを取得している際に、設定情報をクライアントがサーバに問い合わせるメッセージ

7.2 DNS

DNS（Domain Name System）は、インターネットプロトコルの中でも最重要なもののひとつです。DNSはwww.google.comなどのような名前（DNS名）を74.125.159.99のようなIPアドレスに変換し

ます。これにより、通信したいネットワーク機器のIPアドレスがわからないときでも、DNS名でアクセスすることができます。

DNSサーバはIPアドレスとDNS名の対応付けを行う「リソースレコード」というデータベースを持ち、これをクライアントやほかのDNSサーバと共有しています。



DNSサーバのアーキテクチャはかなり複雑なので、ここでは一般的なDNSトラフィックを扱います。<http://www.isc.org/community/reference/RFCs/DNS>で、DNS関連のさまざまなRFCを参照することができます。

7.2.1 DNSのパケット構造

図7-7からわかるように、DNSのパケット構造は、先ほどのDHCPとは若干異なります。DNSパケットのフィールドは次のようになっています。

DNS識別子

DNSクエリに対するレスポンスを対応付けるために用いられます。

QR (Query/Response)

パケットがDNSクエリなのか、レスポンスなのかを示します。

OpCode

メッセージに含まれる照会のタイプを定義します。

AA (Authoritative Answers)

レスポンスパケットでこの値がセットされていた場合、ドメインに権威のあるネームサーバからのレスポンスであることを示します。

TC (Truncation)

レスポンスが長すぎてパケット内に収まらないため、切り詰められたことを意味します。

RD (Recursion Desired)

DNSクエリにこの値がセットされていると、DNSクライアントは目的のネームサーバに対して、要求した情報がない場合に再帰クエリを要求します。

RA (Recursion Available)

レスポンスにこの値がセットされていると、ネームサーバが再帰クエリをサポートしていることを示します。

Z (Reserved)

RFC 1035で、すべて0にセットされることが規定されています。しかしRCodeフィールドの拡張として用いられる場合もあります。

RCode (Response Code)

DNSレスポンスで使われ、エラーの有無を示します。

Question Count

Questionセクションに含まれているエントリ数。

Answer Count

Answerセクションに含まれているエントリ数。

Name Server Count

Authorityセクションに含まれているネームサーバのリソースレコード数。

Additional Records Count

Additional Informationセクションに含まれているその他のリソースレコード数。

Questionセクション

DNSサーバに送られた1つ以上のクエリを含む可変長セクション。

Answersセクション

クエリのレスポンスとなる1つ以上のリソースレコードを含む可変長セクション。

Authorityセクション

名前解決処理を継続する際に使用できる権威あるネームサーバを示すリソースレコードを含む可変長セクション。

Additional Informationセクション

クエリのレスポンスとして必須ではない関連情報を保持するリソースレコードを含む可変長セクション。

DNS												
ビットオフセット	0-15				16-31							
0	DNS識別子				QR	OpCode	A A	T C	R D	R A	Z	RCode
32	Question Count				Answer Count							
64	Name Server Count				Additional Records Count							
96	Questionセクション				Answersセクション							
128	Authorityセクション				Additional Informationセクション							

図7-7 DNSのパケット構造

7.2.2 単純なDNSクエリ

dns_query_response.pcap

DNSはクエリ／レスポンスの形式で機能します。DNS名をIPアドレスに変換したいクライアント

はDNSサーバにクエリを送信し、サーバはレスポンスとして要求された情報を返却します。キャプチャファイル `dns_query_response.pcap` からわかるように、もっとも単純な場合、この処理に必要なのは2個のパケットのみです。

最初のパケットは、クライアント192.168.0.114からサーバ205.152.37.23の53番ポート（DNSの標準ポート）に送られたDNSクエリです（図7-8）。このパケットのヘッダを調べると、DNSもまたUDPを使っているのがわかります❶。

パケットのDNS部分を見ると、先頭のほうの小さなフィールドが、ひとつのフラグセクションへと集約されているのがわかります。このセクションを展開すると、メッセージが標準クエリであり❷、切り詰められておらず、再帰を要求されていることがわかります（再帰については後述します）。Queriesセクションを展開すると、クエリがひとつだけ確認できます。ここでは、`wireshark.org` という名前に対するホスト（type A）インターネット（IN）アドレスのクエリを確認できます❸。つまりこのパケットは「`wireshark.org` ドメインに割り当てられたIPアドレスは？」と尋ねているわけです。

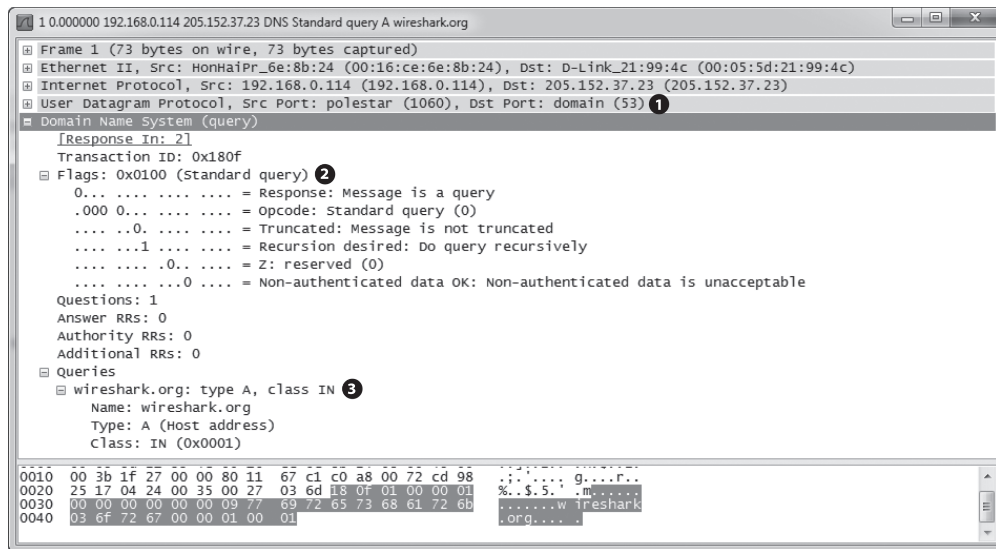


図7-8 DNSのクエリパケット

このリクエストに対するレスポンスが、図7-9で示している2番目のパケットです。このパケットには同一の識別子が付いているので❶、元々のクエリに対応するレスポンスが含まれていることがわかります。

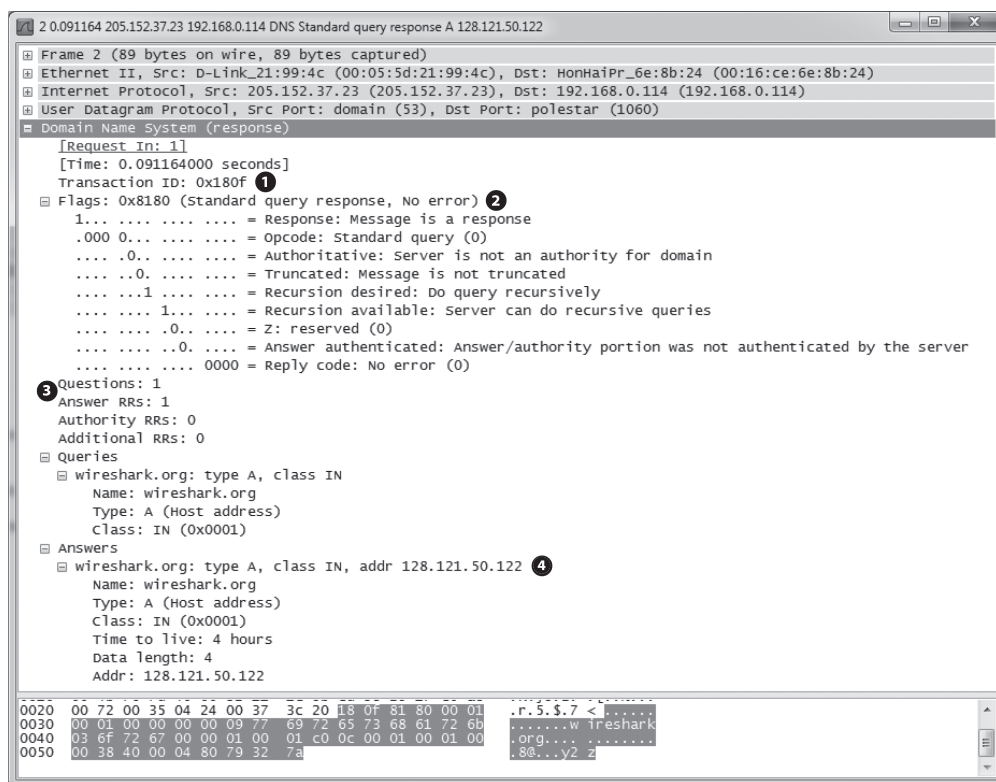


図7-9 DNSのレスポンスパケット

Flagsセクションを見ると、これがレスポンスであり、必要であれば再帰が可能なのがわかります②。このパケットには1つの問い合わせと1つのリソースレコードしか含まれておらず③、元々の問い合わせとそのレスポンスが入っています。Answersセクションを展開すると、wireshark.orgのIPアドレスは128.121.50.122というクエリに対するレスポンスが確認できます④。クライアントはこの情報をもとにIPパケットを生成し、wireshark.orgとの通信を開始します。

7.2.3 DNSの問い合わせタイプ

Typeフィールドは、DNSのクエリとレスポンスで使われる、リソースレコードのタイプを示します。一般的なリソースレコードのタイプを表7-2にまとめました。通常のトラフィックや本書を通じてこれらのタイプを目にすることになるでしょう。

なお、表7-2は網羅的な一覧ではありません。DNSのリソースレコードのタイプを網羅的に参照したい場合は、<http://www.iana.org/assignments/dns-parameters/>を参照してください。

表7-2 一般的なリソースレコードのタイプ

値	タイプ	説明
1	A	IPv4ホストのアドレス
2	NS	権威のあるネームサーバ
5	CNAME	ホストの別名（エイリアス）
15	MX	メールサーバ
16	TXT	テキスト文字列
28	AAAA	IPv6ホストのアドレス
251	IXFR	増分ゾーン転送
252	AXFR	完全ゾーン転送

7.2.4 DNSの再帰

dns_recursivequery_client.pcap

dns_recursivequery_server.pcap

インターネットのDNSは階層的な構造を取っているため、クライアントから送られたクエリに対するレスポンスを送信するときは、DNSサーバ同士が通信する必要があります。内部のDNSサーバは、ローカルのイントラネットサーバのDNS名とIPアドレスの対応付けは知っているはずですが、GoogleやDellのIPアドレスは知るよしもありません。

DNSサーバがIPアドレスを検索する場合、リクエストを行ったクライアントの代理として別のDNSサーバにクエリを行います。DNSサーバがクライアントのように行動するこの処理を「再帰」と呼びます。

DNSクライアントとDNSサーバの両方の視点から再帰プロセスを確認してみましょう。まず、dns_recursivequery_client.pcap ファイルを開きます。このファイルには、クライアントのDNSトラフィックをキャプチャしたふたつのパケットが含まれています。最初のパケットはDNSクライアント172.16.0.8から、DNSサーバ172.16.0.102へと送られたクエリです（図7-10）。

このパケットのDNS部分を開くと、これがwww.nostarch.comというDNS名①のAタイプレコードに対する標準クエリであることがわかります。Flagsセクションを展開すると、このパケットの詳細がわかり、再帰が要求されていることもわかります②。

2番目のパケットは、予想どおり最初のクエリに対応するものです（図7-11）。

このパケットのトランザクションIDはクエリのIDと一致しており①、エラーもないので、www.nostarch.comのAソースレコードを受け取ります②。

このクエリが再帰によって応答されているのを確認する唯一の方法は、dns_recursivequery_server.pcap ファイルにあるように、再帰クエリが行われた際のDNSサーバのトラフィックをキャプチャすることです。このファイルは、クエリが行われた際のローカルDNSサーバ上のトラフィックをキャプチャしたものです。最初のパケットは、先ほどキャプチャしたファイルで見たクエリと同一です。DNSサーバはクエリを受け取り、ローカルのデータベースと照会しましたが、DNS名（nostarch.com）と一致するIPアドレスは何かという問い合わせの回答を見つけられませんでした。パケットには再帰要求ビットがセットされているので、DNSサーバが答えを見つけるためにほかのDNSサーバに問い合わせているのが、2番目のパケットからわかります。

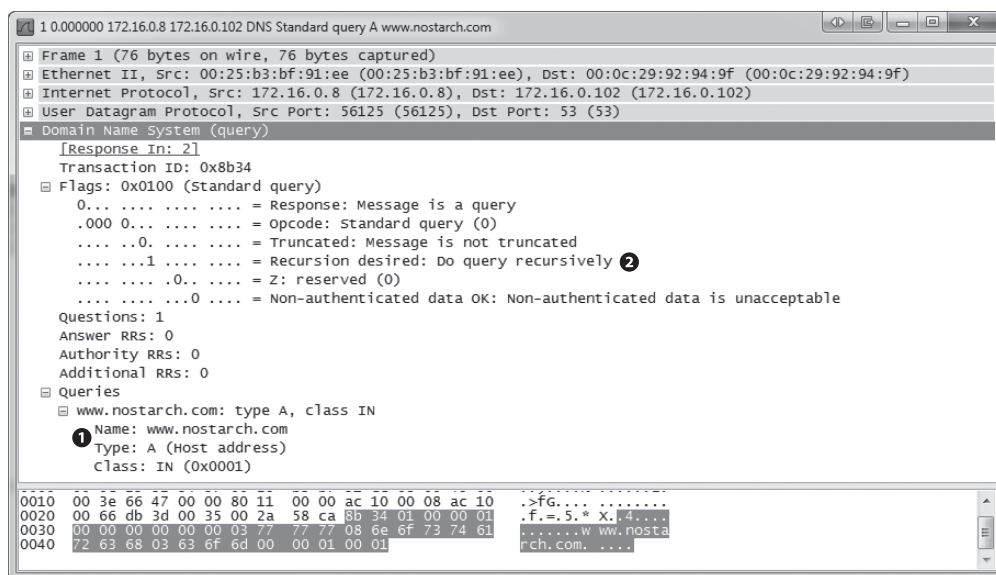


図7-10 再帰要求ビットがセットされたDNSクエリ

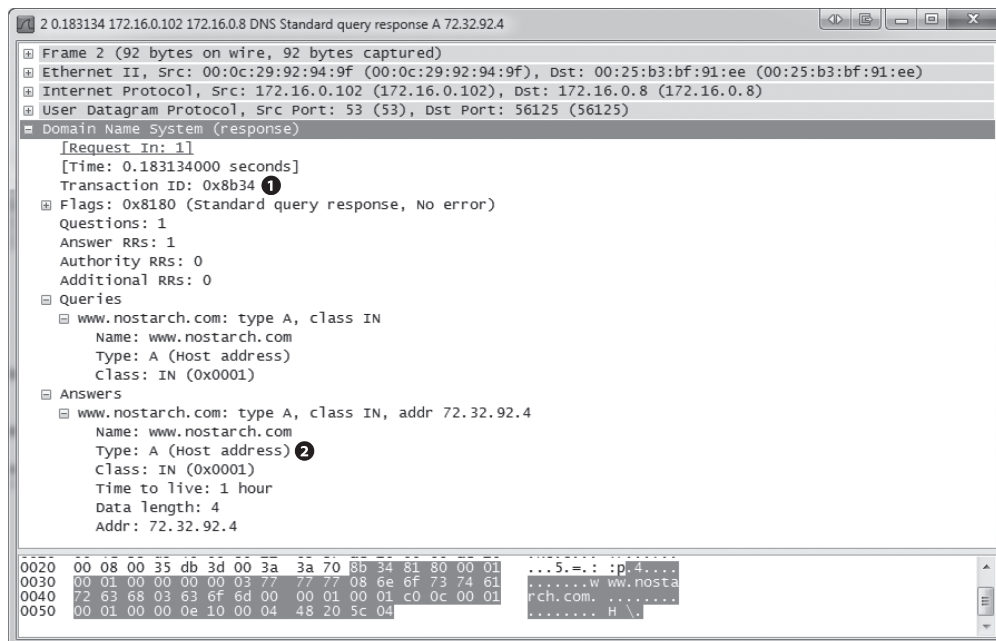


図7-11 DNSクエリのレスポンス

2番目のパケットでは図7-12のように、172.16.0.102のDNSサーバが4.2.2.1に新たなクエリを送っています。4.2.2.1は、上位に対するリクエストの転送先として設定されたサーバです。このクエリは

元々のクエリから生成されたもので、DNSサーバをクライアントへと変貌させています。

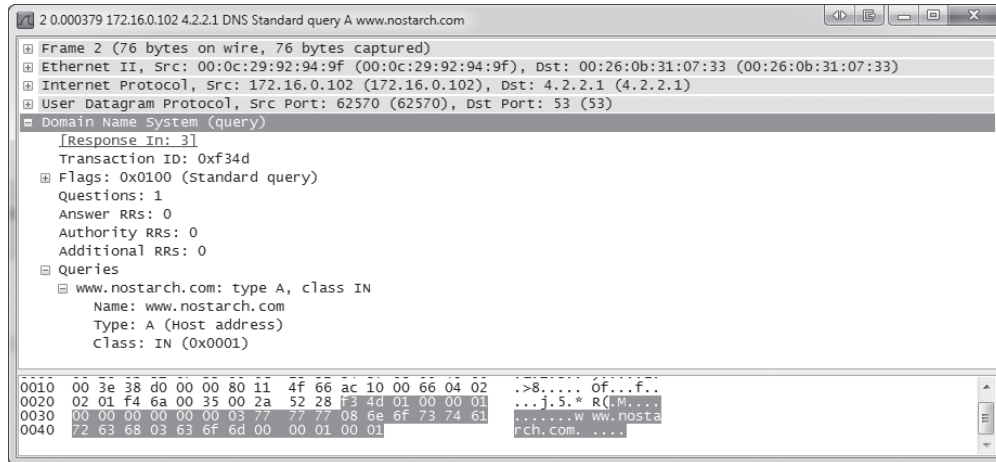


図7-12 再帰クエリ

先ほどのキャプチャファイルとはトランザクションIDが異なるため、これが新たなクエリであることがわかります。サーバ4.2.2.1がこのパケットを受け取ると、ローカルDNSサーバは図7-13のようなレスポンスを受け取ります。

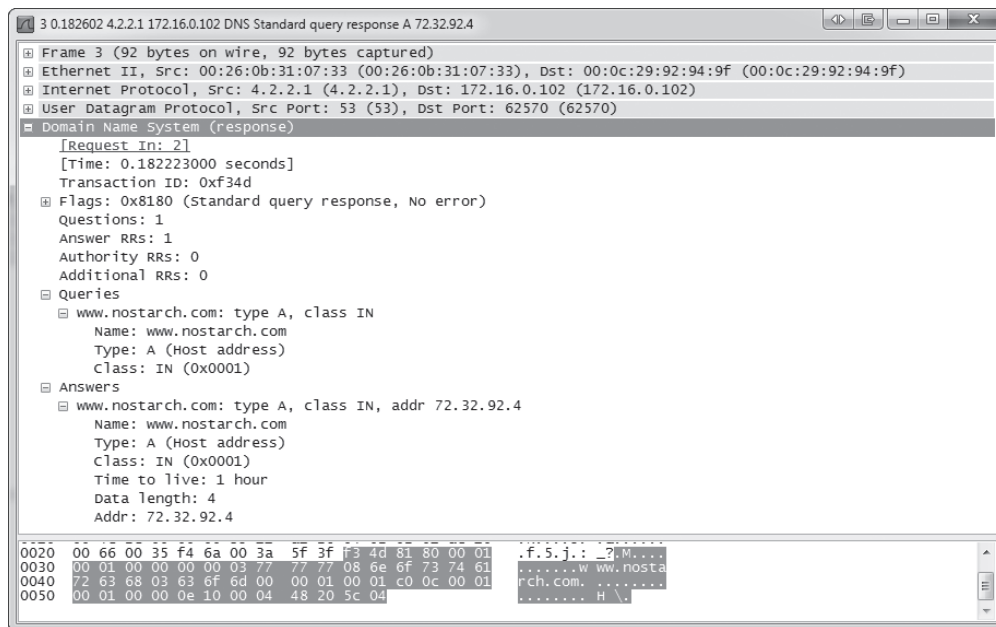


図7-13 再帰クエリへのレスポンス

レスポンスを受け取ると、ローカルDNSサーバは4個目の最後のパケットに要求された情報を格納して、DNSクライアントに送信します。

このサンプルでは再帰は1回だけですが、1回のDNS要求で再帰が何度も行われることがあります。ここではDNSサーバ4.2.2.1から回答を受け取りましたが、DNSサーバは回答を得るために、ほかのDNSサーバに再帰クエリを再送信する場合があります。1つの簡単なクエリが最終的に適切な回答を得るまでに、世界中を巡ることがあるのです。図7-14は再帰クエリの処理を図式化しています。

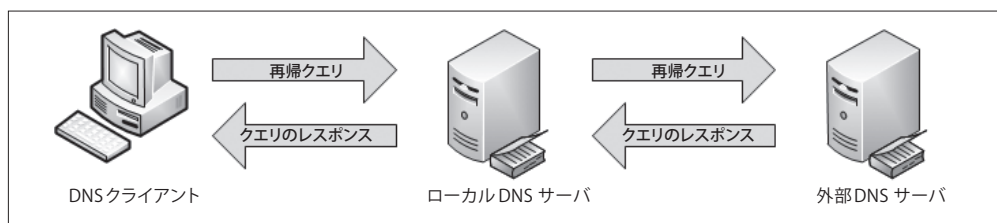


図7-14 再帰クエリ

7.2.5 DNSゾーン転送

`dns_axfr.pcap`

DNSゾーンとは、DNSサーバが管理を任された名前空間（DNS名のグループ）です。仮に「Emma's Diner」が、emmasdiner.comを管理するDNSサーバを持っているとしましょう。この場合、emmasdiner.comのIPアドレスを照会しようとする内外の通信機器は、そのゾーンに権限を持っている該当のDNSサーバとやり取りする必要があります。Emma's Dinerが成長し、DNS名前空間のメール部分（仮にmail.emmasdiner.comとしておきます）を扱うために、2番目のDNSサーバを追加した場合、そのサーバはmailサブドメインに権限を持つことになります。図7-15のように、サブドメインがもっと必要になれば、さらにDNSサーバを追加することができます。

ゾーン転送は、通常冗長化を行う目的で2つの機器の間でゾーンデータが転送される際に発生します。たとえば複数のDNSサーバを持つ組織の管理者は、通常プライマリのDNSサーバが停止した場合に備え、プライマリDNSサーバのDNSゾーン情報の複製を保持するようセカンダリのDNSサーバを設定しています。ゾーン転送には次の2種類があります。

完全ゾーン転送（AXFR）

機器間でゾーン全体を転送します。

増分ゾーン転送（IXFR）

ゾーン情報の一部のみを転送します。

ファイルdns_axfr.pcapには、172.16.16.164と172.16.16.139という2つの機器間での完全ゾーン転送のサンプルが含まれています。

このファイルを初めて見ると、UDPパケットではなくTCPパケットがあるので、間違ったファイル

を開いたのではないかと思います。確かにDNSはUDPを使いますが、ゾーン転送などの一部のタスクについてはTCPを使います。大量のデータを転送する場合、TCPのほうが信頼度が高いからです。このキャプチャファイルの最初の3つのパケットは、TCPの3ウェイハンドシェイクです。

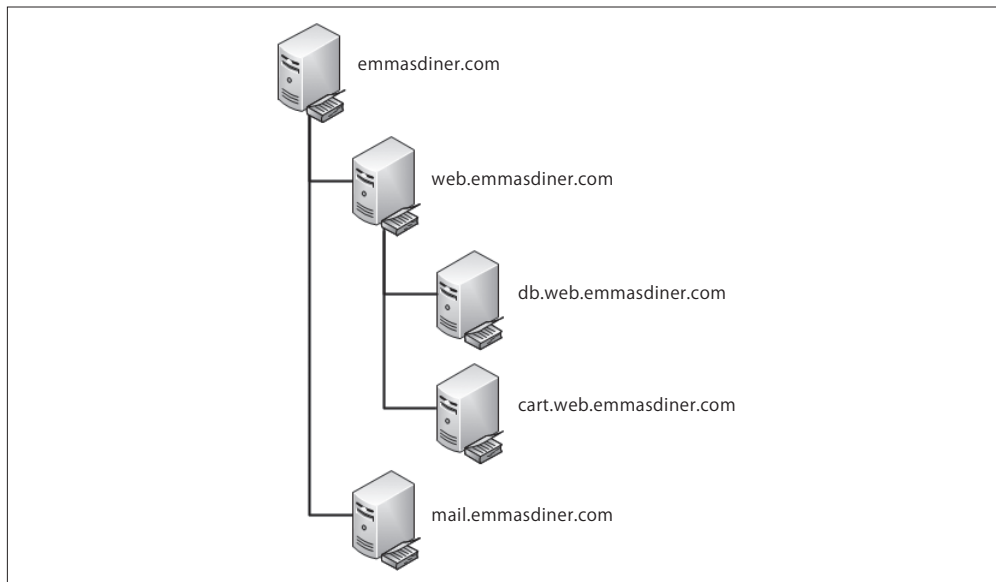


図7-15 DNSゾーンは名前空間の責任範囲を分割する

4個目のパケットは、172.16.16.164と172.16.16.139間のゾーン転送リクエストで始まっています。このパケットにDNS情報は含まれていません。「TCP segment of reassembled PDU」となっているのは、ゾーン転送リクエストのデータが、複数のパケットで送られたためです。4番目と6番目のパケットにデータが含まれています。5番目のパケットは4番目のパケットを受け取ったというACKパケットです。パケットがこのようなわかりやすい形で表示されるのもWiresharkの機能のひとつです。図7-16に示しているように、6番目のパケットを完全なDNSゾーン転送リクエストとして参照することができます。

ゾーン転送要求は標準クエリですが①、AXFRというレコードタイプを1つだけ要求します②。これは、サーバからDNSゾーン全体の受信を要求するという意味になります。サーバからのレスポンスは、図7-17の7番目のパケットのように、ゾーン全体のレコードが含まれたものとなります。ゾーン転送ではかなりの量のデータがやり取りされますが、これはまだ単純なサンプルです。ゾーン転送が完了すると、TCPティアダウン処理で終了します。

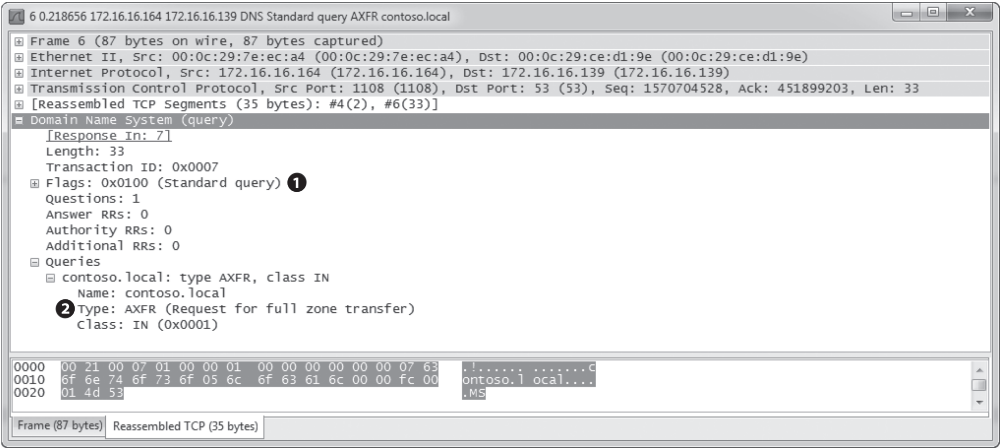


図7-16 DNS完全ゾーン転送要求

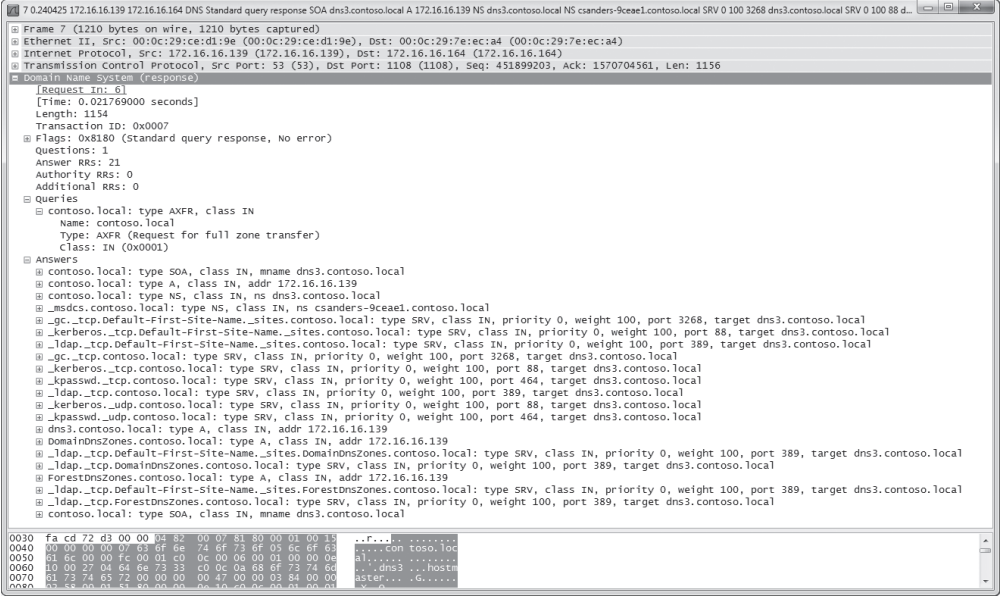
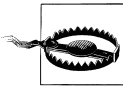


図7-17 DNS完全ゾーン転送の実行



ゾーン転送に含まれるデータが悪意を持った人の手に渡ると非常に危険です。DNSサーバを順に見ていくだけで、ネットワークの全体構造が描けてしまうからです。

7.3 HTTP

HTTP (Hypertext Transfer Protocol) は、World Wide Webの配信機構であり、WebブラウザがWebサーバに接続してWebページを閲覧する行為を実現しています。多くの組織では、HTTPのトラフィック量が他と比較して飛び抜けて多くなっています。Googleで検索したり、Twitterでツイートしたり、あるいはケンタッキー大学のバスケット試合の得点をESPN.comでチェックしたりするたびに、HTTPを使っているのです。

HTTPのパケット構造は、ここでは扱いません。こうしたパケットの内容は目的によってかなりの違いがあるため、その確認は読者にお任せします。ここではHTTPの実例をいくつか見ていきましょう。

7.3.1 HTTPでブラウズする

http_google.pcap

HTTPはWebブラウザを使ってWebページを閲覧する際によく利用されます。キャプチャファイルhttp_google.pcapには、TCPをトランスポート層プロトコルとして利用するHTTPトラフィックが含まれています。通信はクライアント172.16.16.128とGoogleのWebサーバ74.125.95.104との間の3ウェイハンドシェイクで始まります。

通信が確立されると、最初のHTTPパケットが、図7-18のようにクライアントからサーバへと送られます。

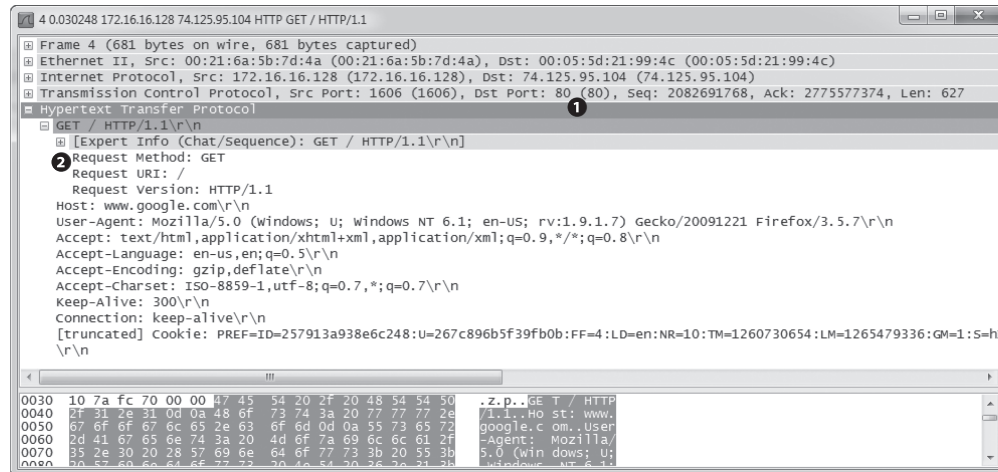


図7-18 最初のHTTP GETリクエストパケット

HTTPパケットはTCP上を、HTTP通信の標準ポートである80番ポート①(8080番もよく使われます)に対して送られます。

HTTPパケットには8つのメソッド(HTTP 1.1で定義されています)があり、これによってパケット送信元が宛先で実行させたいアクションが決まります。図7-18では、GETメソッドが発行されてお

り、リクエストURI (Uniform Resource Indicator) は/, リクエストバージョンはHTTP/1.1となっています❷。この情報から、クライアントはHTTPのバージョン1.1を使っているWebサーバのルートWebディレクトリ (/) をダウンロード (GET) するリクエストを送信していることがわかります。

引き続き、クライアントは自身の情報をWebサーバに送ります。情報には使用しているユーザーエージェント (ブラウザ)、ブラウザが対応可能な言語 (Accept-Language)、cookie情報 (キャプチャの末尾) などが含まれます。サーバはこの情報を利用して、互換性を満たすために、クライアントにどのデータを返却すべきかを判断します。

サーバは4番目のパケットでHTTP GETリクエストを受け取った後、TCP ACKをレスポンスとして返却し、次に6番目から11番目のパケットを使ってリクエストされたデータの転送を開始します。HTTPはクライアントとサーバ間でアプリケーション層のコマンドを発行するためにのみ使用されます。データ転送の段階になると、データストリームの先頭と末尾以外では、アプリケーション層の制御ありません。

図7-19を見ると、サーバからのデータが6番目と7番目のパケットで送られると、クライアントからのACKが8番目のパケットであり、さらに9番目と10番目のパケットでデータが送られると、11番目のパケットとしてACKが送られています。データ転送はHTTPが行っていますが、WiresharkではこれらのパケットはすべてHTTPではなく、TCPとして表示されます。

No.	Time	Source	Destination	Protocol	Info
6	0.101202	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]
7	0.101465	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]
8	0.101495	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082692395 Ack=2775580186 win=4218 Len=0
9	0.102282	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]
10	0.102350	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]
11	0.102364	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082692395 Ack=2775581694 win=4218 Len=0

図7-19 クライアントのブラウザとWebサーバ間のTCPデータ転送

データが転送されると、再構築されたデータストリームが図7-20のように表示されます。

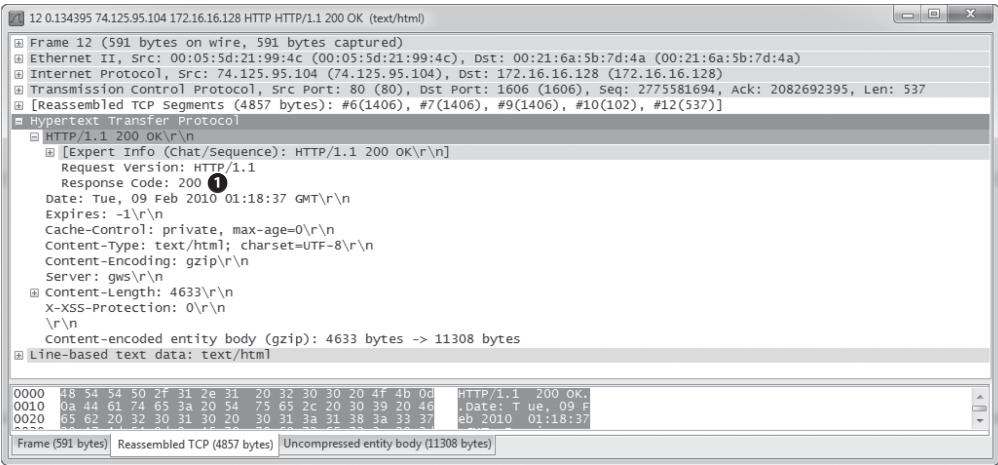


図7-20 ステータスコード200となっている末尾のHTTPパケット

HTTPは定義されているステータスコードの番号を用いてリクエストメソッドの成否を示します。このサンプルではステータスコードが200であり❶、これはリクエストメソッドの成功を意味します。パケットには、タイムスタンプをはじめ、コンテンツの符号化やWebサーバのパラメータ設定に関する追加の情報も含まれています。クライアントがこのパケットを受け取ると、トランザクションが完了します。

7.3.2 HTTPでデータをアップロードする

http_post.pcap

Webサーバからデータをダウンロードする処理を見てきたので、今度はアップロードするのを見てみましょう。ファイルhttp_post.pcapには、ユーザーがWebサイトにコメントを投稿するという、非常に単純なアップロードのサンプルが含まれています。最初の3ウェイハンドシェイクのあと、クライアント（172.16.16.128）は、図7-21のように、Webサーバ（69.163.176.56）にHTTPパケットを送ります。

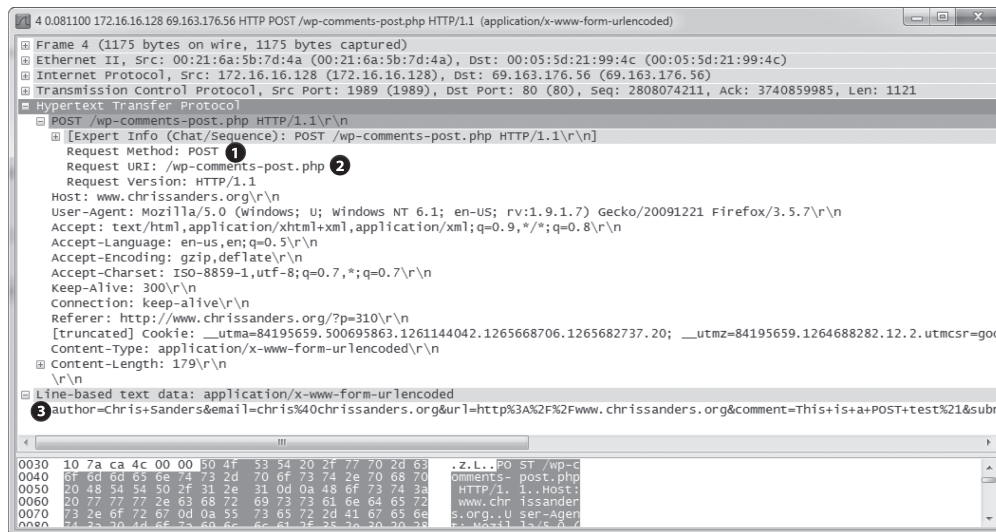


図7-21 HTTP POSTパケット

パケットはPOSTメソッドを使って❶Webサーバにデータをアップロードします。POSTメソッドでは、URIとして/wp-comments-post.php❷、Request versionとしてHTTP1.1が指定されています。アップロードされたデータの内容を見るには、パケットのLine-based text data部分を開きます❸。

POSTでデータが送信されると、ACKパケットが送られます。図7-22のように、サーバは6番目のパケットで、ステータスコード302❶「found」を送信してレスポンスを返却しています。

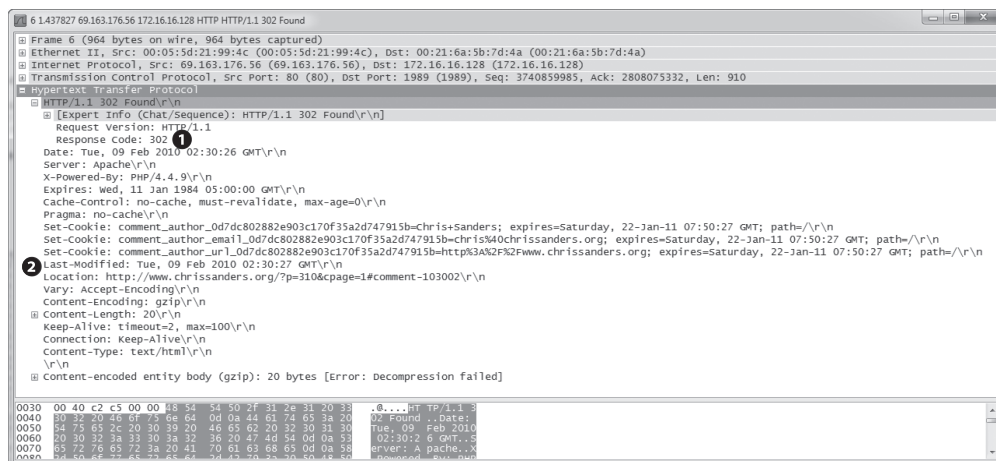


図7-22 HTTPステータスコード302はリダイレクトに使われる

ステータスコード302は、HTTPでは通常リダイレクトを示します。パケット内のLocationフィールドが、クライアントのリダイレクト先を示しています②。ここでは、コメントが投稿された最初のWebページがリダイレクト先となっています。最後にサーバがステータスコード200を送り、ページのコンテンツが続くいくつかのパケットを使って転送されて、処理が完了します。

7.4 まとめ

この章では、アプリケーション層のトラフィックを調査する際によく目にする、一般的なプロトコルを紹介しました。このあとに続く章では、多種多様な実践的なシナリオを通じて新しいプロトコルについて学ぶとともに、本章で紹介したプロトコルについても、未紹介のさまざまな機能について説明していきます。

個々のプロトコルをさらに詳しく知りたければ、関連するRFCを読むか、または『TCP/IP Guide』（Charles Kozierok 著、No Starch Press 2005年）を参照してください。また付録Cの推薦文献も参考にしてください。

8章

現実世界のシナリオの第一歩

この章からは、Wiresharkを使って現実のネットワークトラブルを分析する中で、パケット解析の本質を掘り下げていきます。第1部ではネットワークエンジニア、ヘルプデスク担当者、あるいはアプリケーション開発者として日々直面するシナリオを分析します。これらはいずれも筆者や同僚の実体験によるものです。Wiresharkを使って、Twitter、Facebook、ESPN.comのトラフィックを調べ、よく使われるサービスがどのように機能しているかを見ていきます。

第2部では、現実にかかるトラブルを取り上げます。それぞれのトラブルを取り巻く状況を解説するとともに、その時点でパケット解析者が入手していた情報を提供します。下調べを行ったら、パケット解析に移り、適切なパケットをキャプチャするための手法を説明し、解析作業を段階を踏んで説明します。パケット解析が完了したら、トラブルの解決方法、あるいは解決の糸口を提示して、ここで学んだことの概要を説明します。

パケット解析は非常に臨機応変な作業です。それぞれのシナリオの解析に用いた手法は、読者の手法とは異なるかもしれません。解析手法は人それぞれです。一番大切なのは、解析の結果トラブルが解決したか、その経験から学習できたかといったことなのです。また本章で説明するトラブルの大半は、パケットキャプチャツールを使わずに解決できるかもしれません。しかし、パケット解析の手法に初めて接したときに、パケット解析を用いて別の視点からありがちなトラブルに取り組むことが非常に役立つことに気づき、こうしたシナリオも提示しています。

8.1 パケットレベルでのSNSの分析

まずは2つの人気SNSサイト、TwitterとFacebookのトラフィックを見てみましょう。各サービスの認証処理を調べ、両者の非常によく似た機能が、異なる方法を使って同じ機能を実現している様子を見ていきます。またそれぞれのサービスの主要な機能の動作を見ていくことで、日々の活動で発生するトラフィックへの理解を深めていきましょう。

8.1.1 Twitterトラフィックのキャプチャ

twitter_login.pcap

技術コミュニティの最新ニュースを逐次入手するときから彼女の愚痴をこぼすときまで、Twitterはインターネット上でもっともよく利用されているサービスのひとつです。ファイルtwitter_login.pcapに、Twitterトラフィックのキャプチャがあります。



Webサイトのコードは頻繁に変更されます。そのため以降のセクションのキャプチャを再度作成しようとしても、ここに示したものと結果が異なるかもしれません。

8.1.1.1 Twitterのログインプロセス

パケット解析を指導するとき、学生に最初にやってもらうのが、普段使っているWebサイトへのログインと、そのログイン処理のトラフィックのキャプチャです。これにはふたつの目的があります。パケット一般について知ってもらうのと、平文のパスワードがネットワーク上をやり取りされていることを見つけてもらうことで、日々の作業がどれだけ危険かに気づかせるためです。

幸いにもTwitterの認証処理にはそれほど問題はありません。図8-1からわかるように、最初の3つのパケットは、ローカルのパソコン(172.16.16.128)❶とリモートサーバ(168.143.162.68)❷とのTCPハンドシェイクから構成されています。リモートサーバは443番ポートで接続を待ち受けています❸。これはSSL over HTTP、一般にHTTPSと称されるセキュアなデータ転送方法に使われるポートです。これだけを見ても、これがSSLトラフィックであると推測できます。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.16.128	168.143.162.68	TCP	4669 > 443 [SYN] Seq=4164864060 Win=8192 Len=0 MSS=1460
2	0.072728	168.143.162.68	172.16.16.128	TCP	443 > 4669 [SYN, ACK] Seq=1150193371 Ack=4164864061 Win=18200 Len=0 MSS=1406
3	0.000101	172.16.16.128	168.143.162.68	TCP	4669 > 443 [ACK] Seq=4164864061 Ack=1150193372 Win=16872 Len=0

図8-1 ハンドシェイクで443番ポートと接続

ハンドシェイクに続くパケットは、暗号化されたSSLハンドシェイクです。SSLは鍵、つまり二者間での通信を暗復号化するための文字列に依存しています。ハンドシェイク処理は機器間で鍵を授受する正規のやり取りであると同時に、さまざまなコネクションと暗号化に関する設定のネゴシエーションでもあります。ハンドシェイクが完了すると、セキュアなデータ転送が始まります。

データのやり取りを行っている暗号化されたパケットを見つけるには、[Packet Details] ペインの[Info] カラムがApplication Dataとなっているパケットを探してください。これらのパケットのSSL部分を展開すると、Encrypted Application Dataフィールドが表示され、図8-2のように暗号化されたデータ❶が格納されているはずで、これはログイン時のユーザー名とパスワードの転送を示しています。

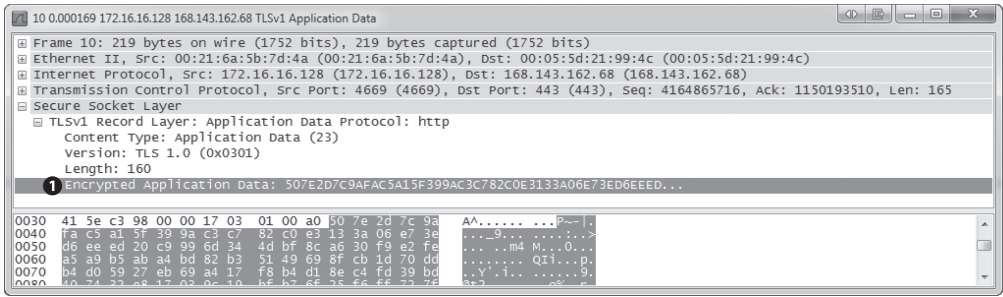


図8-2 暗号化された資格情報が転送されている

16番目のパケットでFIN/ACKによるコネクションのティアダウンドプロセスが開始されるまで認証が続きます。認証が終わると、ブラウザがTwitterのホームページへとリダイレクトされます。図8-3でわかるように、19番目、21番目、22番目のパケットは、同じリモートサーバ（168.143.162.68）ですが、443番ポートではなく80番ポートで新たなコネクションを開始するハンドシェイクプロセスです①。ハンドシェイクが完了すると、23番目のパケットでWebサーバのルートディレクトリ (/) を要求するHTTP GETリクエストが発行されます②。サーバは24番目のパケットでリクエストにACKを行い③、次のいくつかのパケットでのデータ転送を開始します。41番目のパケットのコンテンツはGETリクエストに関連するデータ転送の完了を示しています。

No.	Time	Source	Destination	Protocol	Info
19	0.000117	172.16.16.128	168.143.162.68	TCP	4670 > 80 [SYN] Seq=3871493748 win=8192 Len=0 MSS=1460
21	0.000063	168.143.162.68	172.16.16.128	TCP	80 > 4670 [SYN, ACK] Seq=2866679388 Ack=3871493749 win=18200 Len=0 MSS=1406
22	0.000063	172.16.16.128	168.143.162.68	TCP	4670 > 80 [ACK] Seq=3871493749 Ack=2866679389 win=16872 Len=0
23	0.000371	172.16.16.128	168.143.162.68	HTTP	GET / HTTP/1.1
24	0.080775	168.143.162.68	172.16.16.128	TCP	80 > 4670 [ACK] Seq=2866679389 Ack=3871495149 win=8400 Len=0

図8-3 認証完了後のTwitterホームページのルートディレクトリ (/) へのGETリクエスト

ホームページからリンクされた画像などのファイルを取得するため、さらに何回かのGETリクエストがキャプチャファイルに格納されています。

8.1.1.1 ツイートでのデータ送信 twitter_tweet.pcap

ログインしたら、今度は皆さんが何を考えているかを全世界に伝える番です。私は本を執筆中ですので、「This is a tweet for Practical Packet Analysis, second edition（これは実践パケット解析第2版のためのツイートです）」といった感じにツイートしてみます。このツイートを投稿する際のトラフィックをキャプチャしたのがファイルtwitter_tweet.pcapです。

このキャプチャファイルは、ツイートが送信された直後から始まっており、まずはローカルのコンピュータ172.16.16.134とリモートアドレス168.143.162.100間のハンドシェイクが行われています。4番目と5番目のパケットは、クライアントからサーバへ送られたHTTPパケットです。Wiresharkはこの2つのパケットのデータを組み合わせ、見やすいように5番目のパケットの [Packet Details] ペインにまとめています。

このHTTPヘッダを見るために、5番目のパケットの[Packet Details]ペインのHTTP部分を展開すると、図8-4のようにURL /status/updateに対してPOSTメソッドが使われていることがわかります①。Hostフィールドにtwitter.comの値が含まれているので②、これはツイートからのパケットだということになります。

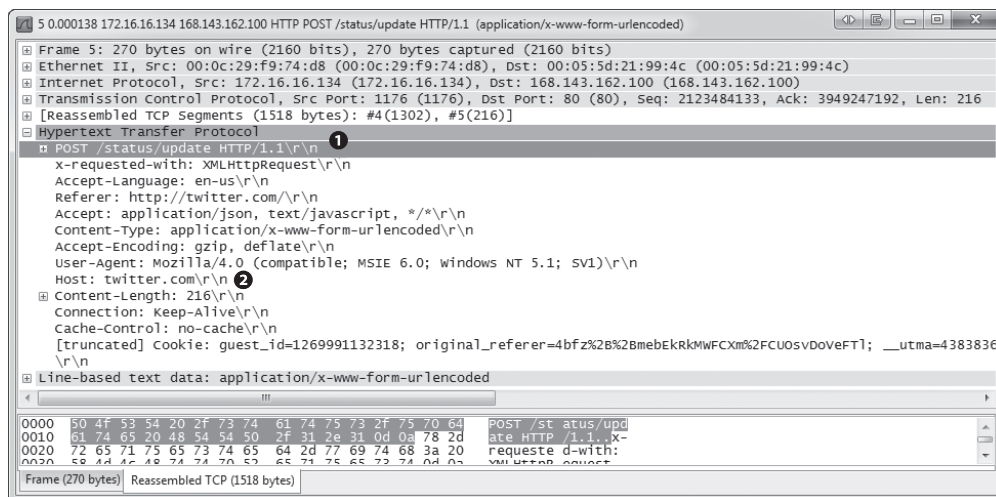


図8-4 Twitter更新のためのHTTP POST

図8-5のLine-based text dataフィールド①に情報が含まれていることに気づいたでしょう。このデータを解析すると、Authenticity Tokenというフィールドと、URLに次のような値を含んだstatusフィールドが確認できます。

```
This+is+a+tweet+for+practical+packet+analysis%2c+second+edition
```

statusフィールドの値は、暗号化されていない平文で送信したツイートです。

なかにはツイートを一部の人々にしか公開していない人もいますので、これには若干のセキュリティ面での不安があります。だからといって誰もがツイートを読めるわけではなく、このトラフィックを受信でき、ツイートの中身をすべて見ることができるのは、同じネットワーク上のユーザーだけです。

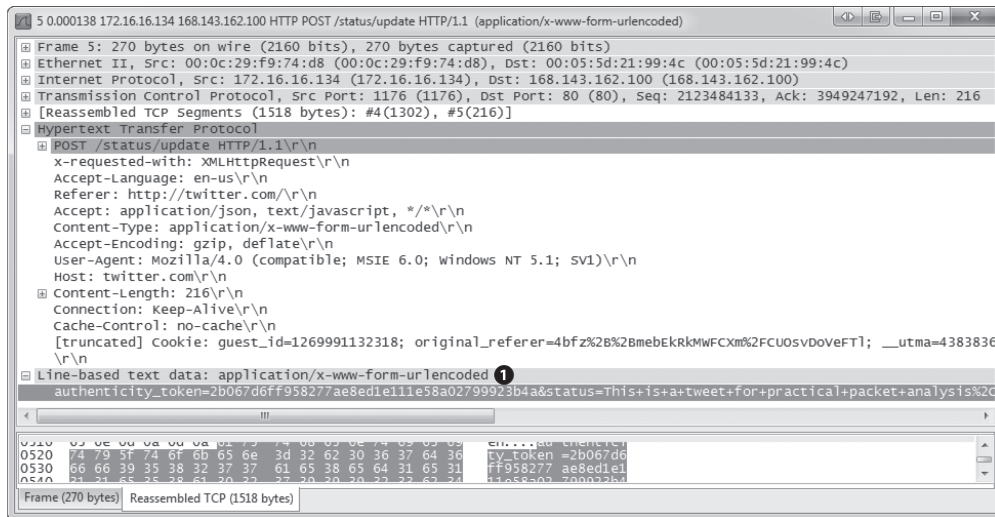


図8-5 平文のツイート

8.1.1.2 Twitterのダイレクトメッセージ

twitter_dm.pcap

今度はある種のセキュリティがかかったシナリオを考えてみましょう。Twitterのダイレクトメッセージは、ユーザーが多分に個人的なメッセージを共有できるものです。twitter_dm.pcapファイルは、Twitterのダイレクトメッセージのパケットキャプチャです。しかし図8-6からもわかるように、ダイレクトメッセージは完全にセキュアなものではありません。

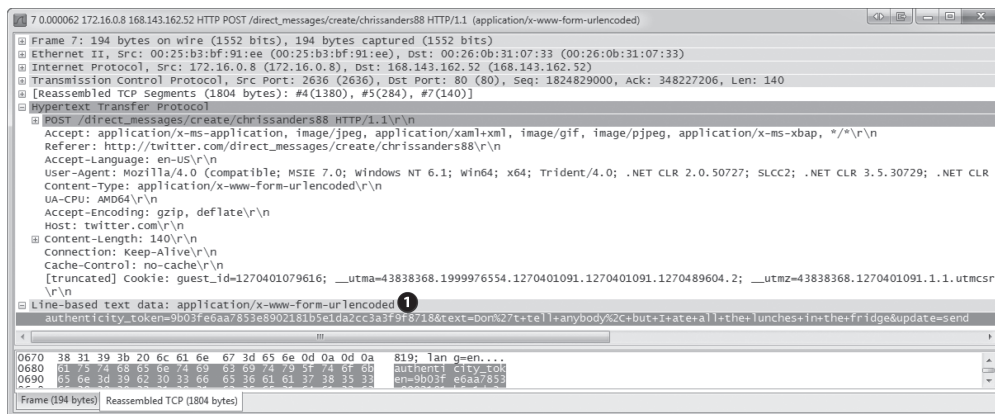


図8-6 ダイレクトメッセージ

図8-6の7番目のパケットは、コンテンツがやはり平文で送信されていることを示しています。先ほどのキャプチャファイルとLine-based text dataフィールドが同じであるのが何よりの証拠です。

ここでTwitterについて言えることは、必ずしも驚愕するようなことではありません。信頼できないネットワーク上では、プライベートなTwitterメッセージを用いて重要なデータを送信するのは考え直したほうがよい、ということです。

8.1.2 Facebookトラフィックをキャプチャする

ツイートの解読が終わったので、今度はFacebookにログインして、友達が夢中になってやっていることを眺めて、自分もやった気になってみたくなりました。Wiresharkを使ってFacebookのトラフィックをキャプチャして解析してみましょう。

8.1.2.1 Facebookのログイン処理

facebook_login.pcap

facebook_login.pcap ファイルにキャプチャしたログイン処理から始めましょう。キャプチャは、図8-7のように資格情報の送信から開始されています。Twitterのログイン処理と同様に、443番ポートでTCPハンドシェイクが行われ①、コンピュータ172.16.0.122②が、認証処理を行うサーバ69.63.180.173③と通信を開始します。ハンドシェイクが完了するとSSLハンドシェイクが行われ④、ログインのための資格情報が送信されます。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.0.122	69.63.180.173	TCP	54595 > 443 [SYN] Seq=2917405622 win=5840 len=0 MSS=1460 TSV=301989713 TSER=0 ws=6
2	0.089900	69.63.180.173	172.16.0.122	TCP	443 > 54595 [SYN, ACK] Seq=2894038304 ack=2917405623 win=1140 len=0 MSS=1380 ws=0 TSV=3479125768 TSER=301989713
3	0.000033	172.16.0.122	69.63.180.173	TCP	54595 > 443 [ACK] Seq=2917405623 Ack=2894038305 Win=92 Len=0 TSV=301989735 TSER=3479125768
4	0.000343	172.16.0.122	69.63.180.173	TLVSI	Client Hello ②
5	0.089522	69.63.180.173	172.16.0.122	TLVSI	Server Hello, Certificate, Server Hello Done
6	0.000031	172.16.0.122	69.63.180.173	TCP	54595 > 443 [ACK] Seq=2917405792 Ack=2894039242 Win=121 Len=0 TSV=301989758 TSER=3479125858
7	0.002848	172.16.0.122	69.63.180.173	TLVSI	Client key exchange, Change cipher spec, Encrypted Handshake Message
8	0.090444	69.63.180.173	172.16.0.122	TLVSI	Change cipher spec, Encrypted Handshake Message
9	0.000533	172.16.0.122	69.63.180.173	TLVSI	Application data
10	0.189619	69.63.180.173	172.16.0.122	TCP	443 > 54595 [ACK] Seq=2894039285 Ack=2917406956 Win=5473 Len=0 TSV=3479126142 TSER=301989781
11	0.073201	69.63.180.173	172.16.0.122	TLVSI	Application data

図8-7 ログインの資格情報がHTTPSによってセキュアに送信される

Twitterの認証処理との違いは、ログイン時の資格情報の送信後、認証に使われた接続のティアダウンがすぐに行われないことです。その代わり図8-8でフォーカスしているように、12番目のパケットのHTTPヘッダに/home.phpへのGETリクエストがあることが確認できます①。

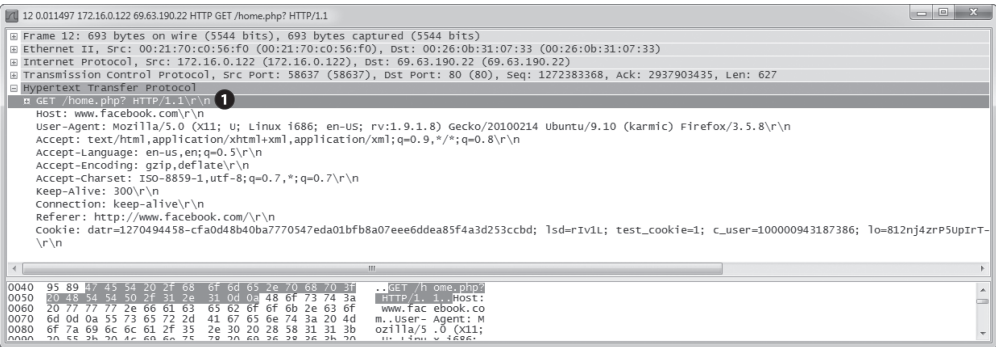


図8-8 認証の後、/home.phpへのGETリクエストが行われる

認証に使われたコネクションは、図8-9のキャプチャファイルの最後にある64番目のパケット①にあるように、home.phpのコンテンツが転送されるとティアダウンされます。まず80番ポートのHTTP接続がティアダウンされ（62番目のパケット）②、次に443番ポートのHTTPS接続がティアダウンされます。

No.	Time	Source	Destination	Protocol	Info
62	300.398161	172.16.0.122	69.63.190.22	TCP	58637 > 80 [FIN, ACK] Seq=1272384963 Ack=2937930926 win=1002 Len=0 TSV=302065222 TSER=3479159094
63	0.000388	172.16.0.122	69.63.180.173	TLSv1	Encrypted Alert
64	0.000027	172.16.0.122	69.63.180.173	TCP	54595 > 443 [FIN, ACK] Seq=2917406979 Ack=2894040467 win=158 Len=0 TSV=302065222 TSER=3479126214
65	0.036439	69.63.190.22	172.16.0.122	TCP	80 > 58637 [ACK] Seq=2937930926 Ack=1272384964 win=7233 Len=0 TSV=3479459532 TSER=302065222
66	0.000082	69.63.190.22	172.16.0.122	TCP	80 > 58637 [FIN, ACK] Seq=2937930926 Ack=1272384964 win=7233 Len=0 TSV=3479459532 TSER=302065222
67	0.000023	172.16.0.122	69.63.190.22	TCP	58637 > 80 [ACK] Seq=1272384964 Ack=2937930927 win=1002 Len=0 TSV=302065232 TSER=3479459532
68	0.052635	69.63.180.173	172.16.0.122	TCP	443 > 54595 [FIN, ACK] Seq=2894040466 Ack=2917406979 win=5496 Len=0 TSV=3479427810 TSER=302065222
69	0.000078	172.16.0.122	69.63.180.173	TCP	54595 > 443 [ACK] Seq=2917406980 Ack=2894040467 win=158 Len=0 TSV=302065245 TSER=3479427810
70	0.459231	172.16.0.122	69.63.180.173	TCP	54595 > 443 [FIN, ACK] Seq=2917406979 Ack=2894040467 win=158 Len=0 TSV=302065360 TSER=3479427810
71	0.088948	69.63.180.173	172.16.0.122	TCP	443 > 54595 [ACK] Seq=2894040467 Ack=2917406980 win=5496 Len=0 TSV=3479428358 TSER=302065360

図8-9 HTTP接続の次にHTTPS接続がティアダウンされる

8.1.2.2 Facebookのプライベートメッセージ facebook_message.pcap

ログイン認証処理に続いて、プライベートメッセージがどう処理されているかを見ていきます。facebook_message.pcap ファイルには、Facebookの筆者のアカウントから別のアカウントへ送られたメッセージをキャプチャしたパケットが含まれています。ファイルを開くとパケットの数が少ないことに驚くでしょう。

最初の2つのパケットは、メッセージそのものを送信するためのHTTPトラフィックです。2番目のパケットのHTTPヘッダを開くと、図8-10のように長めのURL文字列がPOSTメソッドで送信されています①。見てのとおり、文字列にはAJAXへの参照が含まれています。

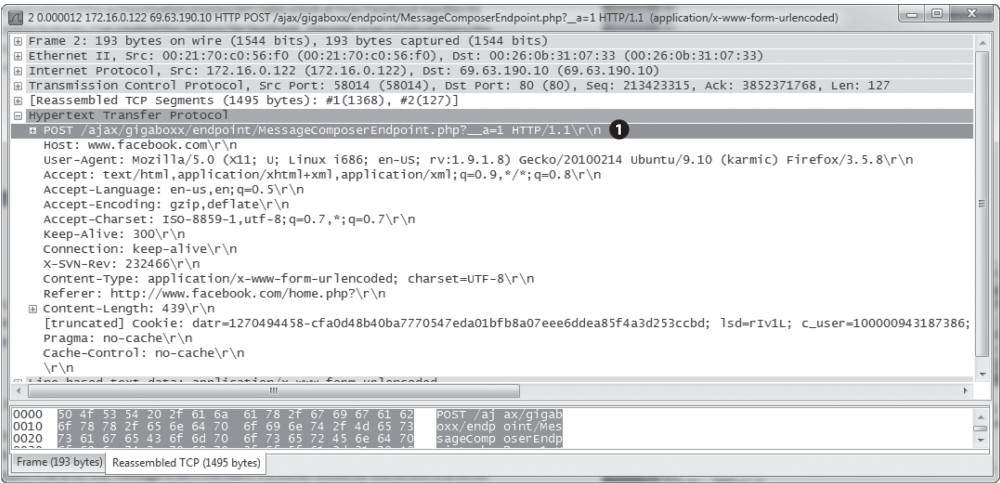


図8-10 AJAXを参照するHTTP POST

AJAX (Asynchronous JavaScript and XML) は、バックグラウンドでサーバから情報を取得する対話的なWebアプリケーションを構築するための、クライアントサイドの手法です。プライベートなメッ

セージがクライアントのブラウザから送信されれば、セッションが別のページへとリダイレクトされる (Twitterのダイレクトメッセージと同様に) と思うかもしれませんが、そうではありません。AJAXを使うというのは、個々のページからではなくある種の対話的なポップアップからメッセージが送信される、つまりコンテンツのリダイレクトやリロードが不要だということを意味します。これがAJAX実装のメリットのひとつなのです。

図8-11のように、2番目のパケットのLine-based text data部分を展開すれば、プライベートメッセージの内容を見ることができます。Twitter同様、Facebookのプライベートメッセージも暗号化されずに送信されているようです。

```
Line-based text data: application/x-www-form-urlencoded
[truncated] 'ids_c4bba389c9d7033d483222[0]=51800021&subject=Secret!&status=Don't%20fe11%20anybody%2c%20but%20i%20ate%20a11%20the
```

図8-11 Facebookのメッセージのコンテンツも平文になっている

8.1.3 TwitterとFacebookの比較

ここまでTwitterとFacebookという2つのWebサービスの認証とメッセージの機構が、それぞれ異なるアプローチを取っていることを見てきました。プログラマなら、Twitterの認証方法のほうが迅速かつ効率的だと主張するでしょう。一方セキュリティの研究家は、すべてのコンテンツが確実に送信されるFacebookの手法を評価するかもしれません。加えて、Facebookでは認証コネクションを終了する前に再度認証する必要がないため、**中間者攻撃** (man-in-the-middle-attack) がより難しくなります (中間者攻撃とは、悪意あるユーザーが二者間でのトラフィックを傍受して攻撃すること)。実際には、この2つの認証方法にはほとんど差がないのですが、2人のプログラマが同じ作業を行うルーチンを開発した場合、違いが起り得るということが実証されています。

この解析のポイントは、TwitterとFacebookがどう機能するかではなく、比較したり対比したりすることができるトラフィックをお見せすることです。この例は、似たようなサービスがうまく機能しない、あるいは遅い理由を調べるのに必要な枠組みを提供してくれるはずです。

8.2 ESPN.comのトラフィックをキャプチャする http_espn.pcap

朝のSNSのチェックが終わると、最新ニュースの見出しとスポーツの得点をチェックするのが習慣になっています。解析したら面白そうなサイトがあるのですが、そのうちのひとつがhttp://www.espn.com/です。http_espn.pcapにESPNサイトをブラウザしたトラフィックをキャプチャしました。

このキャプチャファイルには956個ものファイルが含まれています。それぞれのコネクションや異常を確認するために、手動でスクロールしてファイル全体を見ていくのはあまりにも大変なので、Wiresharkの解析機能を使って作業を簡単にしてみましょう。

8.2.1 Conversationsウィンドウの利用

ESPNのホームページにたくさんのリンクや機能が含まれているのを見れば、データ転送に約1000

個ものパケットが必要な理由も納得できます。大量のデータを転送する場合、データの取得元を知っていると便利ですが、さらに重要なのは、取得元が1つなのか複数なのかを把握しておくことです。Wiresharkの[Conversations]ダイアログ（メニューから[Statistics] → [Conversations]を選択）を利用すればこれがわかります。

図8-12の例では、14のIPによる対話、25のTCPコネクション、14のUDPによる対話が、メインのConversationsウィンドウに詳しく表示されています。たったひとつのサイトとしては、ずいぶん多くのことが行われています。

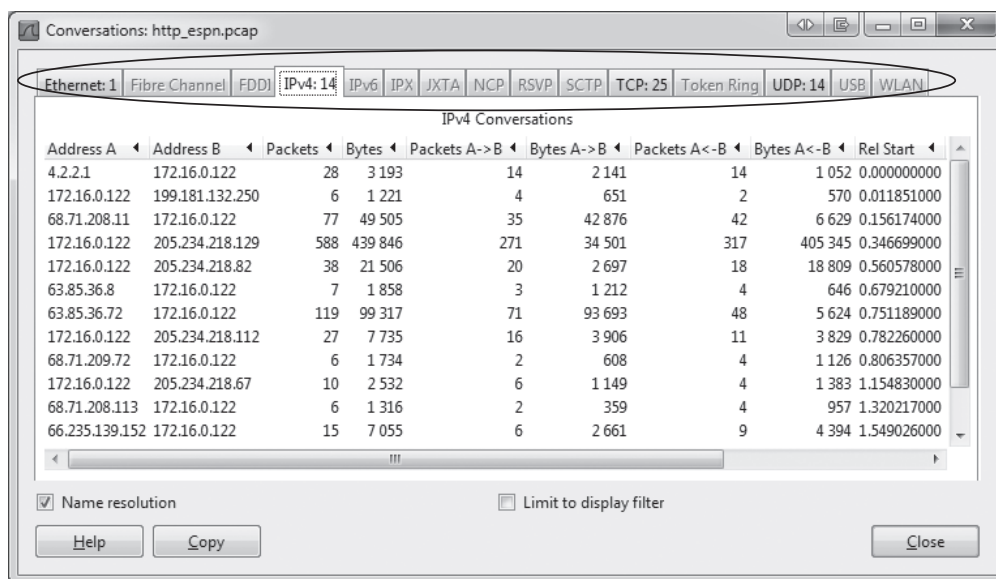


図8-12 複数のコネクションを表示している [Conversations] ダイアログ

8.2.2 [Protocol Hierarchy Statistics] ダイアログの利用

この状況をさらによく見るため、TCPコネクションとUDPで使われているアプリケーション層プロトコルを見てみましょう。メニューから[Statistics] → [Protocol Hierarchy]を選択し、図8-13の[Protocol Hierarchy Statistics]ダイアログを開きます。

Wireshark: Protocol Hierarchy Statistics

Display filter: none

Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	956	652181	2.548	0	0	0.000
Ethernet	100.00 %	956	652181	2.548	0	0	0.000
Internet Protocol	100.00 %	956	652181	2.548	0	0	0.000
User Datagram Protocol	2.93 %	28	3193	0.012	0	0	0.000
Domain Name Service	2.93 %	28	3193	0.012	28	3193	0.012
Transmission Control Protocol	97.07 %	928	648988	2.536	807	567803	2.219
Hypertext Transfer Protocol	12.66 %	121	81185	0.317	62	37579	0.147
Line-based text data	1.46 %	14	9761	0.038	14	9761	0.038
JPEG File Interchange Format	1.78 %	17	12156	0.047	17	12156	0.047
Portable Network Graphics	1.46 %	14	10587	0.041	14	10587	0.041
CompuServe GIF	0.94 %	9	7778	0.030	9	7778	0.030
Media Type	0.31 %	3	2332	0.009	3	2332	0.009
eXtensible Markup Language	0.21 %	2	992	0.004	2	992	0.004

図8-13 プロトコルを表示する [Protocol Hierarchy Statistics] ダイアログ

TCPアカウントはパケットの97.07パーセントを占め①、UDPアカウントは残りの2.93パーセント②となっています。予想どおりTCPトラフィックはHTTPであり③、HTTP経由で転送されるファイルタイプ別にさらに分類されています。

WiresharkではHTTPは12.66パーセントのみと表示されているので、TCPトラフィックはすべてHTTPだというと混乱するかもしれませんが、その他84.41パーセントは純粋なTCPトラフィックという意味なのです（データ転送と制御パケット）。UDP行の下のエントリを見ると、UDPトラフィックはすべてDNSです④。

この情報だけでも、いくつかの結論が導き出せます。ひとつはDNSトランザクションが非常に少ないということです。DNSパケットは28個あるので（図8-13の [Domain Name Service] エントリの隣の [Packets] カラムに並んでいます）、DNSトランザクションは最高でも14回という意味になります。リクエストとレスポンスで一組なので、パケットの総数を2で割ってこの数を出しています。[Conversations] ダイアログのUDP行の下を見ると、確かに14の対話が示されていて、われわれの推測が正しいことが証明されました。

DNSクエリは何らかの契機があって発生します。またキャプチャ内のそのほかのトラフィックはHTTPトラフィックのみです。このことから、ESPNサイト内のHTMLコードはDNS名でほかのドメインまたはサブドメインを参照しているため、複数のクエリが実行されていることが推察されます。

この論理を裏づける証拠を探してみましょう。

8.2.3 DNSトラフィックを参照する

フィルタを作成すれば、DNSトラフィックを見ることが簡単にできます。Wireshark ウィンドウのフィルタ部分に「dns」と入力すると、図8-14のように、すべてのDNSトラフィックが参照できます。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.0.122	4.2.2.1	DNS	Standard query A www.espn.com
2	0.011665	4.2.2.1	172.16.0.122	DNS	Standard query response A 199.181.132.250
9	0.144300	172.16.0.122	4.2.2.1	DNS	Standard query A espn.go.com
10	0.155758	4.2.2.1	172.16.0.122	DNS	Standard query response A 68.71.208.11
21	0.326066	172.16.0.122	4.2.2.1	DNS	Standard query A a.espncdn.com
22	0.337568	4.2.2.1	172.16.0.122	DNS	Standard query response CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net
224	0.542078	172.16.0.122	4.2.2.1	DNS	Standard query A a1.espncdn.com
225	0.549050	172.16.0.122	4.2.2.1	DNS	Standard query A a2.espncdn.com
226	0.553531	4.2.2.1	172.16.0.122	DNS	Standard query response CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net
227	0.560189	4.2.2.1	172.16.0.122	DNS	Standard query response CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net
389	0.650057	172.16.0.122	4.2.2.1	DNS	Standard query A www.masters.com
417	0.679056	4.2.2.1	172.16.0.122	DNS	Standard query response CNAME www.masters.com.edgesuite.net CNAME a1075.g.akamai.net
425	0.737456	172.16.0.122	4.2.2.1	DNS	Standard query A adsatt.espn.go.com
426	0.738032	172.16.0.122	4.2.2.1	DNS	Standard query A log.go.com
427	0.749732	4.2.2.1	172.16.0.122	DNS	Standard query response CNAME adimages.go.com.edgesuite.net CNAME a1412.g.akamai.net
429	0.758282	172.16.0.122	4.2.2.1	DNS	Standard query A assets.espn.go.com

図8-14 標準クエリとレスポンスのように見えるDNSトラフィック

図8-14のDNSトラフィックは、すべてクエリとレスポンスのように見えます。クエリされたDNS名をよく見るには、クエリのみを表示するようなフィルタを作成します。フィルタを作成するには、[Packet List] ペインでクエリのパケットを選択し、[Packet Details] ペインでパケットのDNSヘッダを展開します。次に [Flags : 0x0100 (標準クエリ)] フィールドを右クリックし、[Apply as Filter] にマウスカーソルを持っていき、[Selected] を選択します。

これでdns.flags == 0x0100というフィルタが有効となり、クエリだけが表示されるようになるので、解析しているレコードがずっと読みやすくなります。また図8-14からもわかるようにクエリは14個ありますが(各パケットがクエリを表している)、すべてのドメイン名はESPN関連のように見えますので、ホームページで表示されているコンテンツに関連しているように見受けられます。

8.2.4 HTTPリクエストを見る

最後にHTTPリクエストを調べ、これらのクエリの送信元を検証します。そのためには、メニューから [Statistics] → [HTTP] を選択して [Requests] を選択し、[Create Stat] をクリックします(これを実行する前に、先ほど作成したフィルタを消去するのを忘れずに)。

図8-15は [HTTP Requests] ダイアログを示しています。ここに表示されている14のコネクション(各行は同一ドメインへのコネクションを表しています)は、DNSクエリされたドメインを示しています。

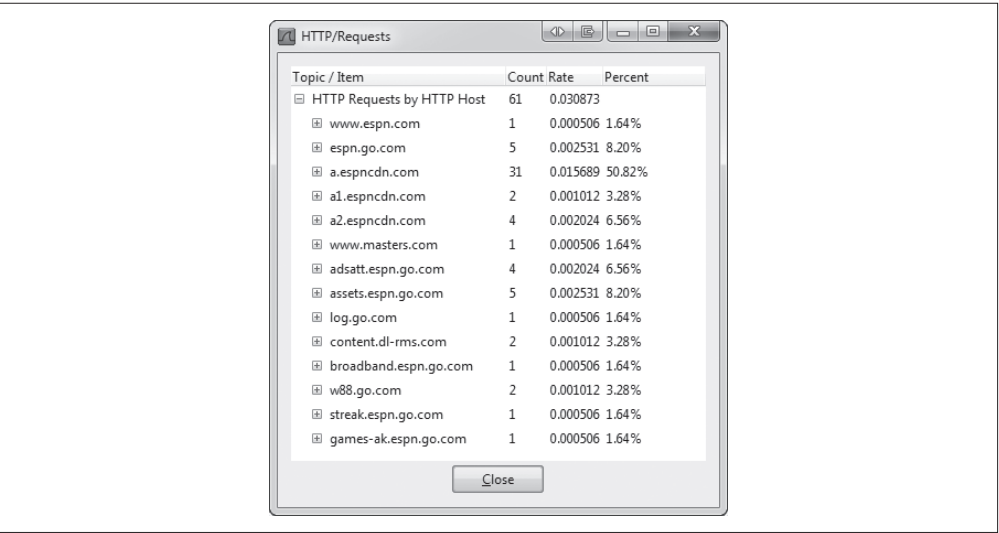


図8-15 アクセスされたドメインを表すHTTPリクエストの概要がウィンドウ内に表示されている

多くのコネクションが発生していますが、これらの一連の処理が一瞬で行われたのかどうか非常に興味のあるところです。これを確認する一番簡単な方法は、トラフィックの概要を見ることです。メニューから [Statistics] → [Summary] を選択しましょう。図8-16に示した [Summary] ダイアログを見ると一連の処理が約2秒で行われたことがわかります①。

Webページを見るという簡単なリクエストが14もの異なるドメインやサブドメインへのリクエストとなり、さまざまなサーバとやり取りを行い、それでもこの処理全体に2秒しかかかっていないというのは不思議な感じがします。

お気に入りのサイトへ訪問しながらトラフィックをキャプチャし、それをこのように解析していくのは練習としても面白いのではないのでしょうか。パケットを見ない限り、データが本当はどこから来ているのかはわからないのですから。

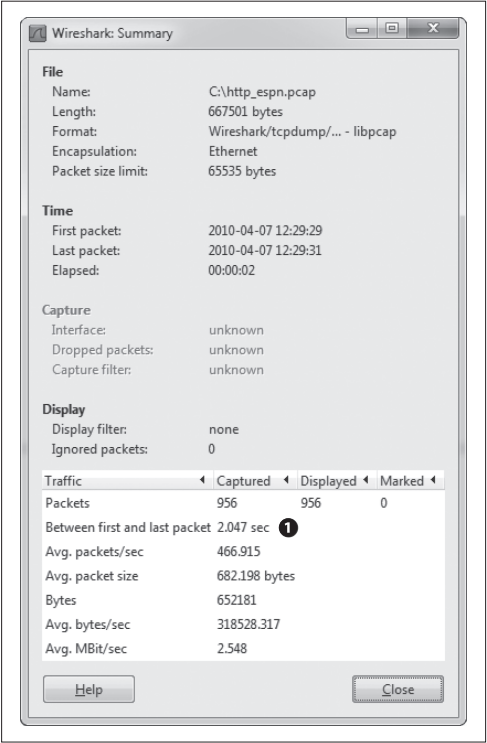


図8-16 [Summary] ダイアログから全体の処理が2秒以内であることがわかる

8.3 現場でのトラブル

今度は問題を含むトラフィックの例をいくつか見ていきましょう。多種多様なインターネットアクセスにまつわるトラブルと、不安定なプリンタや支社からの接続性の問題など、典型的な問題を見ていきましょう。

8.3.1 インターネットに接続できない：設定の問題

最初のトラブルのシナリオは、ユーザーがインターネットに接続できないというものです。他のコンピュータの共有や、ローカルサーバ上のアプリケーションへのアクセスを含む、イントラネット上のすべてのリソースにアクセスできることは確認済みです。

すべてのクライアントとサーバが単純なスイッチ経由で接続されており、ネットワークのアーキテクチャはかなり単純です。インターネットへのアクセスはデフォルトゲートウェイとして機能する単一のルータ経由で処理され、IPアドレス情報はDHCPによって提供されています。小規模なオフィスでは非常に一般的なシナリオです。

8.3.1.1 ケーブルへの潜入

`nowebaccess1.pcap`

トラブルの原因を判断するため、パケットキャプチャツールがネットワークを監視している最中にインターネットのブラウズを試行してもらいます。「2.5 パケットキャプチャツールを実際に設置する」に記載した情報(図2-15)により、パケットキャプチャツールを配置するのに適切な場所を判断します。

ネットワークのスイッチはポートミラーリングに対応していません。テストに協力してもらっている段階で、すでにユーザーの作業を邪魔してしまっているのに、再度オフラインになっても大丈夫でしょう(前述したとおり、タップを使うのがケーブルに潜入する最適の方法なので)。その結果取得したファイルが`nowebaccess1.pcap`です。

8.3.1.2 パケット解析

トラフィックのキャプチャは、図8-17のように、ARPリクエストとレスポンスで始まります。MACアドレス00:25:b3:bf:91:ee、IPアドレス172.16.0.8のユーザーのコンピュータは、1番目のパケットで、デフォルトゲートウェイである172.16.0.10のIPアドレスに対応付けられたMACアドレスを見つけるために、ネットワークセグメント上のすべてのコンピュータにARPブロードキャストパケットを送信します。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00:25:b3:bf:91:ee	ff:ff:ff:ff:ff:ff	ARP	who has 172.16.0.10? Tell 172.16.0.8
2	0.000090	00:24:81:a1:f6:79	00:25:b3:bf:91:ee	ARP	172.16.0.10 is at 00:24:81:a1:f6:79

図8-17 デフォルトゲートウェイに対するARPリクエストとレスポンス

2番目のパケット2でレスポンスを受け取ったユーザーのコンピュータは、172.16.0.10が00:24:81:a1:f6:79であることがわかったので、インターネットへと向かうゲートウェイへの経路が確立され

ました。

ARPレスポンスに続き、コンピュータは3番目のパケットで、DNSを使ってIPアドレスから名前解決を試みる必要があります。図8-18に示しているように、プライマリのDNSサーバである4.2.2.2にDNSクエリパケットを送信します❶。



図8-18 4.2.2.2に送信されたDNSクエリ

普通の状況だと、DNSサーバはDNSクエリに非常に迅速に応答しますが、ここではそうなっていません。応答ではなく、同じDNSクエリが再度別のアドレスへと送信されています。図8-19では、4番目のパケットで2回目のDNSクエリが、セカンダリのDNSサーバ4.2.2.1に送信されています❷。

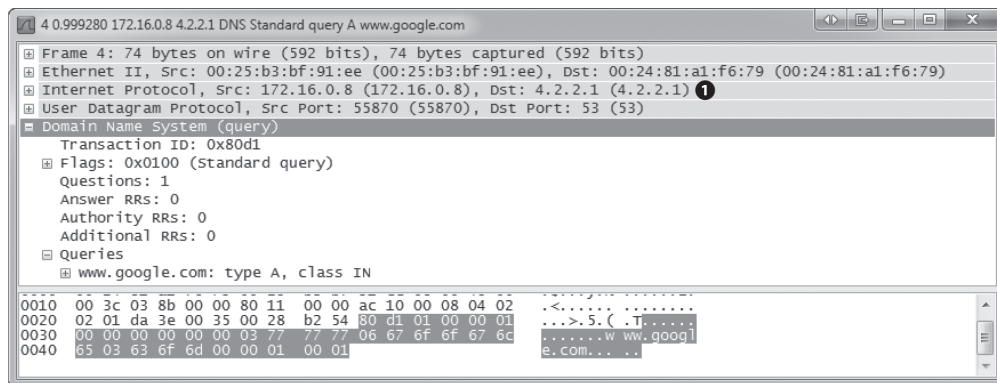


図8-19 4.2.2.1に送信された2回目のDNSクエリ

またもやDNSサーバからの応答はなく、クエリは1秒後に再度、4.2.2.2へと送信されます。このプロセスはからわかるように、プライマリのDNSサーバ❶とセカンダリ❷へと交互にパケットを送るという形で、数秒間繰り返されます。この処理全体にかかる時間は約8秒です❸。これはユーザーのブラウザが、Webサイトに接続できないと報告するまでに要する時間となります。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00:25:b3:bf:91:ee	ff:ff:ff:ff:ff:ff	ARP	who has 172.16.0.10? Tell 172.16.0.8
2	0.000090	00:24:81:a1:f6:79	00:25:b3:bf:91:ee	ARP	172.16.0.10 is at 00:24:81:a1:f6:79
3	0.000105	172.16.0.8	4.2.2.2 ①	DNS	Standard query A www.google.com
4	0.999280	172.16.0.8	4.2.2.1 ②	DNS	Standard query A www.google.com
5	1.999279	172.16.0.8	4.2.2.2	DNS	Standard query A www.google.com
6	3.999372	172.16.0.8	4.2.2.1	DNS	Standard query A www.google.com
7	3.999393	172.16.0.8	4.2.2.2	DNS	Standard query A www.google.com
8	7.999627	172.16.0.8	4.2.2.1	DNS	Standard query A www.google.com
③ 9	7.999648	172.16.0.8	4.2.2.2	DNS	Standard query A www.google.com

図8-20 通信が中断されるまでDNSクエリが繰り返される

パケットからトラブルの原因を特定してみましょう。まず、デフォルトゲートウェイだと考えているルータへのARPリクエストが成功していることから、この通信機器は動作しており、通信可能だとわかります。またユーザーのコンピュータは実際にパケットを転送しているので、コンピュータ自体にプロトコルスタックの問題はないと推定できます。問題は明らかに、DNSリクエストが行われた時点で起きているのです。

ここでは、DNSクエリがインターネット上の外部サーバ（4.2.2.2または4.2.2.1）によって解決されます。つまり名前解決が適切に行われるには、インターネットにパケットをルーティングするルータがDNSクエリをDNSサーバへきちんと転送し、サーバがこれに応答する必要があるのです。これらのことはすべて、WebページのリクエストにHTTPが利用される前に行われます。

インターネットに接続できないユーザーがほかには存在しないので、ルータと外部のDNSサーバはおそらく問題の原因ではないと言えます。すると唯一疑わしいのが、ユーザーのコンピュータです。

問題のコンピュータを詳しく調べてみると、DHCPが割り当てたアドレス情報を受け取る設定になっておらず、手動でアドレス情報が設定されていたため、デフォルトゲートウェイのアドレス設定が適切でなかったことが判明しました。デフォルトゲートウェイとして設定されたアドレスが不適切だったため、DNSクエリパケットを転送できなかったのです。

8.3.1.3 ここで学んだこと

このシナリオの問題は、クライアントの設定ミスにありました。問題自体はかなり単純なものでしたが、ユーザーには多大な影響を与えるものでした。このような単純な設定ミスのトラブルシューティングでも、ネットワーク知識のない人や、ちょっとしたパケット解析を行う能力がなければ、かなりの時間を要してしまいます。パケット解析は大規模で複雑なトラブルを解決するときだけのものではないのです。

このシナリオでは、デフォルトゲートウェイのIPアドレスがわかっていなかったので、Wiresharkだけではトラブルの原因を確定できませんでしたが、どこを確認すべきかを教えてくれたため、貴重な時間を節約できました。デフォルトゲートウェイを調べたり、ISPに連絡を取ったり、外部のDNSサーバのトラブルシューティングを行う方策を探したりすることなく、トラブルの原因であるコンピュータそのものに集中することができたのです。



ネットワークのIPアドレス体系を知っていれば、解析がさらに早く行えました。ARPリクエストが送信されたIPアドレスがデフォルトゲートウェイのアドレスと違うと気づけば、このトラブルの原因はすぐに判明したのです。こうした単純な設定ミスはしばしばネットワークトラブルの根源となりますが、パケット解析を少し行うだけで通常は簡単に解決できます。

8.3.2 インターネットに接続できない：不適切なリダイレクト

このシナリオでも、コンピュータからインターネットに接続できないユーザーに再度登場してもらいましょう。しかしながら先ほどとは異なり、今度のユーザーはインターネットには接続できるのですが、自分のホームページである `http://www.google.com/` に接続できません。Googleが提供するドメインにアクセスしようとする、「Internet ExplorerはWebページを表示できません」というページにリダイレクトされてしまいます。しかもトラブルが起きているのはこのユーザーだけです。

先ほどのシナリオ同様、いくつかの簡単なスイッチと、デフォルトゲートウェイとして機能するルータ1台のみで構成された小規模なネットワークです。

8.3.2.1 ケーブルへの潜入

nowebaccess2.pcap

解析を始めるため、タップを使ってユーザーに `http://www.google.com/` をブラウズしてもらった際に生成されるトラフィックをキャプチャしました。その結果のファイルが `nowebaccess2.pcap` です。

8.3.2.2 パケット解析

トラフィックのキャプチャは、図8-21のように、ARPリクエストとレスポンスで始まります。MACアドレス `00:25:b3:bf:91:ee`、IPアドレス `172.16.0.8` のユーザーのコンピュータは、1番目のパケットで、`172.16.0.102` のIPアドレスに対応付けられたMACアドレスを見つけるために、ネットワークセグメント上のすべてのコンピュータにARPブロードキャストパケットを送信します。この時点で、このIPアドレスの役割は不明です。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00:25:b3:bf:91:ee	ff:ff:ff:ff:ff:ff	ARP	who has 172.16.0.102? Tell 172.16.0.8
2	0.000334	00:21:70:c0:56:f0	00:25:b3:bf:91:ee	ARP	172.16.0.102 is at 00:21:70:c0:56:f0

図8-21 ネットワーク上の別の機器へのARPリクエストとレスポンス

2番目のパケットで、IPアドレス `172.16.0.102` が `00:21:70:c0:56:f0` であるとわかりました。先ほどのシナリオ同様、これはデフォルトゲートウェイのIPアドレスだとして、DNSパケットはこのアドレスを使って外部DNSサーバに転送されるものと推定してみましょう。ところが図8-22を見ると、次のパケットはDNSリクエストではなく、`172.16.0.8` から `172.16.0.102` へのTCPパケットです。SYNフラグがセットされているということは、これは2つの機器間で新たなTCPコネクションを確立するためのハンドシェイクの、最初のパケットだということになります❶。

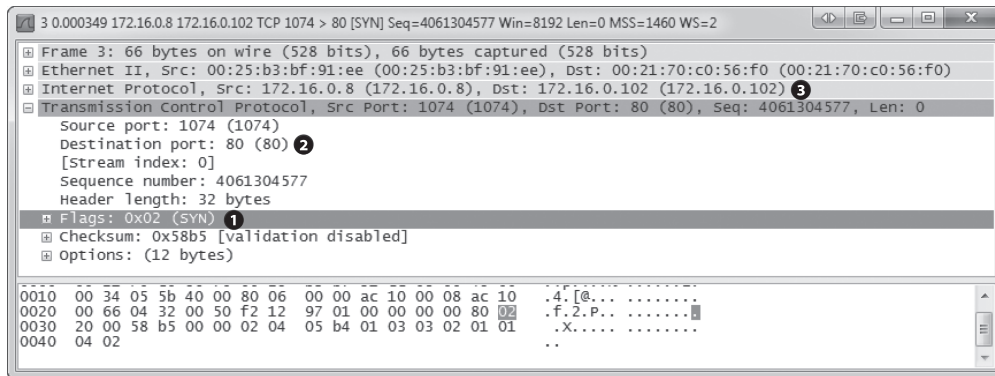


図8-22 内部の機器から別の内部の機器へと送られたTCP SYNパケット

注目すべき点として、通常HTTPトラフィックに使われる80番ポートで**2**、172.16.0.102**3**に対するTCPコネクションの確立が試行されています。このコネクションの試行は、172.16.0.102がRSTとACKフラグをセットしたTCPパケット（4番目のパケット）**1**をレスポンスとして送信した時点で中断してしまいます（図8-23）。

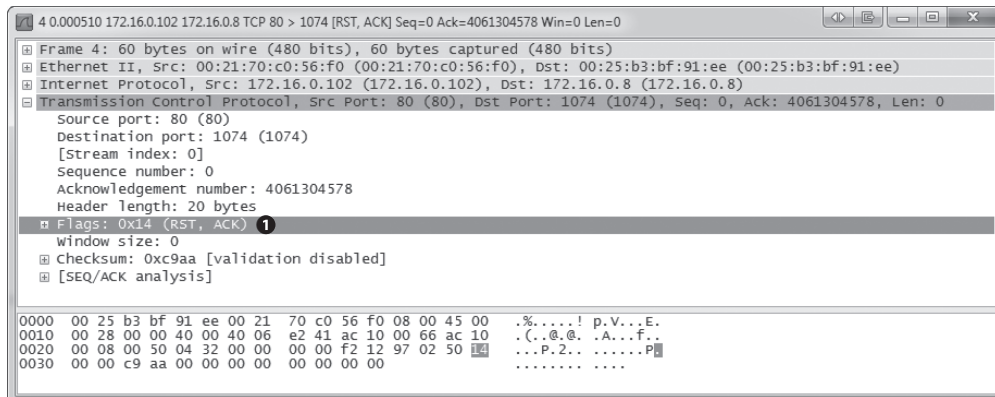


図8-23 TCP SYNへの応答として送られたTCP RSTパケット

6章で、RSTフラグをセットしたパケットはTCPコネクションの終了に用いられるという説明があったことを思い出しましょう。しかしながら今回のシナリオでは、172.16.0.8のコンピュータが、172.16.0.102の機器と80番ポートでTCPコネクションを確立しようとしたところ、この機器では、80番ポートでリクエストを待ち受けるサービスが設定されていなかったため、TCP RSTパケットが送られてコネクションが終了してしまっています。図8-24のように、通信が完全に終了するまでに、ユーザーのコンピュータからSYNが送られ、RSTがセットされたレスポンスが戻ってくるという処理はさらに2回繰り返されています。そのあとに、ユーザーのブラウザに「ページが表示できません」という

内容のメッセージが表示されます。

No.	Time	Source	Destination	Protocol	Info
3	0.000349	172.16.0.8	172.16.0.102	TCP	1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460 W5=2
4	0.000510	172.16.0.102	172.16.0.8	TCP	80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0
5	0.499162	172.16.0.8	172.16.0.102	TCP	1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460 W5=2
6	0.499362	172.16.0.102	172.16.0.8	TCP	80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0
7	0.999190	172.16.0.8	172.16.0.102	TCP	1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460
8	0.999507	172.16.0.102	172.16.0.8	TCP	80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0

図8-24 TCP SYNとRSTパケットは合計3回送られている

きちんと動作している別のネットワーク機器の設定を調べると、1番目と2番目のパケットのARPリクエストとレスポンスに問題がある可能性が浮かび上がってきました。ARPリクエストがデフォルトゲートウェイのMACアドレスではなく、別の不明な機器に送られていたのです。また、ARPリクエストとレスポンスを追跡すれば、www.google.comに対応付けられたIPアドレスを見つけるために、DNSサーバに送信されるDNSクエリが見られると思ったのですが、見つかりません。DNSクエリの生成が行われない理由は2つあります。

- コネクションを確立しようとした機器のDNS名とIPアドレスがすでにDNSキャッシュに格納されている。
- DNS名に対するコネクションを確立しようとした機器が、DNS名とIPアドレスの対応付けを、hostsファイルに保持している。

クライアントのコンピュータをさらに調べてみると、コンピュータのhostsファイルにwww.google.comのエントリがあり、これが内部IPアドレス172.16.0.102に対応付けられていました。この間違ったエントリがユーザーのトラブルを引き起こしていたのです。

一般的に、コンピュータはDNS名とIPアドレスの対応付けに関する情報源としてhostsファイルを利用し、外部に問い合わせる前にそのファイルを確認します。今回のシナリオでは、ユーザーのコンピュータがhostsファイルを確認してwww.google.comのエントリを見つけ、www.google.comはローカルのネットワーク上にあると判断しました。そのためARPリクエストを該当の機器に送ってレスポンスを受け取り、172.16.0.102と80番ポートでTCPコネクションを確立しようとしたのです。しかしこの機器がWebサーバとして設定されていなかったため、コネクションが確立できなかったわけです。

hostsファイルのエントリを削除することで、ユーザーのコンピュータは正しく通信を開始し、www.google.comに接続できるようになりました。



Windowsシステムでhostsファイルを確認するには、C:\¥Windows¥System32¥drivers¥etc¥hostsを参照してください。Linuxの場合は/etc/hostsを参照してください。

このシナリオは非常によく起こります。これは、マルウェアが何年もの間、ユーザーを悪意あるコードが埋め込まれたWebサイトにリダイレクトするために用いてきた方法でもあります。攻撃者がhosts

ファイルを改変し、オンラインバンキングへアクセスしようとするたびに、口座情報を盗み出す偽サイトへリダイレクトされていたとしたらどうでしょう！

8.3.2.3 学んだこと

トラフィックの解析を続けていくと、さまざまなプロトコルの動作と、その遮断の仕方の両方を学んでいきます。今回のシナリオでは、外的な制約や設定ミスではなく、クライアントの設定ミスのために、DNSクエリが送信されませんでした。

パケットレベルでこのトラブルを調べると、不明なIPアドレスと、通信の鍵となるDNSが見当たらないことがすぐにわかります。この情報によって、クライアントが問題の原因であると判明したのです。

8.3.3 インターネットに接続できない：上位の問題

先の2つのシナリオ同様、今回のシナリオでも、ユーザーがコンピュータからインターネットへ接続できないと文句を言っています。このユーザーは問題の原因をひとつのWebサイト、`http://www.google.com`へ絞り込みました。さらに調べてみると、この問題が組織全体に影響していることが判明しました。誰もGoogleドメインへアクセスできないのです。

先の2つのシナリオと同じように、このネットワークもいくつかのスイッチと、インターネットへとつなぐ1台のルータで構成されています。

8.3.3.1 ケーブルへの潜入

トラブルシューティングのために、まず`http://www.google.com`をブラウザしてトラフィックを生成します。トラブルはネットワーク全体に及んでいる、つまり自分のコンピュータも影響を受けており、大規模なマルウェア感染の可能性もあるため、自分のコンピュータから直接キャプチャすべきではありません。このような状況に置かれた場合、タップが最良の選択となります。サービスが少しの間中断されるだけで、あとは何もする必要がないからです。タップによるキャプチャのファイルは`nowebaccess3.pcap`になります。

8.3.3.2 パケット解析

このパケットキャプチャは、ARPトラフィックではなくDNSトラフィックで始まります。最初のパケットは外部アドレス向けで、2番目のパケットにはそのアドレスからのレスポンスとなっているので、ARP処理はすでに行われており、デフォルトゲートウェイのMACとIPアドレスとの対応付けはすでに172.16.0.8の機器のARPキャッシュに存在していると仮定できます。

図8-25に示したのが、コンピュータ172.16.0.8からアドレス4.2.2.1へ最初に送られたパケットのキャプチャ❶で、これはDNSパケットです❷。パケットの内容を調べると、`www.google.com`のAレコードのクエリであることがわかります❸。

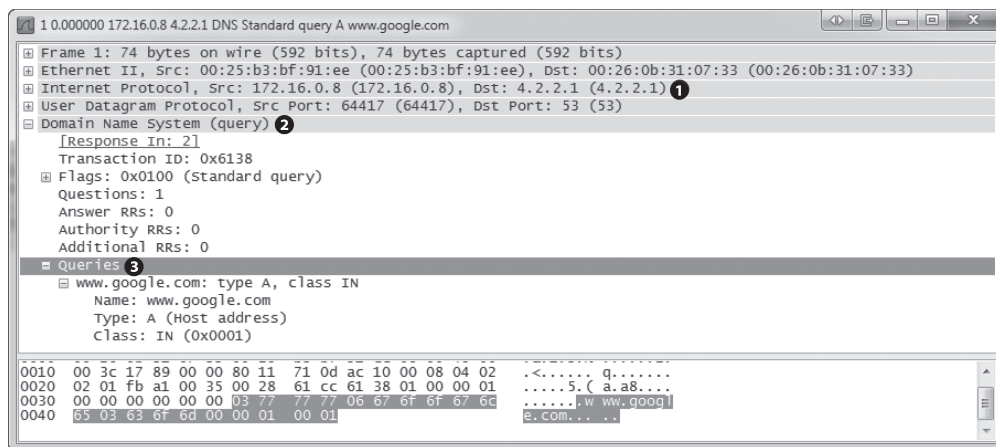


図8-25 www.google.comのAレコードのDNSクエリ

4.2.2.1からのクエリへのレスポンスが、図8-26の2番目のパケットです。[Packet Details] ペインを確認すると、このリクエストに対するレスポンスを返却したネームサーバが、クエリに対して複数の回答を提供していることがわかります①。ここまではすべて良好で、通信も滞りなく行われています。

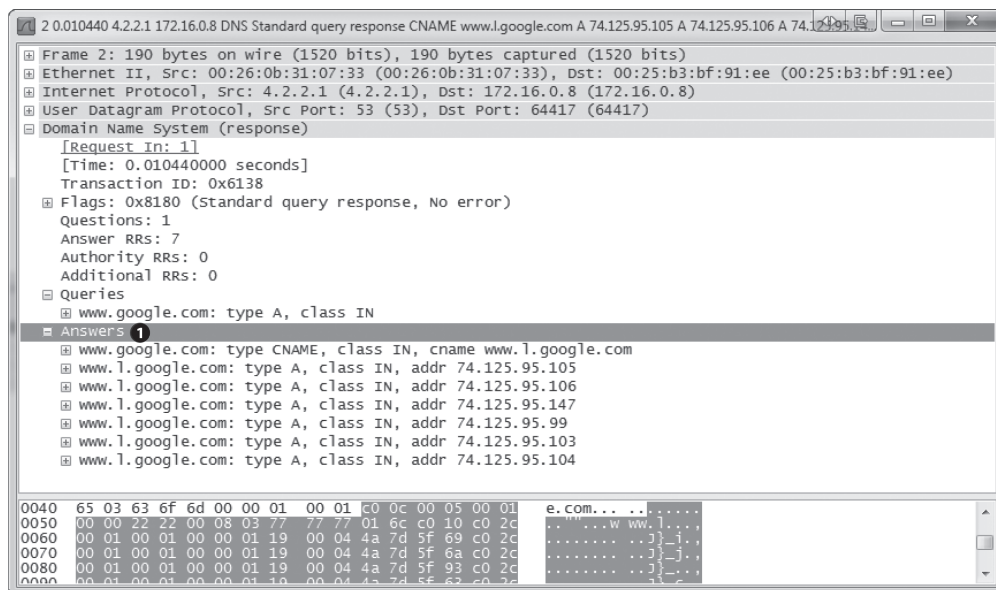


図8-26 複数のAレコードが含まれたDNSレスポンス

ユーザーのコンピュータがWebサーバのIPアドレスを確認したので、Webサーバとの通信の試行が可能になりました。図8-27のように、このプロセスは172.16.0.8から74.125.95.105へ送られた3番目のパケット、TCPパケットで始まります❶。宛先のアドレスは、2番目のパケット、DNSクエリに対するレスポンスで提供されている、Aレコード群のうち先頭のアドレスとなっています。TCPパケットにはSYNフラグがセットされており❷、リモートのWebサーバと80番ポートでの通信を試行しています❸。

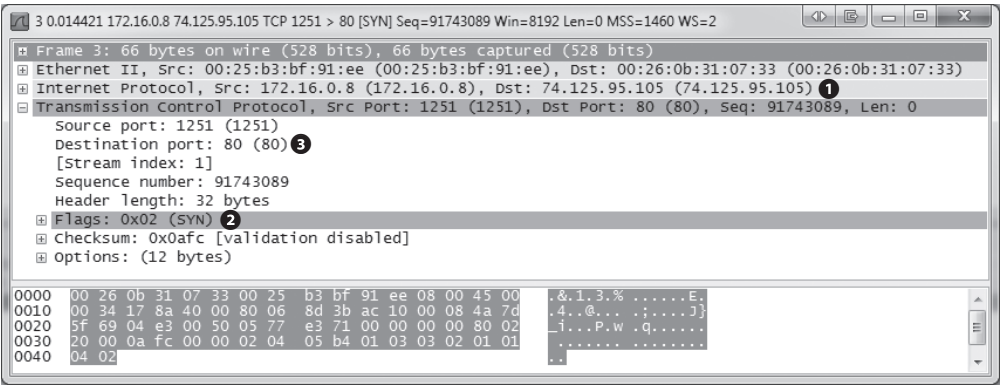


図8-27 80番ポートでのコネクション開始を試行するSYNパケット

これはTCPハンドシェイクプロセスなので、応答としてTCP SYN/ACKパケットが戻されるはずですが、少しすると別のSYNパケットが、送信元から宛先へと送られていました。この処理はさらに約1秒後にも発生しており、ここで通信が中断され、ブラウザはWebサイトが見つからないと報告していました(図8-28)。

No.	Time	Source	Destination	Protocol	Info
3	0.014421	172.16.0.8	74.125.95.105	TCP	1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2
4	0.019417	172.16.0.8	74.125.95.105	TCP	1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2
5	1.016531	172.16.0.8	74.125.95.105	TCP	1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2

図8-28 レスポンスがないためTCP SYNパケットが3度送信されている

4.2.2.1の外部DNSサーバへのDNSクエリが成功していることから、ネットワーク内のコンピュータが外部に接続できるのはわかっています。DNSサーバからのレスポンスに含まれるIPアドレスにも問題はなく、コンピュータはそのうちのアドレスのひとつへの接続を試行しています。接続に用いているローカルのコンピュータもきちんと動作しているようです。

問題は、リモートのサーバがコネクションのリクエストに応答しない、つまりTCP RSTパケットが返信されないことです。これにはいくつかの理由が考えられます。Webサーバの設定ミス、Webサーバのプロトコルスタックの機能不全、あるいはリモートのネットワーク上のパケットフィルタ(ファイアウォール)などです。ローカルでパケットフィルタは行われていないと仮定すると、考えられる解決策はリモートのネットワーク側にあるということになり、お手上げです。ここではWebサーバが正し

く機能しておらず、接続の試みもまったく成功しませんでした。Google側で問題が解決されれば、通信が可能になります。

8.3.3.3 学んだこと

今回のシナリオのトラブルは、こちら側では対応できないものでした。パケット解析の結果、トラブルはローカルのネットワーク上の機器でも、ルータでも、名前解決サービスを提供する外部DNSサーバでもないことが判明したためです。問題の根源は、われわれが管理するネットワーク外にありました。

トラブルの原因が自分たちになんかあることがわかると、ストレスが軽減されるだけでなく、管理部門に文句を言われたときの体面も救われます。自分たちのせいじゃないと主張するISPやベンダー、ソフトウェア会社と何度もケンカしてきましたが、このとおり、パケットは嘘をつきません。

8.3.4 不安定なプリンタ

ITヘルプデスク管理が印刷トラブルにてこずっています。営業部門のユーザーは、営業で使っている大型プリンタがトラブっていると報告しています。大量の印刷ジョブを送ると、数ページ印刷しただけで停止してしまうとのこと。ドライバ設定を何度も変えてみましたが、うまくいかなかったようです。ヘルプデスクのスタッフは、これがネットワークの問題ではないかどうかを確認してほしいと言っています。

8.3.4.1 ケーブルへの潜入

inconsistent_printer.pcap

これはプリンタのトラブルなので、できる限りプリンタに近いところにパケットキャプチャツールを設置することから始めます。プリンタ本体にはWiresharkをインストールできませんが、ネットワークで使用されているのは最新型のL3スイッチなので、ポートミラーリングが使えます。プリンタが接続されているポートを空いているポートへミラーし、Wiresharkがインストールされたコンピュータをこのポートへつなぎます。設定が完了したら、プリンタに大量の印刷ジョブを送ってもらい、出力を監視します。それをキャプチャしたファイルがinconsistent_printer.pcapです。

8.3.4.2 パケット解析

図8-29からわかるように、印刷ジョブを送っているコンピュータ(172.16.0.8)とプリンタ(172.16.0.253)間のTCPハンドシェイクがキャプチャファイルの先頭で行われています。ハンドシェイクに続き、1460バイトのTCPパケットがプリンタに送られます❶。データ量は、[Packet List] ペインの [Info] カラムの一番右端か、[Packet Details] ペインのTCPヘッダ情報の一番下で確認できます。

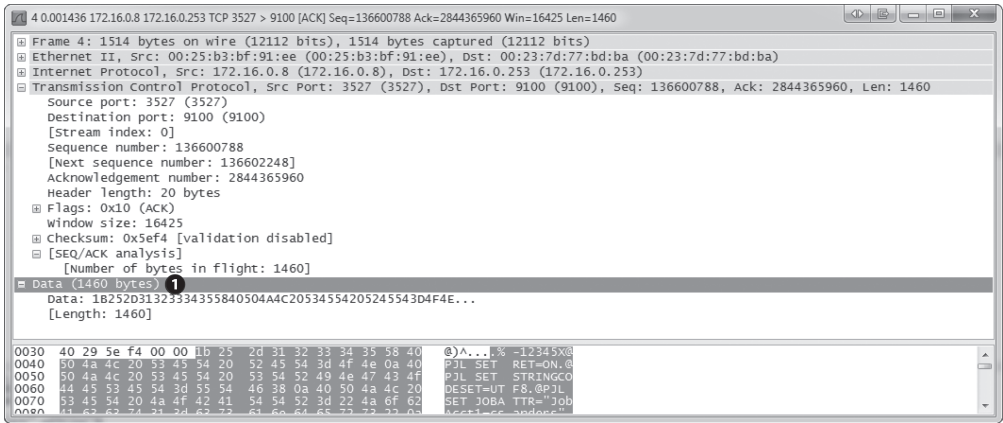


図8-29 TCPでプリンタに転送されたデータ

図8-30のように、4番目のパケットに続き、1460バイトのデータを含んだパケットがもうひとつ送られ、プリンタがACKを返却しています。

No.	Time	Source	Destination	Protocol	Info
3	0.000035	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136600788 Ack=2844365960 win=16425 Len=0
4	0.001436	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136600788 Ack=2844365960 win=16425 Len=1460
5	0.000009	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136602248 Ack=2844365960 win=16425 Len=1460
6	0.003847	172.16.0.253	172.16.0.8	TCP	9100 > 3527 [PSH, ACK] Seq=2844365960 Ack=136603708 win=7888 Len=106
7	0.000068	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136603708 Ack=2844366066 win=16398 Len=1460
8	0.000010	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136605168 Ack=2844366066 win=16398 Len=1460
9	0.000007	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136606628 Ack=2844366066 win=16398 Len=1460
10	0.000007	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136608088 Ack=2844366066 win=16398 Len=1460
11	0.027984	172.16.0.253	172.16.0.8	TCP	9100 > 3527 [ACK] Seq=2844366066 Ack=136609548 win=6144 Len=0
12	0.000057	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136609548 Ack=2844366066 win=16398 Len=1460
13	0.000014	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136611008 Ack=2844366066 win=16398 Len=1460
14	0.000009	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136612468 Ack=2844366066 win=16398 Len=1460
15	0.000009	172.16.0.8	172.16.0.253	TCP	3527 > 9100 [ACK] Seq=136613928 Ack=2844366066 win=16398 Len=1460
16	0.064656	172.16.0.253	172.16.0.8	TCP	9100 > 3527 [ACK] Seq=2844366066 Ack=136615388 win=4400 Len=0

図8-30 正常なデータ転送とTCPのACK

キャプチャファイルの最後の2つのパケットまで、データのやり取りが続きます。121番目のパケットはTCP再送 (TCP Retransmission) パケットで、図8-31が示すように、これがトラブルの最初の兆候です。

TCP再送パケットは、ある通信機器が別の機器へとTCPパケットを送信したのに、その機器がACKを返却しない場合に送られます。再送のしきい値に達すると、送信元は宛先の機器がデータを受信していないと判断し、パケットを再送します。この処理は通信を中断させるまで数回繰り返されます。

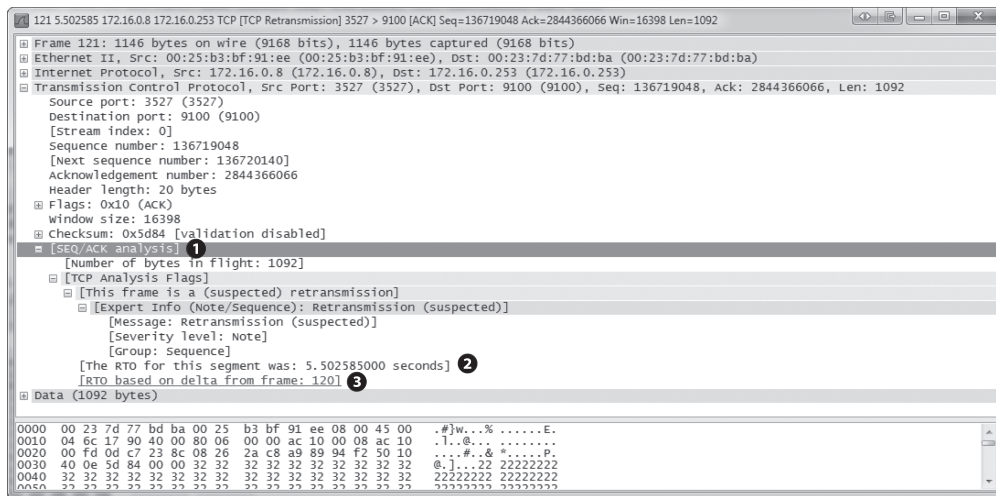


図8-31 TCP再送パケットはトラブルの兆し

今回のシナリオでは、プリンタが送信されたデータに対するACKを返信しなかったために、クライアントのコンピュータがプリンタにTCP再送パケットを送っています。TCPヘッダの[SEQ/ACK analysis]部分を展開し、表示される情報を見れば(図8-31)❶、これがなぜ再送だと判断されたかの詳細がわかります。121番目のパケットは120番目のパケットの再送です❷。また再送パケットの再送タイムアウト(RTO)は約5.5秒です❸。

パケット間の遅延を解析するとき、状況に応じて時刻表示形式を変更することができます。ここでは前のパケットが送信されたのち、どのくらい経ってから再送されたかを見たいので、メニューから [View] → [Time Display Format] を選択し、[Second Since Previous Captured Packet] を選択します。すると元々のパケット（120 番目のパケット）が送信されてから約 5.5 秒後に 121 番目のパケットが再送されていることがはっきりわかります（図 8-32）①。

No.	Time	Source	Destination	Protocol	Info
121	5,502585	172.16.0.8	172.16.0.253	TCP	[TCP Retransmission] 3527 > 9100 [ACK] Seq=136719048 Ack=2844366066 win=16398 Len=1092
122	5,600089	172.16.0.8	172.16.0.253	TCP	[TCP Retransmission] 3527 > 9100 [ACK] Seq=136719048 Ack=2844366066 win=16398 Len=1092

図8-32 パケットの送信間隔を把握することがトラブルシューティングに役立つ

その次のパケットも 120 番目のパケットの再送です。このパケットの RTO は 11.10 秒で、これには先ほどのパケットの RTO である 5.5 秒が含まれています。[Packet List] ペインの [Time] カラムを見れば、先の再送後 5.6 秒後にこの再送が行われているのがわかります。これはキャプチャファイルの最後のパケットであり、またプリンタもほぼ同時に印刷を中止しています。

今回のシナリオでは、自分のネットワーク内のクライアントとプリンタしか扱わないので、どちらに問題があるかを判断するだけで済みます。しばらくの間データの流れを観察すると、ある時点で、プリンタがクライアントへのレスポンスを行っていないことに気づきます。TCP再送からもわかるように、

クライアントはデータを送信しようと最善の努力をしていますが、プリンタがレスポンスを行わなくなっているのです。どのコンピュータがプリンタジョブを送ろうとこの問題は発生するので、プリンタに原因があると仮定してみます。

さらに解析すると、プリンタのRAMに異常を発見しました。大量の印刷ジョブが送られても数ページ分しか印刷しないのは、メモリのある領域にアクセスするまでしか印刷していないかのようです。結局は、メモリの問題によって、プリンタが新しいデータを受け取ることができず、クライアントによる印刷ジョブの送信を中断させてしまっていたのです。

8.3.4.3 学んだこと

このプリンタのトラブルはネットワークのトラブルではありませんが、Wiresharkを使って見つけることができました。これまでのシナリオとは違い、これはTCPトラフィックのみを対象としています。幸いなのは、2つの機器間で通信が中断されてしまった際に、TCPは有益な情報を残してくれることが多いことです。

今回は通信が突然停止したときに、TCPが持つ再送機能のおかげで、問題の箇所を正確に把握することができました。シナリオを読み進めていく中で、より複雑なトラブルを解決する場合でも、こうした機能に頼ることはよくあります。

8.3.5 孤立する支社

今回のシナリオに登場するのは、本社と、新たに設立されたばかりの支社を持つ企業です。この企業のITインフラのほとんどが本社内にあり、Windowsサーバのドメインを用いています。支社にはセカンダリドメインコントローラがあり、このドメインコントローラは、支社のDNSと認証リクエストを処理しています。このドメインコントローラはセカンダリDNSサーバであり、本社にある上位のDNSサーバからリソースレコードの情報を受け取るようになっています。

デプロイメントチームが支社に新インフラの導入を行っている際に、ネットワーク上のイントラネットWebアプリケーションサーバに支社から誰もアクセスできないことが判明しました。これらのサーバは本社にあり、WAN経由でアクセスするようになっています。支社の社員全員がアクセスできないのですが、問題なのはこれら内部サーバのみで、インターネットと支社のほかのリソースにはアクセスできるのです。

図8-33は今回のシナリオに登場する機器を示しています。

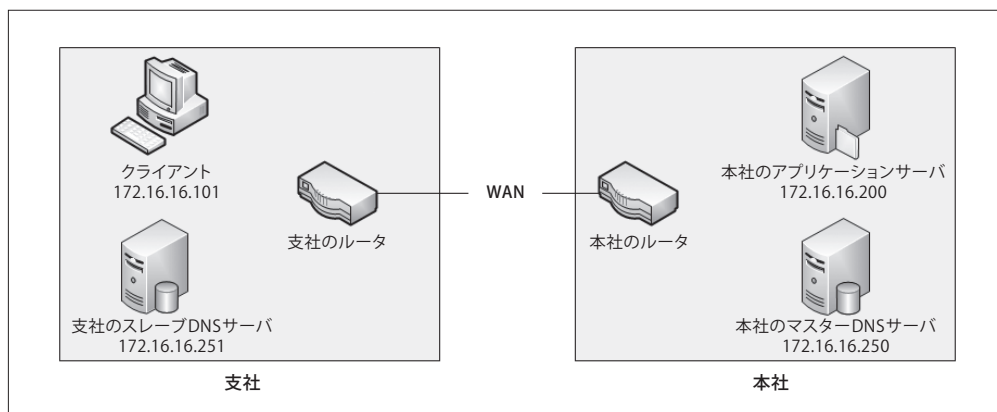


図8-33 標準的な支社のネットワークの構成機器

8.3.5.1 ケーブルへの潜入

stranded_clientside.pcap

問題は本社と支社間の通信にあるので、トラブルの追跡を始めるためにいくつかの場所でデータを収集します。支社のクライアント内に問題がある可能性を考えて、これらのうちの1台をポートミラーリングすることから始めましょう。情報を収集したら、それを使って追加でデータを収集する場所を決めます。クライアントから収集した最初のキャプチャファイルがstranded_clientside.pcapです。

8.3.5.2 パケット解析

stranded_branchdns.pcap

図8-34のように、アドレス172.16.16.101のコンピュータのユーザーが、本社のアプリケーションサーバ172.16.16.200で提供されているアプリケーションにアクセスしようとしたときから、最初のキャプチャファイルが始まっています。このキャプチャには2個のパケットしか含まれていません。最初のパケットに入っているのは、appserver③のAレコード②を求めて172.16.16.251に送られたDNSリクエスト①のようです。これは本社にある172.16.16.200のサーバのDNS名です。

図8-35からわかるように、このパケットに対するレスポンスがサーバ障害 (server failure) ①となっています。これは、DNSクエリが失敗したことを示しています。このパケットはエラー (サーバ障害) のため、クエリに対して応答していません②。

これで、通信のトラブルがDNSに関係していることがわかりました。支社のDNSクエリは172.16.16.251のDNSサーバで解決されるので、これが次のポイントとなります。

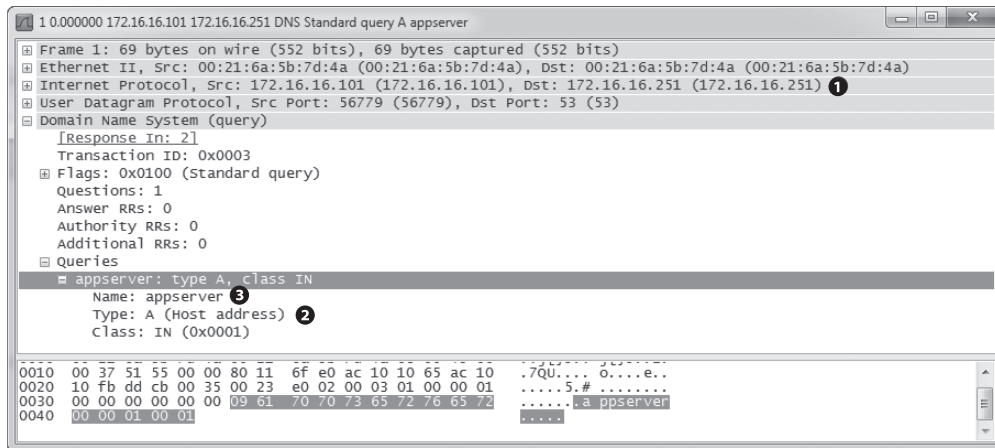


図8-34 appserverのAレコードに対するDNSクエリで始まる通信

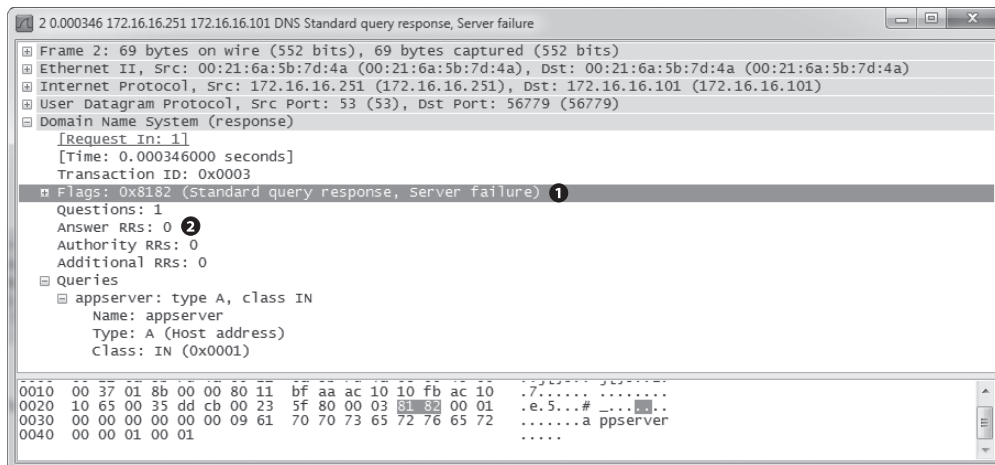


図8-35 クエリのレスポンスが上位の問題を示している

支社のDNSサーバからのトラフィックを適切にキャプチャするため、パケットキャプチャツールは設置したままで、ポートミラーリングの設定を変更してクライアントのトラフィックではなくDNSサーバのトラフィックをミラーするようにします。この結果のファイルがstranded_branchdns.pcapです。

図8-36のように、このキャプチャは先ほど見たクエリとレスポンス、そしてもうひとつのパケットで始まっています。このパケットはちょっと妙な感じです。DNSの標準である53番ポート②で本社のプライマリDNSサーバ(172.16.16.250)①との通信を試みっていますが、UDPではないからです③。

このパケットの目的を知るには、7章のDNSの説明を思い出してください。DNSは通常UDPを使いますが、クエリへのレスポンスが一定サイズを超える場合、TCPを使います。今回の場合、最初のUDPトラフィックがTCPトラフィックを引き起こしているようです。TCPはまたゾーン転送、つまり

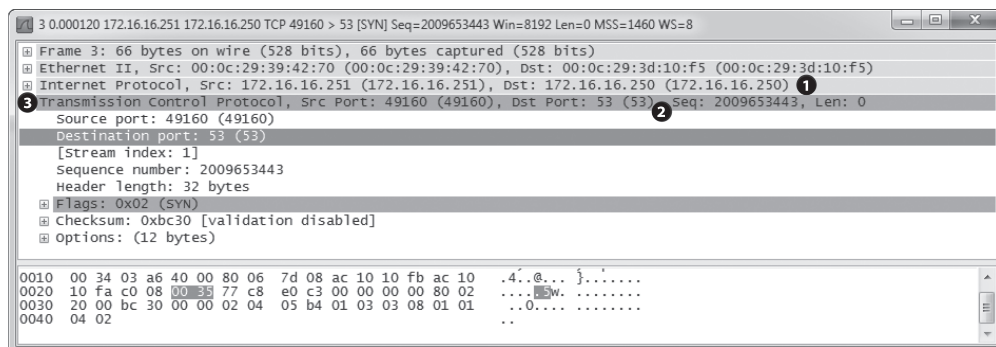


図8-36 このSYNパケットは53番ポートを使っているがUDPではない

リソースレコードがDNSサーバ間で転送される際にも使われますが、今回のケースはこれのようです。

支社にあるDNSサーバは本社のDNSサーバのスレーブであり、リソースレコードについては本社サーバに依存することになります。支社の社員がアクセスしようとしているアプリケーションサーバは本社にあり、本社のDNSサーバがそのサーバを管理しています。支社サーバがアプリケーションサーバへのDNSリクエストを解決するには、そのサーバのDNSリソースレコードが、本社DNSサーバから支社DNSサーバへと転送されなければなりません。キャプチャファイルにSYNパケットが入っているのはこのためだと考えられます。

SYNパケットに応答がないのは、このDNSトラブルが、支社と本社のDNSサーバ間でのゾーン転送障害にあるからです。さらに一歩進んで、ゾーン転送が失敗した理由を見つけてみましょう。犯人は本社支社間のルータか、本社のDNSサーバそのものであるということまで絞り込めます。本社DNSサーバのトラフィックをキャプチャして、SYNパケットがDNSサーバへ送られているかどうかを見えます。

本社DNSサーバトラフィックのキャプチャファイルはありません。トラフィックが存在しないからです。SYNパケットはサーバへ届いていなかったのです。技術者を送って本社と支社をつなぐルータの設定を確認してもらったところ、本社ルータの53番ポートではインバウンドのUDPトラフィックのみが許可され、TCPトラフィックは拒否するよう設定されていたことが判明しました。こうした単純な設定ミスが、サーバ間のゾーン転送を阻害し、支社のクライアントから本社の機器に対するクエリの解決を阻害していたのです。

8.3.5.3 学んだこと

犯罪ドラマを見れば、ネットワーク通信トラブルの捜査についてかなり学習できます。犯罪が起きますと、刑事はまず関係者の取材に着手します。そこで手がかりを得て、さらに捜査するという作業が、犯人が見つかるまで続けられるのです。

今回のシナリオでは、被害者（クライアント）の調査から始め、DNS通信のトラブルを見つけること

で解決への手がかりを得ました。その手がかりから、支社のDNSサーバ、本社のサーバ、最終的にはトラブルの原因であるルータまでたどり着いたのです。

パケット解析を行う場合、パケットは手がかりだと考えるようにしましょう。手がかりは誰が罪を犯したかを教えてはくれるとは限りませんが、最終的には犯人まで導いてくれます。

8.3.6 イライラする開発者

IT業界では、開発者とシステム管理者がしょっちゅう口論しています。開発者はいつも、プログラムの不具合は、ネットワークの手抜き設定と機器の機能不全のせいだと文句を言っています。一方システム管理者は、ネットワークエラーと通信の遅延はコードに問題があると非難しています。

今回のシナリオは、開発者が、複数の店舗の売り上げを追跡して中央データベースへと報告を返すアプリケーションを開発したところです。通常の営業時間中は帯域を節約するため、これはリアルタイムなアプリケーションではありません。報告データは日中蓄積され、夜間にCSVファイルとして中央データベースに書き込まれます。

ところがこの新規開発したアプリケーションが正しく機能しません。店舗から送られたファイルはサーバが受信していますが、データベースに書き込まれるデータに問題があるのです。項目が抜け落ち、データの位置に間違いがあり、完全に抜けているデータもあります。プログラマがネットワークに問題があると言うので、システム管理者は狼狽しています。プログラマは、ファイルが店舗から中央データリポジトリへ転送される途中で消失しているはずだと言うのです。彼の間違いを証明するのが、今回の目標です。

8.3.6.1 ケーブルへの潜入

tickedoffdeveloper.pcap

必要なデータを収集するために、店舗のひとつ、または本社でパケットをキャプチャします。このトラブルはすべての店舗で発生しているので、ネットワークに問題があるとしたら、すべての店舗に唯一共通する箇所である、本社で発生しているはずです。

ネットワークスイッチはポートミラーリングに対応しているので、サーバが接続しているポートをミラーし、トラフィックをキャプチャします。キャプチャしたトラフィックは、サーバへCSVファイルをアップロードしている店舗ごとに分割します。このキャプチャファイルがtickedoffdeveloper.pcapです。

8.3.6.2 パケット解析

プログラマが開発したアプリケーションについては、まったく知識がありません。キャプチャファイルはFTPトラフィックで始まっているようなので、これが実際にファイルを転送しているメカニズムなのかどうかを調査します。通信をまとめてみるには、通信フローグラフが便利です。メニューから[Statistics] → [Flow Graph]を選択し、[OK] ボタンをクリックしましょう。図8-37がそのグラフです。



図8-37 フローグラフでFTP通信がひとめでわかる

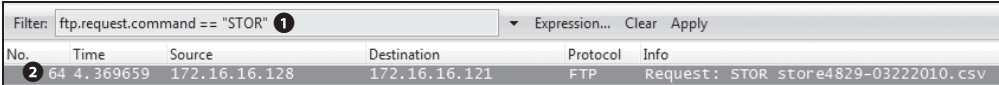
このフローグラフを見ると、172.16.16.128と172.16.16.121との間でFTP接続が設定されています①。172.16.16.128が接続を開始しているの②、これがクライアントで、172.16.16.121はデータを収集して処理するサーバだと推測できます。フローグラフから、このトラフィックはFTPプロトコルのみを使っていることが確認できます。

ここである種のデータ転送が行われているはずなので、FTPの知識を使って、転送が始まってい

るパケットの場所を見つけます。FTPコネクションとデータ転送はクライアント側から始まるので[†]、FTPサーバへのデータアップロードに使用される、FTP STOR コマンドを探しましょう。そのためにはフィルタを設定するのが一番簡単です。

このキャプチャファイルの中にはFTPリクエストのコマンドが散乱しているので、[Filter Expression] ダイアログで数百ものプロトコルやオプションの一覧からフィルタを選択していくのではなく、[Packet List] ペインで直接フィルタを構築しましょう。それにはまず、FTPリクエストコマンドのあるパケットを選択する必要があります。一覧の上部に近いので、5番目のパケットを選びましょう。次に[Packet Details] ペインで[FTP]を展開し、[USER]を展開します。[Request Command: USER] フィールドを右クリックし、[Prepare a Filter]を選択し、最後に[Selected]を選びます。

これでFTP USERリクエストコマンドを含むパケットすべてを抽出するフィルタが[Filter] ダイアログに準備されました。その後、図8-38のようにUSERをSTORに置き換えるよう、フィルタを編集します^①。



No.	Time	Source	Destination	Protocol	Info
2	64 4.369659	172.16.16.128	172.16.16.121	FTP	Request: STOR store4829-03222010.csv

図8-38 このフィルタはデータ転送が始まる場所を識別するのに役立つ

Enterキーを押してフィルタを有効にすると、STORコマンドは、64番目のパケット1つしか存在していないことがわかります^②。

データ転送の始まる場所がわかったので、[Packet List] ペイン上部の[Clear] ボタンをクリックしてフィルタを解除します。

キャプチャファイルを64番目のパケットから調べてみると、このパケットがstore4829-03222010.csvファイルの転送を指示していることがわかります(図8-39)。

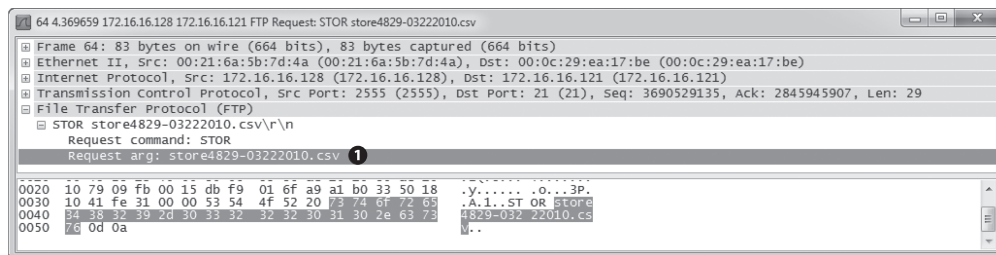


図8-39 FTPを使って転送されるCSVファイル

STORコマンドに続くパケットは異なるポートを使っていますが、FTP-DATA転送として識別されています。データが転送されていることは確認できましたが、プログラミングの間違いは証明できてい

[†] 監訳注：PASVモードでない「通常の」FTPトラフィックの場合、データ転送はサーバ側から開始されます。

ません。キャプチャしたパケットから転送されたファイルの内容を抽出して、ネットワーク上を転送されたあともファイルの内容がそのままであることを示す必要があります。

ファイルが暗号化されていない形式で転送される場合は、セグメントに分割されて、宛先で組み立てられます。今回のシナリオでは、宛先には到着したものの、まだ組み立てられていないパケットをキャプチャしています。データはすべて揃っているので、ファイルをデータストリームとして抽出し、組み立てるだけです。組み立てるには、FTP-DATAストリームにあるパケット（66番目のパケットなど）を選び、[Follow TCP Stream] をクリックします。すると図8-40のようにTCPストリームが表示されます。

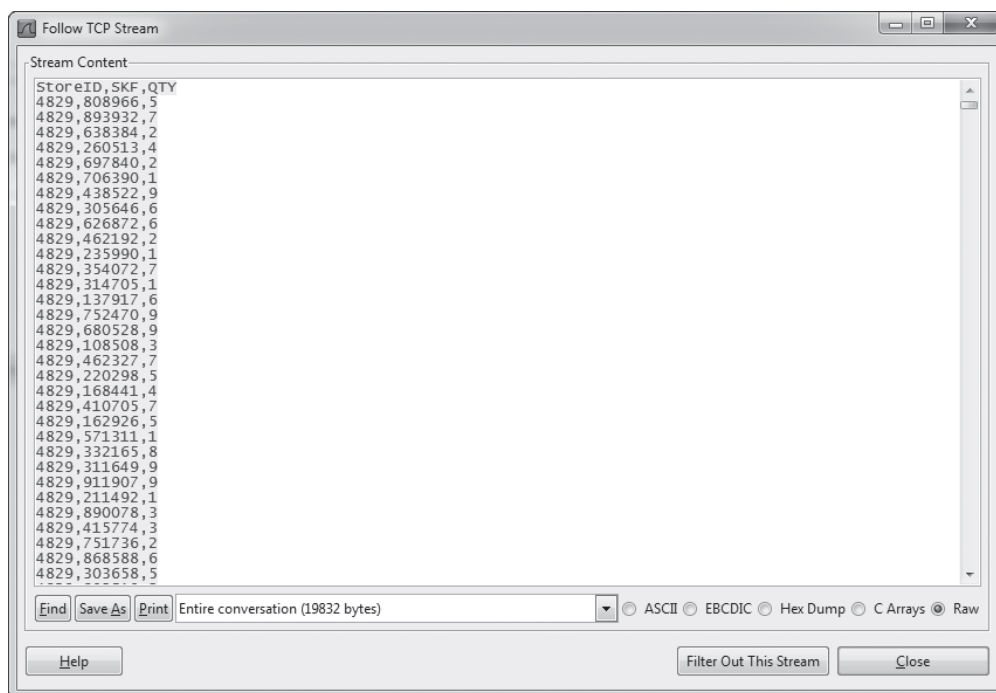


図8-40 転送されているデータがTCPストリームに表示される

データはFTP上を平文の状態転送されているため表示されていますが、ストリームからは、このファイルが改変されていないかどうかを判断できません。組み立てたデータは元々の形式で保存できるので、[Save As] ボタンをクリックし、図8-41のように64番目のパケットに表示されるファイル名を指定します。[Save] ボタンをクリックしてください。

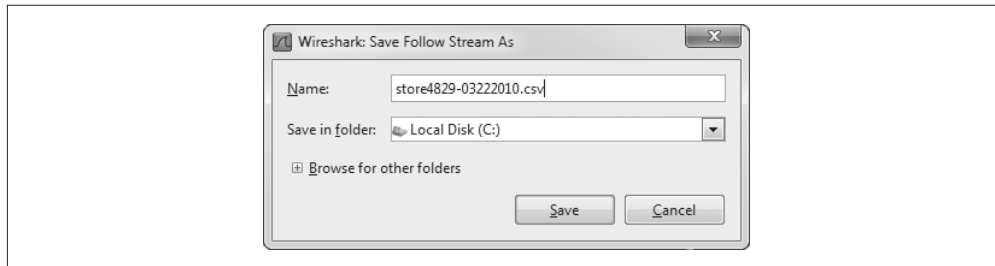


図8-41 ストリームを元々のファイル名で保存する

保存すると、店舗システムから転送したファイルとバイトレベルでまったく同じ内容のCSVファイルができるはずです。元々のファイルのMD5ハッシュと抽出したファイルのMD5ハッシュを比較することでこれを検証でき、図8-42のように、MD5ハッシュが同じになるはずです。

ファイルを比較すれば、アプリケーション内で起きているデータベース障害の原因がネットワークでないことが証明できます。店舗システムから中央の収集サーバへと転送したファイルは、サーバに到着した時点では改変されていないので、ファイルの破損はアプリケーションがファイルを処理している間に起こったこととなります。

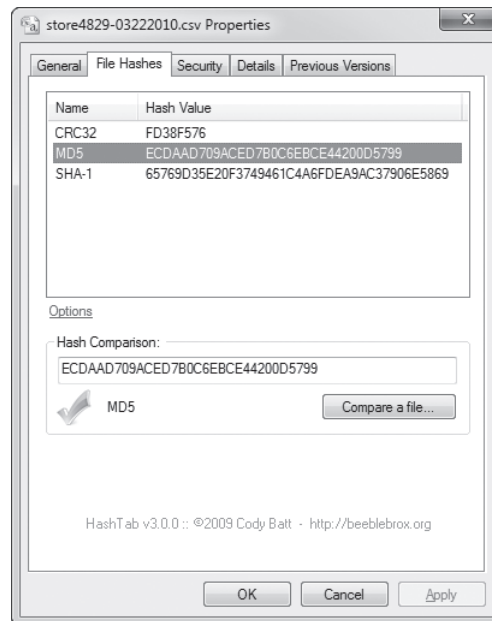


図8-42 元々のファイルと抽出したファイルのMD5ハッシュが同一

8.3.6.3 学んだこと

パケットレベルの解析が素晴らしいのは、ごちゃごちゃしたアプリケーションと対峙しなくて済む点です。作りが雑なアプリケーションは、優れたものよりはるかに多いものですが、パケットレベルでは問題になりません。プログラマはアプリケーションが依存している不可思議なコンポーネントについて気に病んでいますが、数百行ものコードを使った複雑なデータ転送も、結局はFTPやTCP、あるいはIP以上のものではありません。これらの基本プロトコルの知識を利用すれば、通信処理が正しく処理されていることを確認し、ファイルを抽出してネットワークの堅牢性を証明することさえ可能です。問題がどれほど複雑に見えたとしても、しょせんはパケットにすぎないと覚えておきましょう。

8.4 まとめ

本章では、パケット解析によって問題のある通信の理解が容易になるような基本的なシナリオをいくつか例として挙げました。一般的なプロトコルで基本的な解析を使えば、ネットワークのトラブルを追跡し、短時間で解決できるものです。まったく同じシナリオには直面しないでしょうが、ここで紹介した解析テクニックは、個々に発生するトラブルの解析にきっと役立つはずです。

9章

ネットワークの遅延と戦う

ネットワーク管理者は、日々ネットワークの遅延と戦わなくてはなりません。しかしながら、ネットワークが遅いからといって、それが即ネットワークの非難につながるわけではないのです。ネットワークの遅延と格闘する前に、まずは本当にネットワークが遅いのかどうかを確認する必要があります。この章ではその手法を紹介します。

まずはTCPのエラーリカバリとフロー制御機能について説明します。次にネットワーク遅延の原因を追及する方法を探ります。最後にネットワークと、ネットワーク上で動作する機器やサービスのベースライン設定を見ていきます。この章が終わる頃には、ネットワーク遅延を検出、診断し、トラブルシューティングできる力が身についているはずです。



ネットワーク遅延の解決に利用できるテクニックはいくつかあります。この章ではもっとも扱うことの多いTCPに的を絞りました。TCPを使えば、(ICMPと同様に) 余分なトラフィックを生成することなく時系列での解析を行うことができます。

9.1 TCPのエラーリカバリ機能

TCPのエラーリカバリ機能は、ネットワーク上の高遅延箇所を確認、診断、修復するための最良のツールです。コンピュータネットワーキング用語では、**遅延**（レイテンシ）とはパケットが送信されてから受信されるまでの時間のことです。

遅延は一方方向（送信元から宛先まで）でも、往復（送信元から宛先までに加え、宛先から送信元まで）でも計測されます。通信機器間での通信速度が速く、パケットがある地点から別の地点へ届くまでの時間が短ければ、**低遅延**ということになります。反対にパケットが届くまでに相当な時間がかかる場合、**高遅延**とみなされます。高遅延はネットワーク管理者の最大の敵です。

6章では、TCPがシーケンスとACK番号を使うことで、パケットを確実に転送する方式について説明しました。本章では、再度シーケンスとACK番号に着目し、高遅延により受信時にシーケンス番号が乱れた場合（あるいはまったく受信しなかった場合）、TCPがどう応答するかを見ていきます。

9.1.1 TCP再送

`tcp_retransmissions.pcap`

パケットの再送は、TCPのもっとも基本的なエラーリカバリ機能のひとつです。この機能はパケット消失を防ぐのを目的としています。

パケット消失には、アプリケーションの異常、ルータのトラフィック輻輳、一時的なサービス障害といった、さまざまな原因が考えられます。パケットレベルでは状況が頻繁に変化し、パケット消失は一時的なものである場合が多いため、TCPとしては迅速にこれを検知し、消失をリカバリすることが重要となります。

パケット再送が必要かどうかを決める主な機構は、**再送タイマー**と呼ばれています。タイマーは**RTO（再送タイムアウト）**と呼ばれる値に従って動作します。TCPを使ってパケットが送信されると再送タイマーがスタートし、そのパケットのACKが受信されるとストップします。パケットが送信されてからACKパケットが受信されるまでの時間を**RTT（ラウンドトリップタイム）**と呼びます。この時間の平均値が、最終的なRTO値の算出に用いられます。

RTO値が決まるまで、送信はデフォルトのRTT設定に基づいて行われます。この設定は機器間の最初の通信のために設定されたもので、受信したパケットのRTTをもとに、実際のRTOが算出されます。

RTO値が決まると、送信する各パケットに対して、再送タイマーがパケットの消失が発生したかどうかを判断するために用いられます。図9-1はTCPの再送処理を図式化したものです。

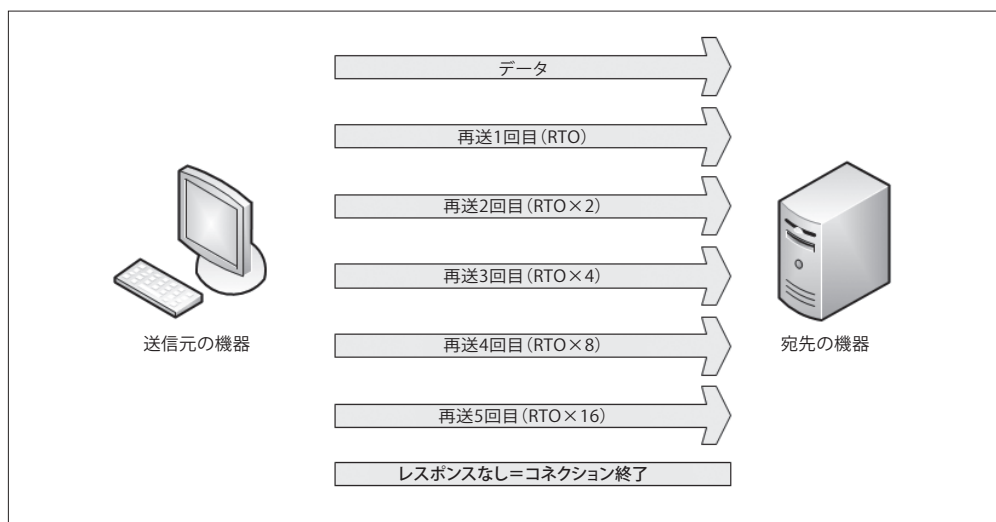


図9-1 TCP再送処理の概念図

パケットを送信した際に、受信者がTCP ACKパケットを送信しないと、送信元は送信したパケットが消失したと判断してパケットを再送します。再送が行われた際のRTO値は倍になります。RTO値に達する前にACKパケットが届かない場合、パケットは再再送されます。それでもACKが届かない

場合、RTO 値はさらに倍になります。ACK パケットを受信するまで、または最初に設定した再送回数の最大値に達するまで、この処理は繰り返されます。RTO 値は再送するたびに倍増していきます。

再送回数の最大値はOSの設定によります。Windowsの場合、デフォルトの最高再送回数は5回です。大半のLinuxの場合は15回です。

TCP再送のサンプルを見るため、ファイルtcp_retransmissions.pcapを開きましょう。これには6個のパケットが含まれています。図9-2に最初のパケットを示します。

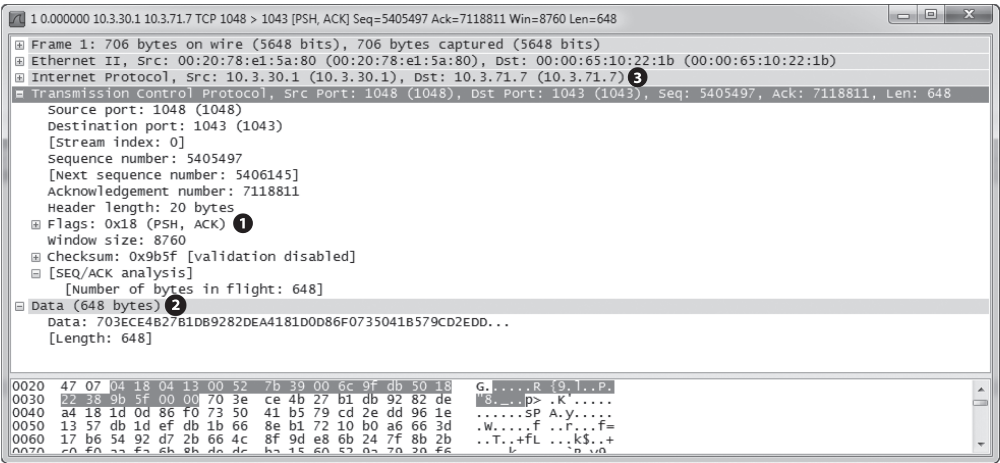


図9-2 データを含んだ単純なTCPパケット

これは648バイトのデータ②を含むTCP PSH/ACKパケット①で、10.3.30.1から10.3.71.7へ送られたものです③。典型的なデータパケットです。

通常の状況であれば、最初のパケットが送信されるとすぐ、TCP ACKパケットがレスポンスとして送られてくるはずですが。しかしここでは、次のパケットが再送パケットとなっています。これは、[Packet List] ペインを見るとわかります。[Info] カラムにははっきりと [TCP Retransmission] とあり、パケットは黒字に赤いテキストで表示されています。図9-3は [Packet List] ペインにおけるパケット再送の例です。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.3.30.1	10.3.71.7	TCP	1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648
2	0.206000	10.3.30.1	10.3.71.7	TCP	[TCP Retransmission] 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648
3	0.600000	10.3.30.1	10.3.71.7	TCP	[TCP Retransmission] 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648
4	1.200000	10.3.30.1	10.3.71.7	TCP	[TCP Retransmission] 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648
5	2.400000	10.3.30.1	10.3.71.7	TCP	[TCP Retransmission] 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648
6	4.000000	10.3.30.1	10.3.71.7	TCP	[TCP Retransmission] 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 win=8760 Len=648

図9-3 [Packet List] ペインにおけるパケット再送

図9-4のように、Packet Detailsおよび [Packet Bytes] ペインでも、パケットが再送されたかどうかがわかります。

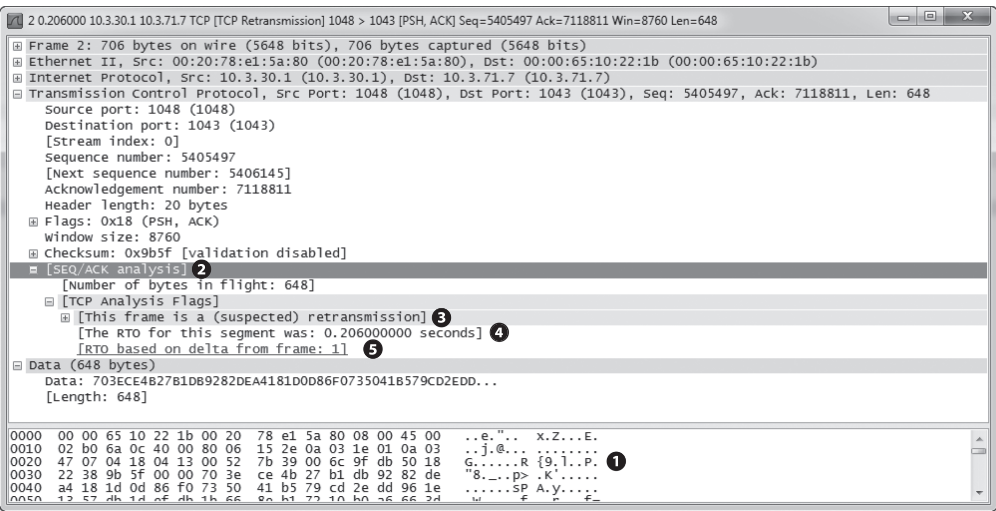


図9-4 再送されたパケット

このパケットは (IPの「識別子」および「Checksum」フィールドを除き) 元々のパケットと同じものです。2つのパケットの「Packet Bytes」ペインを比較すれば、同一であることが検証できます①。

「Packet Details」ペインを見ると、再送されているパケットの「SEQ/ACK Analysis」行の下に、いくつか情報があることに気づくでしょう。これはWiresharkが提供している情報で、パケットそのものには含まれていません。SEQ/ACK解析から、これが確かに再送であり③、RTO値は0.206秒④、そしてRTO値は1番目のパケットからのデルタタイム (相対時間) をもとにしていることが確認できます。

残りのパケットも同様で、IPの「識別子」および「Checksum」フィールドとRTO値だけが異なる値となります。各パケットの時間差を目で確認するには、「Packet List」ペインの「Time」カラムを参照します (図9-5)。再送でRTO値が倍増するごとに、時間間隔が大幅に延びていることがわかります。

TCPの再送機能は、送信者がパケット消失を検出した際に、リカバリを行うために用いられます。今度は受信者がパケットの消失を検出し、リカバリする際に用いる重複ACK (duplicate acknowledgments) 機能を見てみましょう。

No.	Time
1	0.000000
2	0.206000
3	0.600000
4	1.200000
5	2.400000
6	4.805000

図9-5 RTO値の増加を示す「Time」カラム

9.1.2 重複ACKと高速再送

tcp_dupack.pcap

重複ACKは、受信者が順番の乱れたパケットを受け取ったときに送るTCPパケットです。TCPは、データが送信されたときと同じ順番で受信され、再度組み立てられることを保証するために、ヘッダ内のシーケンス番号とACK番号フィールドを使っています。



TCPパケットは厳密に言えばTCPセグメントと呼ぶのが適切ですが、一般にはパケットと呼ばれています。

新たなTCPコネクションが確立されたときに、ハンドシェイク処理で交換される非常に重要な情報のひとつが**イニシャルシーケンス番号 (ISN)**です。ISNがコネクションの両側で設定されると、パケットが送信されるごとに、そのパケットのデータのサイズ分だけシーケンス番号が増えていきます。

ある機器のISNが5000で、500バイトのパケットを送信するとします。このパケットが受信されると、受信側はACK番号5500のTCP ACKパケットで返信します。

シーケンス番号 + 受信されたデータのバイト容量 = ACK番号

送信元へ返されるACK番号は、受信者が次に受け取るシーケンス番号だということになります。これを図式化したのが図9-6です。



図9-6 TCPシーケンス番号とACK番号

受信者はシーケンス番号を見れば、パケット消失を検出できます。シーケンス番号を追跡していれば、番号が乱れていないかどうかを確認できるのです。

想定外のシーケンス番号を受信した場合、パケットが消失したと仮定できます。データを適切に組み立てるには、消失したパケットを受信する必要があるため、消失したパケットのシーケンス番号を含むACKパケットを再送し、送信元に再送を促します。

送信者は3回重複ACKを受け取ると、パケットが本当に消失したと判断し、即座に**高速再送**を行います。高速再送が開始されると、高速再送パケットが送信されるまで、ほかのすべてのパケットの送信が一時停止されます。この処理を図に表したのが図9-7です。

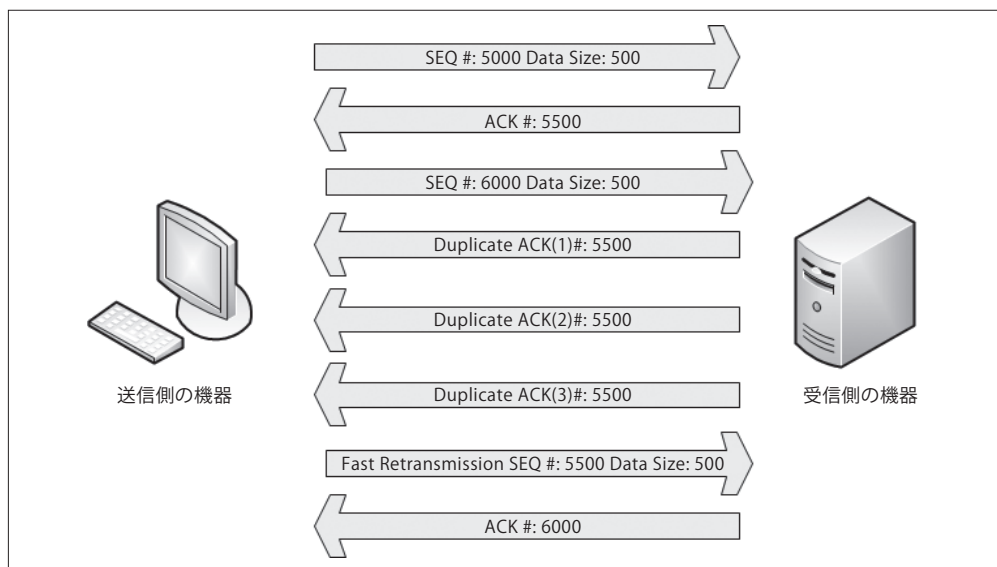


図9-7 受信側からの重複ACKが高速再送につながる

ファイルtcp_dupack.pcapに重複ACKと高速再送のサンプルを示します。このキャプチャファイルの最初のパケットが図9-8です。

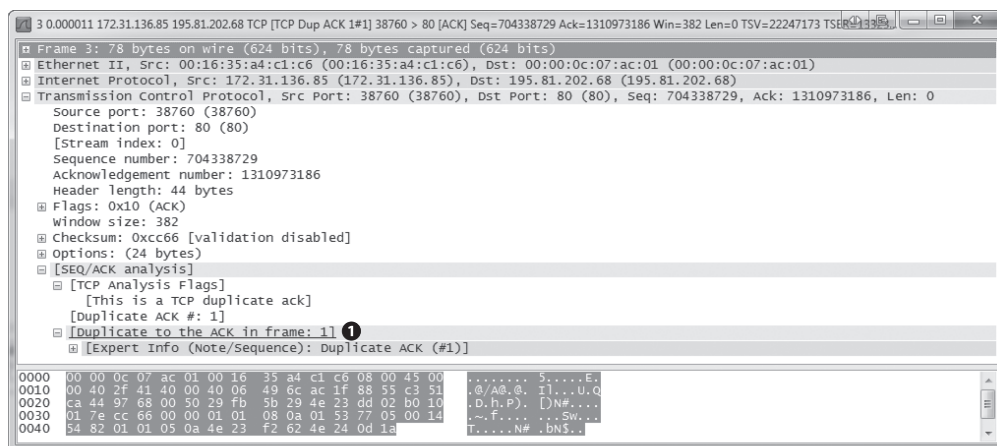


図9-8 ACKが次のシーケンス番号を示している

データ受信者（172.31.136.85）から送信者（195.81.202.68）へ送信されたこのTCP ACKパケット①は、このキャプチャファイルには含まれていないパケットで送られたデータに対するACKです。



Wiresharkのデフォルト設定ではシーケンス番号の解析を簡単にするために、相対シーケンス番号を使うようになっていますが、次ページの例や画面キャプチャではこの機能を使っていません。この機能を無効にするには、メニューから [Edit] → [Preferences] を選択し、[Preferences] ダイアログで [Protocols]、[TCP] を順に選び、[Relative sequence numbers] の横のボックスのチェックを外してください。

このパケットのACK番号は1310973186❷で、これが次に受信するパケットのシーケンス番号になるはずです (図9-9)。

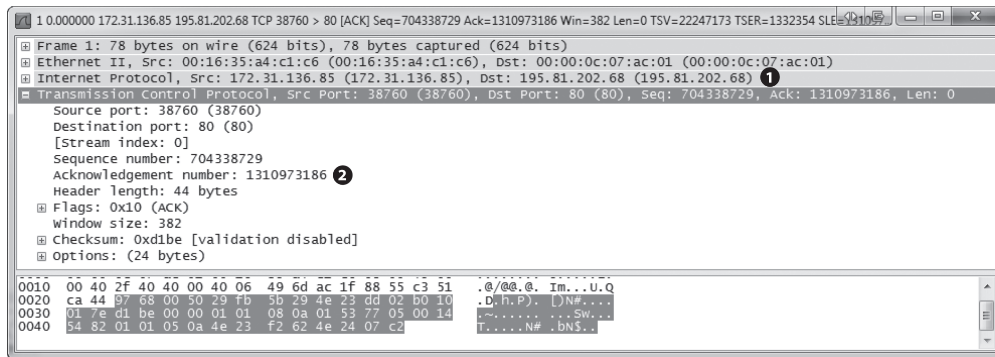


図9-9 予測していないシーケンス番号のパケット

残念ながら次のパケットのシーケンス番号は1310984130❶でした。これは予期したものではなく、パケットが送信の途中で消失したことを意味しています。受信側の機器は、このパケットのシーケンス番号が乱れていることを検出し、3番目のパケットとして図9-10の重複ACKを送信します。

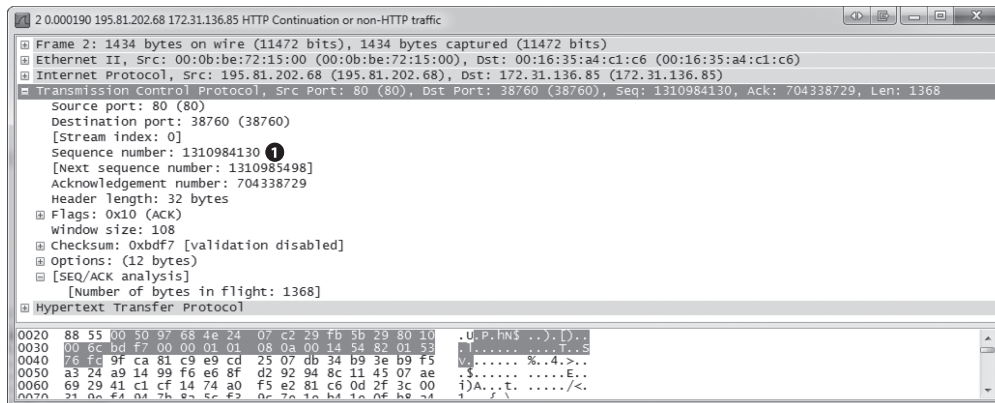


図9-10 最初の重複ACKパケット

次のいずれかの方法で、このパケットが重複ACKパケットであることを確認できます。

- [Packet Details] ペインの [Info] カラムが、黒字に赤い文字で表示されている。
- [Packet Details] ペインの SEQ/ACK Analysis 行を展開してみると、1 番目のパケットの重複 ACK であることが表示されている。

図9-11のように、続くいくつかのパケットでもこれが続きます。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.31.136.85	195.81.202.68	TCP	38760 > 80 [ACK] Seq=704338729 Ack=1310973186 win=382 Len=0 TSv=22247173 TSEr=
2	0.000190	195.81.202.68	172.31.136.85	HTTP	Continuation or non-HTTP traffic
3	0.000001	172.31.136.85	195.81.202.68	TCP	[TCP Dup ACK 1#1] 38760 > 80 [ACK] Seq=704338729 Ack=1310973186 win=382 Len=0
4	0.000093	195.81.202.68	172.31.136.85	HTTP	Continuation or non-HTTP traffic
5	0.000010	172.31.136.85	195.81.202.68	TCP	[TCP Dup ACK 1#2] 38760 > 80 [ACK] Seq=704338729 Ack=1310973186 win=382 Len=0
6	0.000121	195.81.202.68	172.31.136.85	HTTP	Continuation or non-HTTP traffic
7	0.000010	172.31.136.85	195.81.202.68	TCP	[TCP Dup ACK 1#3] 38760 > 80 [ACK] Seq=704338729 Ack=1310973186 win=382 Len=0

図9-11 シーケンス番号が乱れたために生成された重複ACK

送信側から送られた4番目のパケットは、不適切なシーケンス番号が付いていたので①、受信側は2個目の重複ACKを送信します②。するとまた不適切なシーケンス番号のパケットが届いたため③、3個目の最後の重複ACKが送られました④。

送信元は3個目の重複ACKを受け取るとすぐに、すべてのパケットの送信を中断し、消失パケットを再送します。図9-12は消失パケットの高速再送です。

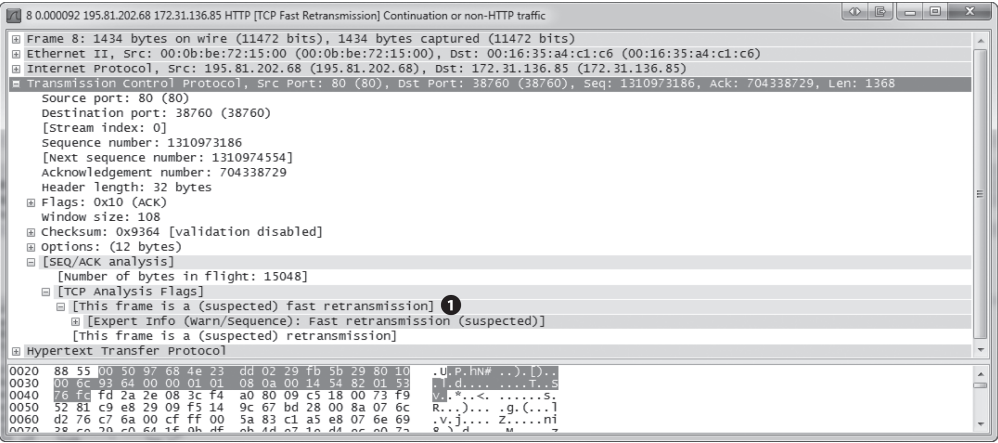


図9-12 重複ACKによる消失パケットの高速再送

繰り返しますが、パケットの再送は [Packet Details] ペインの [Info] カラムで確認できます。先の例同様、黒地に赤文字で書かれているのでよくわかります。[SEQ/ACK Analysis] から、これが高速再送であることがわかります①（繰り返しますが、このパケットが高速再送だと示しているのは、パケットではなくWiresharkの機能です）。キャプチャファイルの最後のパケットは、高速再送の受信を示すACKパケットとなります。



パケット消失が起きたTCP通信のデータフローに影響を与える場合のある機能として、セレクティブACK (Selective Acknowledgement) 機能が挙げられます。上のパケットキャプチャでは、最初の3ウェイハンドシェイクプロセスにおいて、セレクティブACKが有効な機能としてネゴシエートされています。この場合パケットが消失して重複ACKを受け取ると、消失したパケット以外のパケットの受信が成功していれば、消失したパケットのみが再送されます。セレクティブACKが有効になっていないと、消失したパケットのあとに送信されたすべてのパケットも、消失パケットとともに再送信されます。つまりセレクティブACKによって、データのリカバリを効率的に行うことができるのです。最新のTCP/IPスタックの実装はセレクティブACKをサポートしているので、通常はこの機能が実装されているはずです。

9.2 TCPのフロー制御

再送と重複ACKはパケットの消失に対するリカバリを行うためのTCP機能です。TCPにパケット消失を予防する機能がなかったとしたら、悲惨なことになっていたと思いますが、幸いにしてTCPにはその機構も備わっています。

TCPは、パケット消失が起ころうになると、データ転送レートを調整してこれを防ぐスライディングウィンドウという機構を実装しています。スライディングウィンドウは、受信者の受信ウィンドウを使ってデータフローを制御します。

受信ウィンドウは受信者が指定する値で、TCPヘッダに格納されており(単位: バイト)、TCPのバッファ領域に格納するデータ量を送信者に伝えるものです。このバッファ領域はデータが一時的に保管される領域で、その後、データはアプリケーション層プロトコルに渡されて処理を待ちます。送信元が1回に送信できるデータ量は、Window Size フィールドに指定された値で決まります。送信者がさらに多くのデータを送信するには、受信者側が先のデータを受信したというACKを送信する必要があります。また受信者はデータを処理して、TCPバッファ領域を空ける必要があります。図9-13に受信ウィンドウの仕組みを図式化しています。

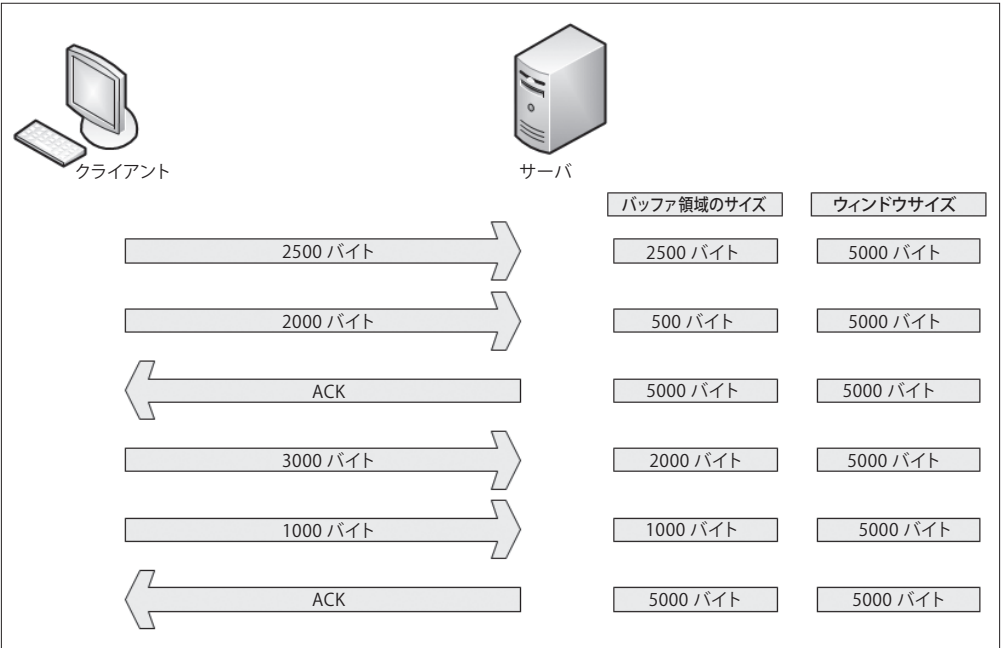


図9-13 受信者が受け取るデータ量を制御する受信ウィンドウ

図9-13では、受信ウィンドウのサイズが5000バイトのサーバにクライアントがデータを送信しています。クライアントが2500バイトのデータを送信するとサーバのバッファ領域は2500バイトへと減少し、さらに2000バイトが送られると500バイトへと減少します。ここでサーバはこのデータのACKを送ります。バッファのデータを処理したので、またバッファに空きができました。この処理が繰り返され、クライアントが3000バイト、1000バイトとデータを送信し、サーバのバッファは1000バイトへと減少します。クライアントは再度ACKを送り、バッファのデータを処理します。

9.2.1 ウィンドウサイズの調整

ウィンドウサイズの調整は明確な作業ですが、常にうまくいくとは限りません。TCPがデータを受け取ると、ACKが生成されレスポンスとして送信されますが、受信者のバッファにあるデータが常に迅速に処理されるわけではないのです。

サーバが多数のクライアントからのパケット処理に忙殺されてしまうと、バッファを空ける作業が遅延してしまい、次のデータを受け取る場所が確保できなくなります。フロー制御が行われなければ、これはパケット消失とデータ障害につながります。幸いにも、サーバが輻輳してデータ処理が遅延している場合、受信ウィンドウのサイズを調整することができます。これは、ACKパケットのTCPヘッダで、ウィンドウサイズの値を小さくすることで行われます。図9-14にその例を示します。

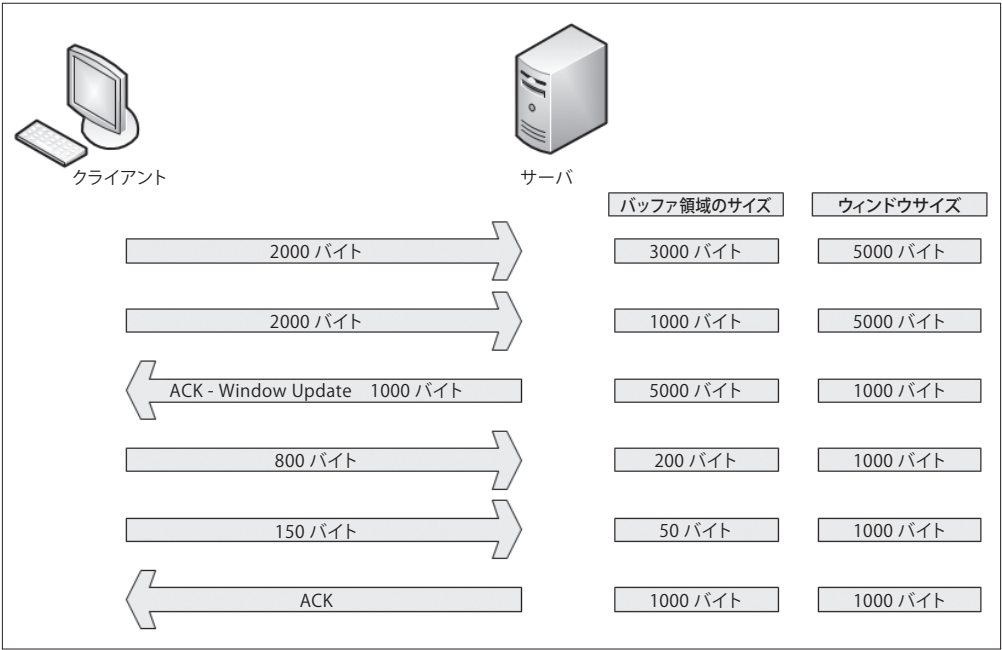


図9-14 サーバが輻輳した際にウィンドウサイズを調整する

図9-14では、サーバのウィンドウサイズは5000バイトから始まっています。クライアントが2000バイトのデータを送り、さらに2000バイト送ったので、バッファ領域の空きは1000バイトになってしまいました。サーバはバッファがすぐにいっぱいになってしまうと認識し、このペースでデータが転送されれば、パケット消失が起こると判断しました。そこでサーバは、ウィンドウサイズを1000バイトにするというACKをクライアントに送ります。結果としてクライアントから送信されるデータ量が減少し、サーバはデータフローを一定速度で保ちつつ、バッファのデータが処理できるようになりました。

ウィンドウサイズを大きくする場合もあります。サーバがより高速にデータ処理ができれば、ウィンドウサイズを拡大するというACKパケットを送信します。

9.2.2 ゼロウィンドウ通知でのデータフローの一時停止

サーバがこれ以上クライアントからのデータを処理できないという場合があります。原因としてはメモリ不足、処理能力の不足といった要因が考えられます。これらが起きるとパケット消失や通信処理の中断につながりますが、受信ウィンドウを使えば被害を最小限に留めることが可能です。

こうした問題が発生した場合、サーバは受信ウィンドウサイズがゼロであるというゼロウィンドウ通知を含むゼロウィンドウパケットを送信します。クライアントがこのパケットを受信すると、データ転送を停止しますが、キープアライブパケットを送り、サーバとの接続は維持します。キープ

アライブパケットは、サーバの受信ウィンドウの状況を確認するため、一定間隔で送られます。サーバがデータ処理を再開する場合には、ゼロ以外のウィンドウサイズを通知するパケットを送ることで、通信が再開させます。図9-15はゼロウィンドウ通知の例を示しています。

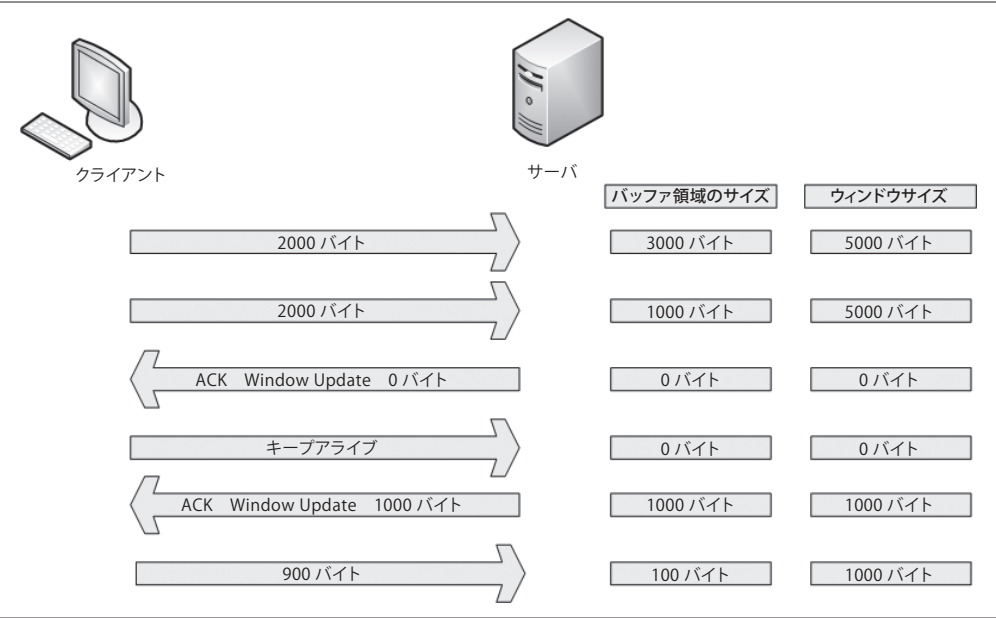


図9-15 ウィンドウサイズがゼロに設定されるとデータ転送が停止する

図9-15では、サーバはウィンドウサイズ5000バイトでデータ受信を開始しています。クライアントから4000バイトのデータを受信すると、プロセッサ負荷が非常に高くなり、クライアントからのデータを一切処理できなくなっていました。サーバはWindow Size フィールドを0に設定したパケットを送信し、クライアントはデータ転送を中止してキープアライブパケットを送ります。その後サーバでデータ受信が可能になり、ウィンドウサイズを1000バイトにするという通知を送ったので、クライアントはデータ送信を再開しました。

9.2.3 TCPスライディングウィンドウの実例

tcp_zerowindowrecovery.pcap
tcp_zerowindowdead.pcap

ここまでTCPスライディングウィンドウの原理について説明してきました。次に、キャプチャファイルtcp_zerowindowrecovery.pcapで実際に見てみましょう。

このファイルは192.168.0.20から192.168.0.30へ送られた複数のTCP ACKパケットで始まっています。一番の着眼点はWindow Size フィールドの値ですが、これは[Packet List] ペインの[Info] カラムと、[Packet Details] ペインのTCPヘッダの両方で見られます。最初の3つのパケットでは、このフィールドの値が減っていることがすぐにわかります(図9-16)。

No.	Time ❷	Source	Destination	Protocol	Info	❶
1	0.000000	192.168.0.20	192.168.0.30	TCP	2235 > 1720 [ACK] Seq=1422793785 Ack=2710996659 win=8760 Len=0	
2	0.000237	192.168.0.20	192.168.0.30	TCP	2235 > 1720 [ACK] Seq=1422793785 Ack=2710999579 win=5840 Len=0	
3	0.000193	192.168.0.20	192.168.0.30	TCP	2235 > 1720 [ACK] Seq=1422793785 Ack=2711002499 win=2920 Len=0	

図9-16 パケットのWindow Sizeが減っている

最初のパケットの8760バイトが2番目のパケットでは5840バイトに、3番目のパケットではさらに2920バイトへと減っています❶。ウィンドウサイズの減少は、受信先からの通信に遅延が起きている場合の典型的な証拠です。[Time] カラムを見ると、サイズが非常に早く減らされているのがわかります❷。ウィンドウサイズがこれだけ早く減らされると、ゼロまで下がる場合が多く、実際に4番目のパケットがそうになっています (図9-17)。



図9-17 ゼロウィンドウ通知は機器がデータをこれ以上受信できないという意味

4番目のパケットも192.168.0.20から192.168.0.30へ送られたものですが、これは、これ以上データを受け取れないと通知するのが目的です。TCPヘッダにあるゼロ値❶、そして[Packet List] ペインの[Info] カラムと、TCPヘッダのSEQ/ACK Analysisセクション❷を見れば、これがゼロウィンドウパケットだということがわかります。

ゼロウィンドウパケットが送信されると、192.168.0.30の通信機器は、192.168.0.20からウィンドウサイズの増加を通知するWindow Updateを受け取るまで、データを送信しません。ここでゼロウィンドウを引き起こした問題は一時的なものです。そのため、図9-18のように、次のパケットでWindow Updateが図9-18のように送られています。

これでウィンドウサイズは健全な範囲の64240バイトまで増加しました❶。SEQ/ACK Analysis行を見れば、これがWindow Updateだとわかります。

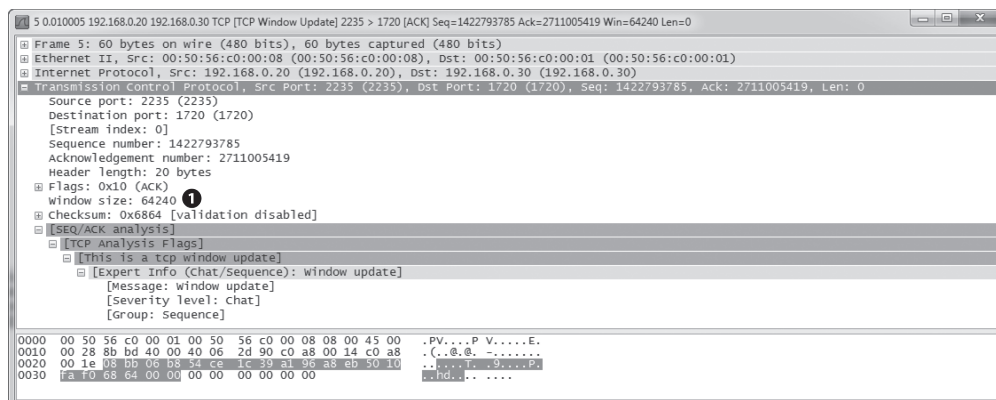


図9-18 データ送信が可能になったことを伝えるTCPのWindow Updateパケット

Window Update パケットが受信されると、192.168.0.30のホストは再びデータ送信が可能になるので、6番目と7番目のパケットを送ります。この処理は非常に迅速に行われます。少しでも余分に時間がかかると、データ転送の遅延や障害を引き起こすからです。

もう一度だけ、ファイルtcp zerowindowdead.pcapにあるスライディングウィンドウを調べてみましょう。最初のパケットは、195.81.202.68から172.31.136.85への普通のHTTPトラフィックです。このパケットのすぐあとに、172.31.136.85からゼロウィンドウパケットが送られています(図9-19)。

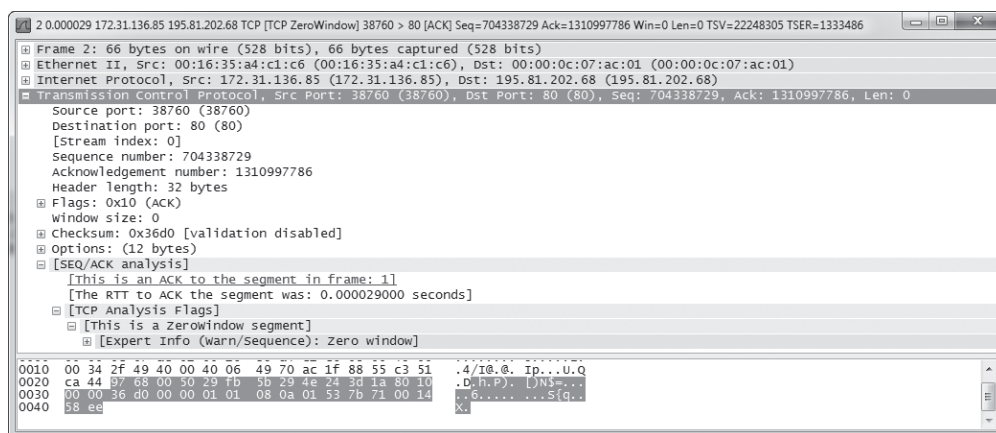


図9-19 データ転送を止めるゼロウィンドウパケット

これは図9-17のゼロウィンドウパケットとよく似ていますが、結果はまったく違います。172.31.136.85の機器がWindow Updateパケットを送って通信再開を通知するのではなく、図9-20のようにキープアライブパケットが送られています。

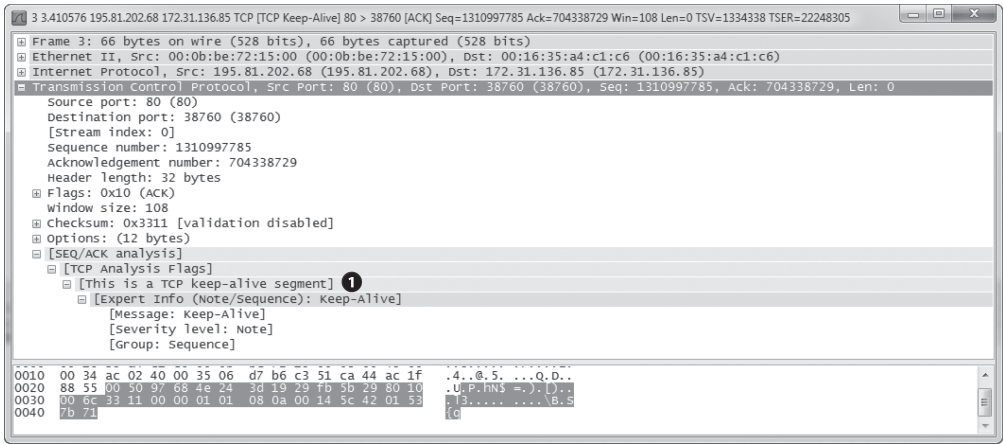


図9-20 キープアライブパケットはゼロウィンドウの機器とのコネクションを維持する

このパケットには、[Packet Details] ペインのTCPヘッダのSEQ/ACK Analysisセクションで、キープアライブという印が付いています①。[Time] カラムを見ると、最後のパケットが受信された3.4秒後に、このパケットが送られたことがわかります。図9-21のように、片方の機器がゼロウィンドウパケットを送り、もう一方がキープアライブパケットを送るという処理が、何回か続きます。

No.	Time ①	Source	Destination	Protocol	Info
2	0.000029	172.31.136.85	195.81.202.68	TCP	[TCP zerowindow] 38760 > 80 [ACK] Seq=704338729 Ack=1310997785 win=0 Len=0 TSV=22248305
3	3.410576	195.81.202.68	172.31.136.85	TCP	[TCP keep-alive] 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 win=0 Len=0 TSV=1334338
4	0.000031	172.31.136.85	195.81.202.68	TCP	[TCP zerowindow] 38760 > 80 [ACK] Seq=704338729 Ack=1310997785 win=0 Len=0 TSV=22249158
5	6.784127	195.81.202.68	172.31.136.85	TCP	[TCP keep-alive] 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 win=0 Len=0 TSV=1336035
6	0.000029	172.31.136.85	195.81.202.68	TCP	[TCP zerowindow] 38760 > 80 [ACK] Seq=704338729 Ack=1310997785 win=0 Len=0 TSV=22250854
7	13.536744	195.81.202.68	172.31.136.85	TCP	[TCP keep-alive] 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 win=0 Len=0 TSV=1339416
8	0.000047	172.31.136.85	195.81.202.68	TCP	[TCP zerowindow] 38760 > 80 [ACK] Seq=704338729 Ack=1310997785 win=0 Len=0 TSV=22254238

図9-21 ゼロウィンドウパケットとキープアライブパケットの送信が何度か繰り返される

キープアライブパケットは、3.4秒、6.8秒、13.5秒の間隔で送信されています①。通信機器で動作しているOSによっては、この処理がかなり長くなります。今回の場合、[Time] カラムの値を加算していくと、コネクションがほぼ25秒間停止しています。ドメインコントローラでの認証や、インターネットからファイルをダウンロードしようとしているときに、25秒もの遅れがあったらどうでしょう。我慢できるわけがありません。

9.3 TCPエラー制御とフロー制御パケット

TCP再転送、重複ACK、スライディングウィンドウについて考察し、遅延に関するトラブルシューティングを行うときの注意点をいくつか挙げておきます。

再転送パケット

再転送が行われるのは、クライアントが送ったデータをサーバが受信していないことにクライアントが気づいたからです。つまりパケット解析する場所によっては、再転送に気づかな

いわけです。たとえばサーバ上でデータをキャプチャしている場合、クライアントから送信、再送信されたパケットは受け取れないので、再転送パケットを目にすることもできずに立ち往生してしまいます。サーバ側でパケット消失の被害者になっているかもしれないと思ったら、クライアント側でトラフィックをキャプチャできないか考えてみましょう。そうすれば再転送パケットの存在が確認できます。

重複ACKパケット

筆者は、重複ACKを再転送の逆のようなものだと考えています。というのも、クライアントからのパケットが途中で消失したことをサーバが検出したときに送られるからです。重複ACKは大半の場合、どちら側でトラフィックをキャプチャしたときでも確認できます。これは、受け取ったパケットがシーケンスから外れているときに発生するものであることを思い出してください。たとえば、サーバが3個のパケットのうち1番目と3番目だけを受け取った場合は重複ACKが送られ、クライアントは2番目のパケットを高速再送します。サーバが1番目と3番目を受け取っている場合、2番目のパケットが届かないのは一時的な原因による可能性が高く、通常重複ACKは無事送受信されます。もちろんこのシナリオが常に正しいとは限りません。サーバ側でのパケット消失が疑われ、重複ACKが見当たらない場合、クライアント側でのパケットキャプチャを検討してください。

ゼロウィンドウ状態とキープアライブパケット

スライディングウィンドウは、サーバでデータを受信、処理できなくなる状況と直接関係します。ウィンドウサイズが縮小したり、ゼロウィンドウ状態になったりするのは、サーバで何か問題が起きているということなので、どちらかの事態が生じたら、原因を調べるべきです。通常はサーバとクライアントの両側で、Window Update パケットが確認できるはずです。

9.4 高遅延の原因を突き止める

パケット消失が遅延の原因でない場合があります。2つの機器間の通信が遅くても、TCP再送や重複ACKが見つからないことがあるのです。このような場合、高遅延の原因を見つける別のテクニックが必要となります。

高遅延の原因を突き止める効果的な方法のひとつは、最初のハンドシェイクと、そのあとに送られるパケットをいくつか調べることです。仮にクライアントがWebサーバとコネクションを確立していて、そのWebサーバでホストされているサイトをブラウザしようとしたとしましょう。ここで注意が必要なのは、TCPハンドシェイク、最初のHTTP GETリクエスト、GETリクエストに対するACK、サーバがクライアントに送った最初のデータパケットから構成される、最初の6個のパケットです。



この項を理解するには、Wiresharkの時間表示形式を適切に設定する必要があります。
[View] → [Time Display Format] → [Seconds Since Previous Displayed Packet] を選択します。

9.4.1 正常な通信 latency1.pcap

ネットワークベースラインの詳細については、本章の以降の部分で説明します。ここでは、高遅延の状況と比較するために、正常時の通信のベースラインが必要だということだけを理解しておいてください。このあとのサンプルでは、ファイルlatency1.pcapを使用します。TCPハンドシェイクとHTTP通信についてはもう説明したので繰り返しません。また [Packet Details] ペインも見ません。ここでは [Time] カラムだけに焦点を絞ります (図9-22)。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	1606 > 80 [SYN] Seq=2082691767 win=8192 Len=0 MSS=1460 WS=2
2	0.030107	74.125.95.104	172.16.16.128	TCP	80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 win=5720 Len=0 MSS=1406 WS=6
3	0.000075	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 win=4218 Len=0
4	0.000066	172.16.16.128	74.125.95.104	HTTP	GET / HTTP/1.1
5	0.048778	74.125.95.104	172.16.16.128	TCP	80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 win=109 Len=0
6	0.022176	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]

図9-22 このトラフィックはすぐに始まっており正常だと考えられる

この通信シーケンスは非常に速く、全体で0.1秒もかかっていません。
以降のキャプチャファイルは、同じトラフィックではありますが、パケットの送受信される時刻が違います。

9.4.2 通信の遅延：回線遅延 latency2.pcap

今度はファイルlatency2.pcapを見てみましょう。図9-23からわかるように、2つのファイルのパケットは、時刻を除けばすべて同じです。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	1606 > 80 [SYN] Seq=2082691767 win=8192 Len=0 MSS=1460 WS=2
2	0.878530	74.125.95.104	172.16.16.128	TCP	80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 win=5720 Len=0 MSS=1406 WS=6
3	0.016604	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 win=4218 Len=0
4	0.000335	172.16.16.128	74.125.95.104	HTTP	GET / HTTP/1.1
5	1.155228	74.125.95.104	172.16.16.128	TCP	80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 win=109 Len=0
6	0.015866	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]

図9-23 2番目と5番目のパケットが高遅延を示している

6個のパケットを見ていくとすぐ、最初の遅延の兆候に気づきます。クライアント (172.16.16.128) がTCPハンドシェイクを開始するために最初のSYNパケットを送ってから、サーバ (74.125.95.104) からのSYN/ACKを受け取るまでに、0.87秒の遅れが見られます。これはクライアントとサーバ間にある通信機器によって引き起こされる回線遅延の最初の兆しです。
転送されるパケットの性格上、これが回線の遅延であると判断できます。サーバがSYNパケットを受け取ってから、応答を送るまでに必要な処理はごくわずかです。というのも、トランスポート層より上位が処理にかかわることはないからです。サーバのトラフィックがかなり多いとしても、通常はSYNパケットに対してSYN/ACKパケットにより迅速に応答することが可能です。つまりサーバは高遅延の原因ではありません。
クライアントも遅延の原因ではありません。この時点ではSYN/ACKパケットを受信する以上の処理が行われていないからです。
クライアントとサーバの両方に問題がないとすると、最初の2個のパケットの間に、遅延の原因が

あるということになります。

引き続き見ていくと、3ウェイハンドシェイクを完了するACKパケット送信はすぐに行われ、クライアントからHTTP GETリクエストが送られます。この2個のパケットを生成する処理は、SYN/ACKを受け取ったあとにクライアントでローカルに行われるので、クライアントの負荷が高くない限り、2個のパケットの送信は迅速に行われます。

5番目のパケットも、非常に遅延しています。最初のHTTP GETリクエストが送信されたあと、サーバから返却されたACKパケットが届くまでに1.15秒かかっています。HTTP GETリクエストを受け取ると、サーバはデータ送信を開始する前にまずTCP ACKを送りますが、この処理もほとんど負担はかかりません。これも回線遅延の別の兆候です。

回線遅延が発生すると、最初のハンドシェイクでのSYN/ACKだけでなく、ほかのACKパケットでも遅延の兆候が伺えます。この情報からはネットワークの高遅延の原因はわかりませんが、クライアントとサーバが原因でないことはわかるので、両者の間の通信機器が遅延の原因だとわかります。そこでファイアウォール、ルータ、プロキシなどを調べることで、犯人を見つけることができます。

9.4.3 通信の遅延：クライアントの遅延

latency3.pcap

次の遅延のシナリオは、ファイルlatency3.pcapに含まれています (図9-24)。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	1606 > 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=2
2	0.023790	74.125.95.104	172.16.16.128	TCP	80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 WS=6
3	0.014894	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 Win=4218 Len=0
4	1.345023	172.16.16.128	74.125.95.104	HTTP	GET / HTTP/1.1
5	0.046121	74.125.95.104	172.16.16.128	TCP	80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=109 Len=0
6	0.016182	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]

図9-24 最初のHTTP GETパケットが遅延している

このキャプチャは迅速なTCPハンドシェイクで正常に始まり、ここでは遅延の兆しは見られません。ハンドシェイクが完了したあと、4番目のパケットであるHTTP GETリクエストが送られるまでは、すべてが正常のようです。ところがこのパケットで、1.34秒の遅延が発生しました。

遅延の原因を見極めるには、3番目と4番目のパケットの間で何が起きたかを調べなければなりません。3番目のパケットはクライアントからサーバに送られたTCPハンドシェイクの最後のACKパケットで、4番目のパケットはクライアントからサーバに送られたGETリクエストです。両者に共通しているのは、どちらもクライアントから送信されたもので、サーバとは無関係である点です。両方ともクライアントが起点となっているので、ACKが送られたすぐあとに、GETリクエストが行われるべきです。

残念ながら、ACKからGETへの遷移は迅速にいきませんでした。GETパケットの生成と送信にはアプリケーション層での処理が必要ですので、この処理の遅れというのは、クライアント上の処理が追いついていないという意味になります。つまりこの通信の高遅延の原因は、クライアントにあるのです。

9.4.4 通信の遅延：サーバの遅延

latency4.pcap

最後の遅延のシナリオはサーバの遅延で、ファイル latency4.pcap を使います (図9-25)。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	1606 > 80 [SYN] Seq=2082691767 win=8192 Len=0 MSS=1460 WS=2
2	0.018583	74.125.95.104	172.16.16.128	TCP	80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 win=5720 Len=0 MSS=1406 WS=6
3	0.016197	172.16.16.128	74.125.95.104	TCP	1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 win=4218 Len=0
4	0.000172	172.16.16.128	74.125.95.104	HTTP	GET / HTTP/1.1
5	0.047936	74.125.95.104	172.16.16.128	TCP	80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=109 Len=0
6	0.982983	74.125.95.104	172.16.16.128	TCP	[TCP segment of a reassembled PDU]

図9-25 最終パケットまで高遅延の兆候が見えない

2つの機器間のTCPハンドシェイクはスムーズかつ迅速に行われ、うまくいっています。最初のGETリクエストとその応答のACKパケットの送信も素早く行われました。このファイルでは最終パケットまで、高遅延の兆候が見えないのです。

この6番目のパケットは、クライアントからのGETリクエストに応じてサーバから送られた最初のHTTPデータパケットですが、サーバがGETリクエストに対して送ったTCP ACKの到着から0.98秒遅れています。5番目から6番目のパケットへの遷移は、ハンドシェイクでのACKからGETリクエストへの移行と非常によく似ています。しかしここでは、サーバが焦点となります。

5番目のパケットは、クライアントから受け取ったGETリクエストに対するサーバのACKです。サーバはACKパケットを送信したらすぐ、データ送信を開始しなければなりません。このパケットの送信データ取得、パケット組み立て、送信はHTTPプロトコルによって行われます。これはアプリケーション層プロトコルなので、サーバによる処理がある程度必要となります。したがってこのパケットの遅延は、サーバ上のデータ処理が追いついていないことを意味しており、高遅延の理由はサーバだということになります。

9.4.5 遅延を見つけるフレームワーク

6個のパケットを使って、ネットワーク高遅延の原因を探しました。これらのシナリオは少々複雑に見えたかもしれませんが、図9-26が実際の遅延トラブルを解決するときに役立つはずです。こうした基本はどんなTCP通信にも当てはまります。



UDP遅延についてはほとんど触れていません。UDPは高速ですが信頼性が低く、遅延を検出し回復する機能を内蔵していないからです。その代わりに、データ配送の信頼性を高めるために用いられるアプリケーション層プロトコル (やICMP) が、その役割を担っています。

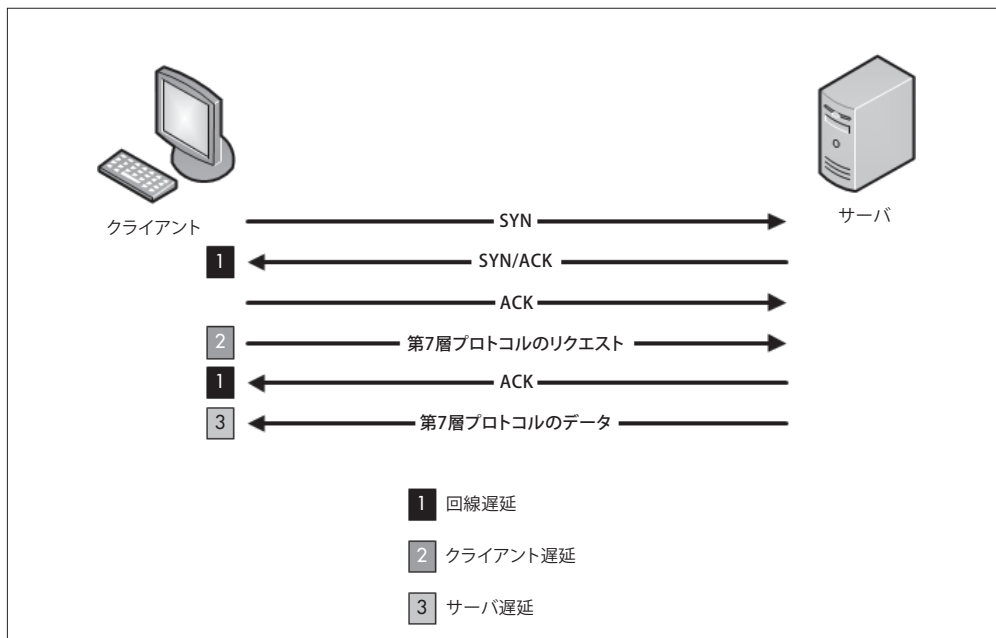


図9-26 実際の遅延トラブルの解決に役立つ図

9.5 ネットワークベースラインの確立

何もかもがうまくいかない場合、ネットワークベースラインが、ネットワークの遅延を解決するための重要なデータとなります。ここではネットワークベースラインを、ネットワーク上のさまざまな位置で収集されたトラフィックのサンプルであり、「正常な」ネットワークトラフィックとみなされるものであると定義します。ネットワークベースラインを確立する目的は、ネットワークや通信機器が正しく機能していないときに、比較対象とするためです。

たとえばネットワーク上のいくつかのクライアントが、ローカルのWebアプリケーションサーバへのログインに時間がかかるという苦情を言ってきたとしましょう。このトラフィックをキャプチャしてネットワークベースラインと比較すると、Webサーバは正常に応答しているのに、Webアプリケーションに組み込まれた外部コンテンツによる外部DNSリクエストの速度が、正常時より2倍も遅いことがわかるかもしれません。

ネットワークベースラインと比較しなくても外部DNSサーバが遅いと気づくかもしれませんが、少しの差異だとなかなかそうもいきません。10個のDNSクエリの処理が正常時より0.1秒ずつ遅いの、1個のクエリに正常時より1秒以上かかるのと同じくらいの問題ですが、ネットワークベースラインがなかったら、前者を検知するのはかなり困難です。

同じネットワークは存在しないため、ネットワークベースラインのコンポーネントもかなり違ってき

ます。次の項ではネットワークベースラインのコンポーネント例を挙げます。すべてが自分のネットワークに該当する場合もあれば、ほとんど当てはまらない場合もあるでしょう。いずれにせよ、各コンポーネントはベースラインの3つの基本カテゴリであるサイト、ホスト、アプリケーション上で計測することが可能です。

9.5.1 サイトのベースライン

サイトのベースラインの目的は、ネットワーク上の物理的なサイトのトラフィックに関する俯瞰的なスナップショットを得ることです。WANのすべてのセグメントで実行するのが理想です。

このベースラインに含まれるコンポーネントは次のようになります。

使用しているプロトコル

ネットワークのエッジ（ルータやファイアウォール）でセグメントのすべての通信機器からのトラフィックをキャプチャする際に、メニューから [Statistics] → [Protocol Hierarchy] を選択して [Protocol Hierarchy Statistics] ダイアログを開き、すべての通信機器からのトラフィックを参照します。のちほどこれと照らし合わせることで、正常時に見えるプロトコルは存在しているか、あるいは新しいプロトコルが含まれていないかを確認できます。特定プロトコルのトラフィックが大幅に増えていないかどうかともわかります。

ブロードキャストトラフィック

ネットワークセグメント上のすべてのブロードキャストトラフィックが含まれます。サイト内のどこでキャプチャしてもブロードキャストトラフィックはキャプチャできます。正常時はどういった機器が大量のブロードキャストトラフィックを送っているのかが確認でき、ブロードキャストの量が多すぎたり少なすぎたりしないかがすぐにわかります。

認証シーケンス

Active Directory、Webアプリケーション、組織固有のソフトなど、あらゆるサービスに対するさまざまなクライアントからの認証処理のトラフィックが含まれます。一般に認証には時間がかかります。ベースラインによって、認証が通信遅延の原因かどうか判断できます。

データ転送レート

ネットワーク上のあるサイトからさまざまな別のサイトへの大量データ転送の測定結果から構成されます。キャプチャの概要とWiresharkのグラフ機能を利用して、転送率とコネクションとの整合性が確認できます。おそらくこれがサイトベースラインの中でもっとも重要なものでしょう。通信が遅く感じられる場合、ベースラインと同じデータの転送を実行し、その結果を比較してください。そうすれば通信が本当に遅いのかどうか分かり、またどこで遅延トラブルが発生しているのかを探る手がかりにもなります。

9.5.2 ホストベースライン

ホストベースラインといっても、すべてのホストのベースラインを確立する必要はありません。トラフィックが非常に多い、またはミッションクリティカルなサーバでのみ行えば十分です。あるサーバが遅いと管理部門から怒りの電話がかかってくるようであれば、そのサーバのベースラインは確立しておきましょう。

ホストベースラインのコンポーネントは以下のとおりです。

使用しているプロトコル

[Protocol Hierarchy Statistics] ダイアログを使って、ホストからのトラフィックをキャプチャします。のちほどこれと照らし合わせることで、正常時に見えるプロトコルが存在しているか、あるいは新しいプロトコルが含まれていないかを確認できます。特定プロトコルのトラフィックが大幅に増えていないのかもわかります。

アイドル、ビジー時のトラフィック

ピーク時およびオフピーク時の通常トラフィックのキャプチャで構成されます。時間帯によるコネクション数や使用帯域を把握しておけば、遅延がユーザー負荷のせいなのか、それともほかに問題があるのかがわかります。

起動／シャットダウン

このベースラインを得るには、ホストの起動とシャットダウンの際に生じるトラフィックをキャプチャする必要があります。機器が起動しない、シャットダウンしない、あるいは起動やシャットダウンの際の速度が極端に遅い場合、このベースラインと照合し、遅延がネットワーク関連かどうかを判断できます。

認証シーケンス

ホスト上で動作しているあらゆるサービスの認証処理のトラフィックをキャプチャする必要があります。一般に認証には時間がかかります。ベースラインによって、認証が通信遅延の原因かどうか判断できます。

関連／依存

このホストがどのホストに依存しているか（あるいは、このホストにどのホストが依存しているか）を知るために、長時間のキャプチャを行います。メニューから [Statistics] → [Conversations] を選択して [Conversations] ダイアログを開き、関連性や依存性を確認します。これによりたとえば、WebサーバがSQL Serverサーバに依存していたりすることがわかります。ホスト間の依存関係は、常に意識しているものではないので、これらの確認にホストベースラインが役立ちます。これによって、ホストが故障や高負荷のために適切に動作していないかどうかを確認できます。

9.5.3 アプリケーションベースライン

ネットワークベースラインの最後のカテゴリは、アプリケーションベースラインです。これは業務上不可欠なネットワークベースのアプリケーションでは必ず取得すべきです。

アプリケーションベースラインのコンポーネントは以下のとおりです。

使用しているプロトコル

このベースラインでも、Wiresharkの[Protocol Hierarchy Statistics]ダイアログを使いますが、今回はアプリケーションを実行しているホストからのトラフィックをキャプチャします。のちほどこれと比較することで、該当のアプリケーションが依存するプロトコルが、正常に機能しているかどうかわかります。

起動／シャットダウン

このベースラインを得るには、アプリケーションの起動とシャットダウンの際に生じるトラフィックをキャプチャする必要があります。アプリケーションが起動しない、あるいは起動やシャットダウンの際の速度が極端に遅い場合、このベースラインと照合し、原因を突き止めることができます。

関連／依存

このアプリケーションが依存しているホストやアプリケーションを知るために、長時間のキャプチャを行い、[Conversations]ダイアログを使って確認します。アプリケーション間の依存関係は、常に意識しているものではないので、これらの確認にこのベースラインが役立ちます。これによって、アプリケーションが障害や高負荷のために適切に動作していないかどうかを確認できます。

データ転送レート

キャプチャの概要とWiresharkのグラフ機能を利用して、ベースラインを作成した際の正常時のアプリケーションサーバへの転送率とコネクションとの整合性が確認できます。アプリケーションが遅いという報告があれば、このベースラインを使って、遅延の原因がアプリケーションの負荷が高すぎるためか、ユーザー側の負荷が原因かを判断できます。

9.5.4 ベースラインについての追記

ネットワークベースラインを作成する際に、心に留めておくべき点をいくつか追加しておきます。

- ベースラインを作成する場合、最低3回ずつは作成しましょう。トラフィックの少ない時間帯（早朝）、トラフィックの多い時間帯（午後）、トラフィックがほとんどない時間帯（深夜）といった具合です。
- 可能なら、ベースラインの対象となるホストで、直接キャプチャするのは避けましょう。トラフィックが多い時間帯だと、キャプチャが負荷を増やしたり、パフォーマンスを低下させたりし

て、結果としてパケットの消失を招き、作成したベースラインが無意味になってしまう場合があります。

- ベースラインにはネットワークに関する秘匿情報が含まれているので、セキュリティに注意しましょう。許可された人々だけがアクセスできる安全な場所に保存することです。同時にすぐ利用できるよう、手元に置いておきましょう。USBドライブか、暗号化したパーティションに入れておくべきです。
- ベースライン関連の.pcap ファイルをまとめ、関連や平均データ転送レートなど、よく参照する値の「カンベ」を作っておきましょう。

9.6 まとめ

この章では、ネットワーク遅延のトラブルシューティングに焦点を当てました。TCPの問題を検出し、リカバリするための便利で、信頼性の高い方法を説明し、ネットワーク通信の高遅延の原因を探る方法を実演し、ネットワークベースラインの重要性とそのコンポーネントについて説明しました。ここで説明したテクニックと、Wiresharkのグラフおよび解析機能（5章で説明）を使えば、ネットワークが遅いという苦情の電話を受けても、対応することができるはずです。

10章

セキュリティとパケット解析

本書の大半はパケット解析を利用したネットワークのトラブルシューティングに割かれていますが、現実にはかなりのパケット解析がセキュリティ対応として行われています。これは不正侵入者を防ぐためにネットワークトラフィックを確認するアナリストや、侵入されたホストでのマルウェア感染の範囲を突き止めるフォレンジック調査官の仕事だったりします。セキュリティ対応のパケット解析は大きなテーマであり、本が一冊書けるほどです。ここではセキュリティに的を絞ったパケット解析のさわりだけを紹介します。

この章ではセキュリティ関係者の立場で、ネットワークレベルでのシステム侵入をさまざまな角度から見ていき、まずはネットワークの偵察、悪意あるトラフィックのリダイレクト、システムへの侵入などについて説明します。次に侵入アナリストとなって、侵入検知システム (IDS) からの警告をもとに、トラフィックを細かく分析します。本章を読めば、セキュリティを担当していなくても、ネットワークセキュリティの本質が理解できるでしょう。

10.1 偵察

攻撃者が最初に行うのは、標的とするシステムの徹底的な調査です。この段階を一般に「フットプリンティング」と呼び、標的とする企業のWebサイトやGoogleでの検索など、公開されているさまざまな情報を利用して行われます。調査が完了すると、攻撃者は開いているポートや実行されているサービスを見つけるため、標的のIPアドレス（またはドメイン名）のスキャンを開始します。

攻撃者はスキャンによって、標的が稼働しているか、アクセスできるかを判断します。銀行強盗が、メインストリート123番にある大手銀行での強盗を計画していたとしましょう。強盗が念入りの強盗計画を立てて現地へ行ってみたところ、銀行はヴァインストリート555番へ引っ越しているかもしれません。もっと悪いケースを考えると、昼間の営業時間中に徒歩で銀行に押し入って、金庫から盗むつもりでいたのに、銀行が休業日だったとしたらどうでしょう。標的が稼働していて、アクセスできるかを確認するのが、最初に越えなければならないハードルなのです。

スキャンによるもうひとつの重要な成果は、標的のどのポートが待ち受けしているかわかることです。銀行強盗のたとえに戻って、強盗が銀行施設の見取り図の知識がまったくないままに、銀行へ押

し入ろうとしたらどうでしょう。セキュリティの弱点を知らないわけですから、侵入する方法も思いつけないでしょう。

この項では、ホスト、開いているポート、ネットワークの脆弱性を確認するのに使われる、一般的なスキャン手法についていくつか説明します。



ここまでは、送信者と受信者、あるいはクライアントとサーバといった呼称でコネクションの両端について言及してきましたが、本章では攻撃者および被害者という呼称を用います。

10.1.1 SYNスキャン

`synscan.pcap`

システムに対して最初に行われるのがTCP SYNスキャンで、これはステルススキャンまたはハーフオープンスキャンとも呼ばれます。SYNスキャンがもっともよく使われるのには理由があります。

- 非常に早く信頼性が高い。
- TCPスタックの実装にかかわらず、すべてのプラットフォームで適切に機能する。
- ほかのスキャン手法よりもノイズが少ない。

TCP SYNスキャンは、標的のどのポートが開いているかを判断するのに3ウェイハンドシェイク処理を用います。攻撃者はポートと正常な通信を確立するふりをして、被害者のポートにTCP SYNパケットを送ります。被害者がこのパケットを受け取ると、図10-1のようなことが起こります。

被害者のホストのサービスがSYNパケットを受け取ったポートで待ち受けていると、TCPハンドシェイクの第2段階として、攻撃者にTCP SYN/ACKパケットをレスポンスとして送ります。すると攻撃者はそのポートが開いていて、サービスが待ち受けしていることがわかります。通常であれば、ハンドシェイクを完了させるために最終のTCP ACKパケットが送られますが、この場合攻撃者はこれ以上ホストと通信する気がないため、TCPハンドシェイクを完了させません。

スキャンしたポートでサービスが待ち受けしていない場合、攻撃者はSYN/ACKパケットを受け取りません。被害者のOSの設定によっては、攻撃者はポートが閉じていることを意味するRSTパケットを応答として受け取ります。あるいは何の応答もない場合もあります。これはファイアウォールやホスト自身によって、ポートにフィルタがかけられているという意味です。あるいは送信の途中で、レスポンスが消失しただけかもしれません。一般にはポートが閉じているということですが、決定的な証拠ではありません。

ファイルsynscan.pcapは、Nmapツールを利用したSYNスキャンの好例です。Nmapは、Fyodorが開発した堅牢なネットワークスキャンアプリケーションで、どんなスキャンでも実行することができます。Nmapは<http://www.nmap.com/download.html>から無料で入手可能です。

サンプルのキャプチャには約2000個のパケットが入っていますので、手頃なサイズでしょう。スキャンの範囲を確認する最良の方法のひとつは、[Conversations] ダイアログを見ることです (図10-2)。

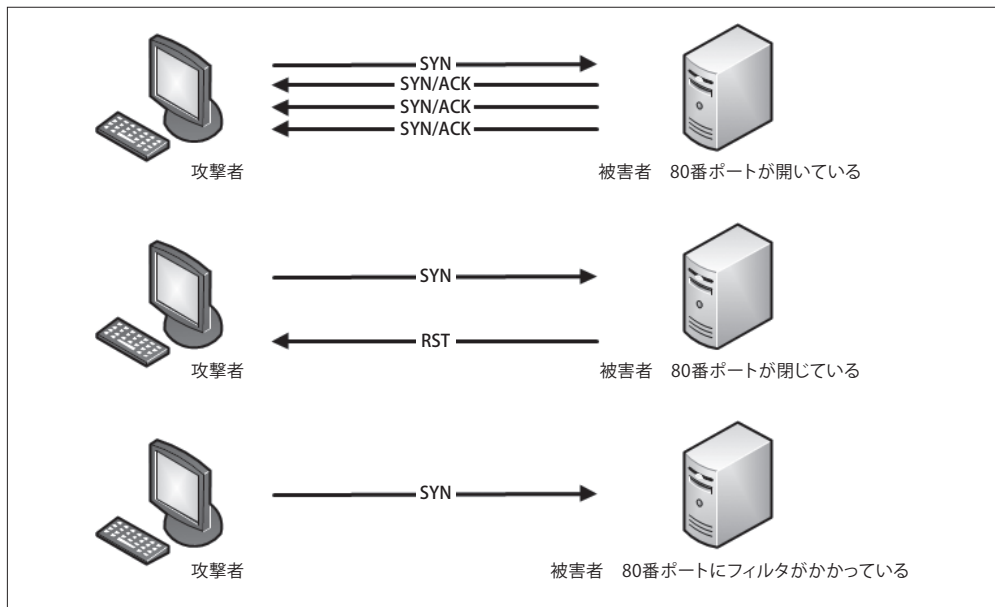


図10-1 TCP SYNスキャンによって起こる可能性のあること

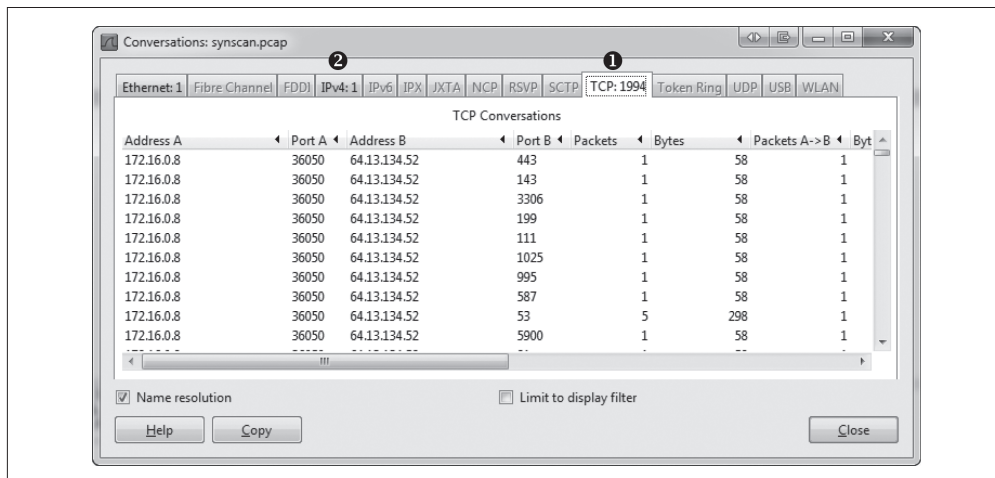


図10-2 ささまざまなTCP通信が行われていることを示す [Conversations] ダイアログ

ここには攻撃者（172.16.0.8）と被害者（63.13.134.52）間のIPv4の対話が1つしかありません①が、この2つのホスト間に、TCPの対話が1994あることがわかります②。

スキャンは非常に早く行われるため、それぞれのSYNパケットの応答を見つけるのに、キャプチャファイルをスクロールするのはあまり良い方法とは言えません。パケットへの応答を受け取る前に、

さらにパケットが送られているかもしれないからです。幸いなことに、フィルタを作成することで適切なトラフィックを確認することができます。

10.1.1.1 SYNスキャンでのフィルタの使用

フィルタの一例として、被害者の443番ポート (HTTPS) に送られた最初のパケット、SYNパケットを取り上げます。このパケットに対する応答を見るため、443番ポートを出入りするすべてのトラフィックを表示するフィルタを作成します。迅速に行う方法は次のとおりです。

1. キャプチャファイルの最初のパケットを選択します。
2. [Packet Details] ペインのTCPヘッダを展開します。
3. [Destination Port] フィールドを右クリックし、[Prepare as Filter] を選択し、[Selected] をクリックします。
4. これで443番ポートを宛先とするすべてのパケットからなるフィルタが、フィルタダイアログに表示されます。443番ポートを送信元とするパケットも表示したいので、画面の上部にあるフィルタダイアログをクリックし、「dst」を消去します。

このフィルタにより攻撃者から被害者へ送られた2個のTCP SYNパケットが表示されます (図10-3)。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.0.8	64.13.134.52	TCP	36050 > 443 [SYN] Seq=3713172248 win=3072 Len=0 MSS=1460
32	0.000065	172.16.0.8	64.13.134.52	TCP	36051 > 443 [SYN] Seq=3713237785 win=2048 Len=0 MSS=1460

図10-3 SYNパケットによる2回のコネクション確立の試行

どちらのパケットにも応答はないので、被害者のホストか中間にある機器などでフィルタされたか、あるいはポートが閉じている可能性があります。443番ポートに対するスキャンだけでは判断できません。

同じ手法をほかのパケットでも試して、違う結果が出るかどうかを見てみましょう。まずフィルタの横の [Clear] ボタンをクリックし、先ほど作成したフィルタを削除します。次にリストから9番目のパケットを選びます。これはポート53番、通常DNSに割り当てられているポートへ送られたSYNパケットです。先ほどと同様に、このポートに向かうパケットのフィルタを作り、「dst」を消去して、TCPポート53番を通過するすべてのパケットを抽出します。このフィルタを適用したら、5つのパケットが確認できるはずです (図10-4)。

No.	Time	Source	Destination	Protocol	Info
9	0.000052	172.16.0.8	64.13.134.52	TCP	36050 > 53 [SYN] Seq=3713172248 win=3072 Len=0 MSS=1460
11	0.001832	64.13.134.52	172.16.0.8	TCP	53 > 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 win=5840 Len=0 MSS=1380
529	0.057126	64.13.134.52	172.16.0.8	TCP	53 > 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 win=5840 Len=0 MSS=1380
2006	3.930109	64.13.134.52	172.16.0.8	TCP	53 > 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 win=5840 Len=0 MSS=1380
2009	10.029025	64.13.134.52	172.16.0.8	TCP	53 > 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 win=5840 Len=0 MSS=1380

図10-4 ポートが開いていることを示す5個のパケット

最初のパケットは、キャプチャの先頭のほうにある選択したSYNパケットです。2番目は被害者からの応答で、3ウェイハンドシェイクの確立を行う際に返却されるべきTCP SYN/ACKパケットです。

通常であれば、次のパケットは最初にSYNを送ったホストからのACKであるはずですが、ここでは、攻撃者はコネクションを確立させないので、応答を送りません。その結果、被害者はSYN/ACKを3回ほど再送します。53番ポートでの接続を試みるとSYN/ACK応答が戻ってきたので、このポートでサービスが待ち受けしていると考えてよいでしょう。

この作業を13番目のパケットでもう一度繰り返してみましょう。これは113番ポート、IRCでの識別や認証サービスに使われるIdentプロトコルに割り当てられているポートへ送られたSYNパケットです。このポートに同じタイプのフィルタを適用すると、図10-5のように4つのパケットが表示されます。

No.	Time	Source	Destination	Protocol	Info
13	0.000070	172.16.0.8	64.13.134.52	TCP	36050 > 113 [SYN] Seq=3713172248 win=4096 Len=0 MSS=1460
14	0.061491	64.13.134.52	172.16.0.8	TCP	113 > 36050 [RST, ACK] Seq=2462244745 Ack=3713172249 win=0 Len=0
530	0.006942	172.16.0.8	64.13.134.52	TCP	36061 > 113 [SYN] Seq=3696394776 win=2048 Len=0 MSS=1460
571	0.000827	64.13.134.52	172.16.0.8	TCP	113 > 36061 [RST, ACK] Seq=1027049353 Ack=3696394777 win=0 Len=0

図10-5 SYNのあとにRSTが送信され、ポートが閉じていることを示している

最初のSYNパケットのあと、被害者からはすぐにRSTが送信されています。これはこのポートでコネクションを許可しておらず、サービスが稼働していないことを意味しています。

10.1.1.2 ポートの開閉を見極める

SYNスキャンに対する応答の違いを理解したら、今度はどのポートが開いていて、どれが閉じているのかを迅速に判断する方法を考えてみましょう。今度の答えも [Conversations] ダイアログにあります。このダイアログでは、パケット数によってTCP対話をソートし、[Packets] カラムを2回クリックすることで数値が大きい順に並べることができます (図10-6)。

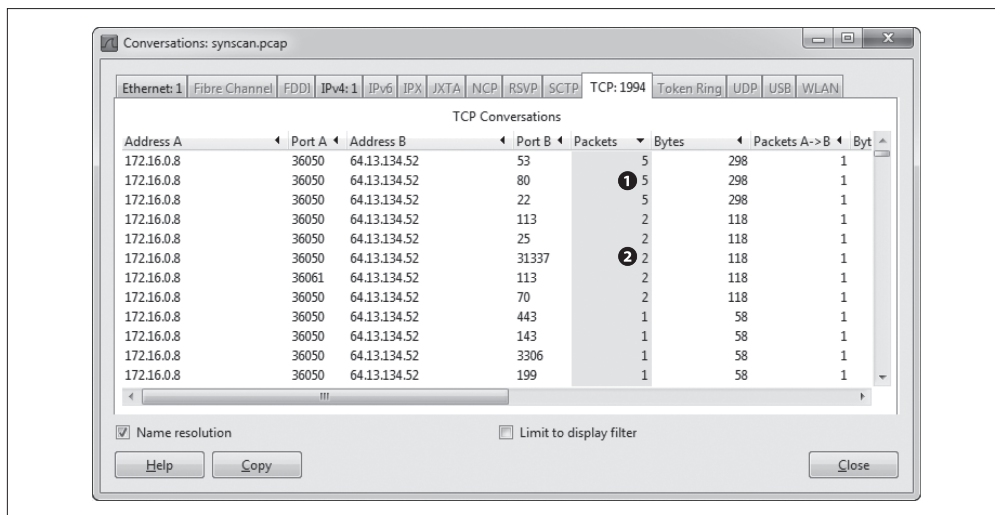


図10-6 [Conversations] ダイアログで開いているポートを探す

3つのポートは、対話に5つのパケットが表示されました❶。これら5つのパケットは最初のSYN、それに応答するSYN/ACK、被害者から再送されたSYN/ACKを表しているので、53番、80番、22番のポートが開いていることがわかります。

それ以外の5つのポートでは、2つのパケットしか表示されていません❷。1つ目は最初のSYN、そして2番目はRSTです。つまり113番、25番、31337番、113番、70番は閉じています。

残りのエントリには1つしかパケットが存在しないので、被害者のホストが最初のSYNに応答しなかったこととなります。したがって残りのポートは閉じていると考えられますが、定かではありません。

10.1.2 OSフィンガープリント

攻撃者は標的のOSを知るために相当な努力を行います。OSがわかれば適切な攻撃が仕掛けられるからです。またOSを特定することで重要なファイルやディレクトリがどこにあるかがわかるので、システムにアクセスしやすくなります。

OSフィンガープリントは、実際にそのシステムにアクセスすることなく、システム上で実行されているOSを特定するのに用いられるテクニックの総称です。OSフィンガープリントには、パッシブとアクティブの2種類があります。

10.1.2.1 パッシブフィンガープリント

passiveosfingerprinting.pcap

パッシブフィンガープリントでは、標的から送られたパケット内の特定のフィールドを調査することによって、OSを判断します。この方法がパッシブ（受動的）とされるのは、標的が送信するパケットをチェックするだけで、自らはパケットを送信しないためです。隠密に実行できるため、攻撃者にとっては理想的な方法です。

そうはいっても、送られてきたパケットを調べるだけで、どうやって標的のOSを特定できるのでしょうか。実はこれは非常に簡単で、RFCで定義されたプロトコルの仕様のおかげで可能になっているのです。TCP、UDP、IPヘッダにはいくつもフィールドがありますが、デフォルト値が設定されていません。つまりTCP/IPスタックを実装する場合、各OSはこれらのフィールドのデフォルト値を定義する必要があります。表10-1は一般的なフィールドと、さまざまなOSでのデフォルト値を示しています。

ファイルpassiveosfingerprinting.pcapに含まれている2つのパケットが良いサンプルです。どちらも80番ポートに送られたTCP SYNパケットですが、違うホストから送信されています。このパケットに含まれる値を表10-1に照合するだけで、それぞれのホストのOSが特定できるはずです。各パケットの詳細は図10-7に示しました。

表10-1 一般的なパッシブフィンガープリントの値

プロトコル ヘッダ	フィールド	デフォルト値	OS
IP	初期TTL	64	Nmap, BSD, Mac OS X, Linux
		128	Novell, Windows
		255	Cisco IOS, Palm OS, Solaris
IP	フラグメント禁止 フラグ	設定	BSD, Mac OS X, Linux, Novell, Windows, Palm OS, Solaris
		不設定	Nmap, Cisco IOS
TCP	最大セグメントサ イズ	0	Nmap
		1440	Windows, Novell
		1460	BSD, Mac OS X, Linux, Solaris
TCP	ウィンドウサイズ	1024-4096	Nmap
		65535	BSD, Mac OS X
		2920-5840	Linux
		16384	Novell
		4128	Cisco IOS
		24820	Solaris
TCP	SackOK	Variable	Windows
		設定	Linux, Windows, OpenBSD
		不設定	Nmap, FreeBSD, Mac OS X, Novell, Cisco IOS, Solaris

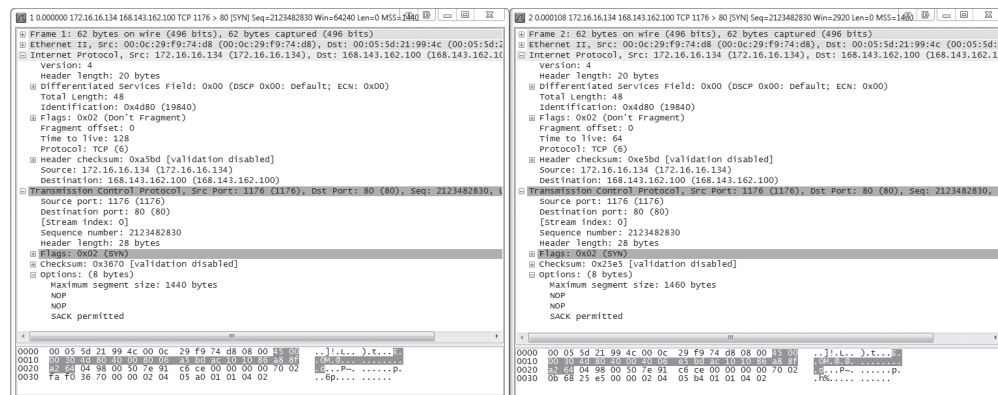


図10-7 パケットからOSが特定できる

表10-1を参照して、パケットごとに関連するフィールドの詳細(表10-2)を作りました。

表10-2 パケット別のOSフィンガープリント

プロトコルヘッダ	フィールド	パケット1の値	パケット2の値
IP	初期TTL	128	64
IP	フラグメント禁止フラグ	セット	セット
TCP	最大セグメントサイズ	1,440 バイト	1,460 バイト
TCP	ウィンドウサイズ	64,240 バイト	2,920 バイト
TCP	SackOK	セット	セット

これらの値によって、パケット1はWindows機器から、パケット2はLinux機器から送られたと結論づけることができます。

表10-1の一般的なパッシブフィンガープリントのフィールド一覧は、網羅的なものではありません。さまざまな要因により、値に狂いが生じる場合もあります。そのため、パッシブフィンガープリントから得られた結果に全面的に頼ることはできません。



OSフィンガープリント手法に使えるツールのひとつがp0fです。このツールはパケットキャプチャから各フィールドを解析し、可能性のあるOSを出力します。p0fのようなツールを使うと、OSの種類だけでなく、そのバージョンやパッチまでも特定できる場合があります。p0fは<http://lcamtuf.coredump.cx/p0f.shtml>からダウンロードできます。

10.1.2.2 アクティブフィンガープリント

activeosfingerprinting.pcap

受動的にトラフィックを監視するだけでは結果が得られない場合、より直接的なアプローチが必要になります。これをアクティブフィンガープリントと呼びます。攻撃者はOSを特定する応答を引き出すため、特別に作ったパケットを被害者に送信します。この手法では被害者と直接やり取りを行うので秘匿性はありませんが、かなりの効果があります。

ファイルactiveosfingerprinting.pcapには、Nmap スキャンユーティリティを使ったアクティブフィンガープリントスキャンのサンプルが含まれています。ファイル内のパケットは、OSを特定する応答を引き出すよう設計されたプローブを、Nmapが送った結果です。Nmapはこれらプローブへの応答を記録してフィンガープリントを作成し、データベースの値と比較してOSを特定します。



NmapがアクティブにOSを特定するために用いる手法はかなり複雑です。Nmapによるアクティブなフィンガープリントがどのように実行されているかをさらに学ぶには、Nmapの作者であるGordon "Fyodor" LyonによるNmapガイド「Nmap Network Scanning」を参照してください。

10.2 侵入

攻撃者はみな、侵入を目的としています。調査を行い、標的を偵察し、脆弱性を見つけ、標的にアクセスするための攻撃プログラムを準備します。この章の残りの部分では、Microsoftのちょっと前の脆弱性に対する攻撃プログラム、ARPキャッシュポイズニングによるトラフィックのリダイレクト、データの抜きだしを行うリモートアクセス型トロイの木馬といった、さまざまな侵入手法を見ていきましょう。

10.2.1 Operation Aurora

aurora.pcap

2010年1月、Operation Auroraは当時まだ知られていなかったInternet Explorerの脆弱性を攻撃しました。攻撃者はこの脆弱性を悪用し、Googleをはじめとする企業のマシンをroot権限で遠隔操作したのです。

ユーザーが脆弱性のあるInternet ExplorerでWebサイトを訪れるだけで、この悪意のあるコードが実行され、攻撃者は管理者権限でユーザーのホストにアクセスできるようになりました。これには悪意あるサイトへのリンクを含むメールを送信するスパイ型攻撃の手口が使われました。

スパイ型攻撃のメールは信用できる相手から送信されているように見せかけているため、成功率が高いのです。

Auroraは、標的とされたユーザーがスパイ型攻撃のメールのリンクをクリックしたときから始まっています。ファイルaurora.pcapに結果のパケットが入っています。

キャプチャは、被害者（192.168.100.206）と攻撃者（192.168.100.202）との3ウェイハンドシェイクで始まっています。最初の接続は80番ポートで行われているので、HTTPトラフィックと考えてよいでしょう。この推測は、4番目のパケットである、/infoのHTTP GETリクエスト①によって確認できます（図10-8）。

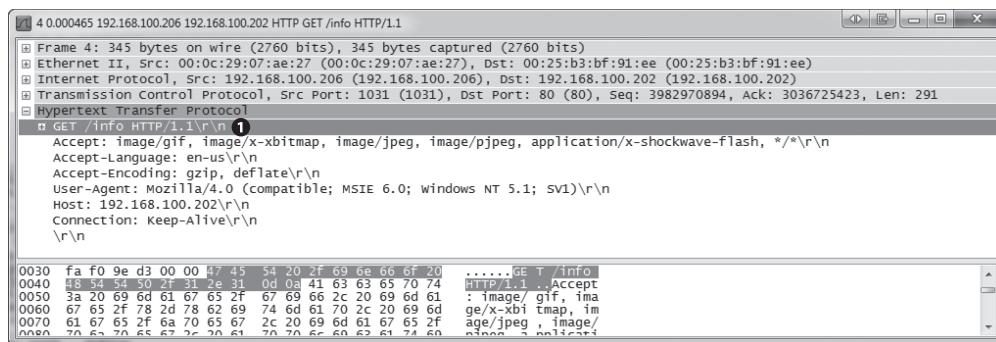


図10-8 被害者が/infoのGETリクエストを送信

攻撃者はGETリクエストにACKを返却した後、6番目のパケットでステータスコード302 (Moved Temporarily) を返却します。このステータスコードは一般に、別のページにリダイレクトする場合に用いられます。ステータスコード302❶とともに、[Location] フィールドで /info?rFfWELUjLJHpP がリダイレクト先として指定されています (図10-9)。

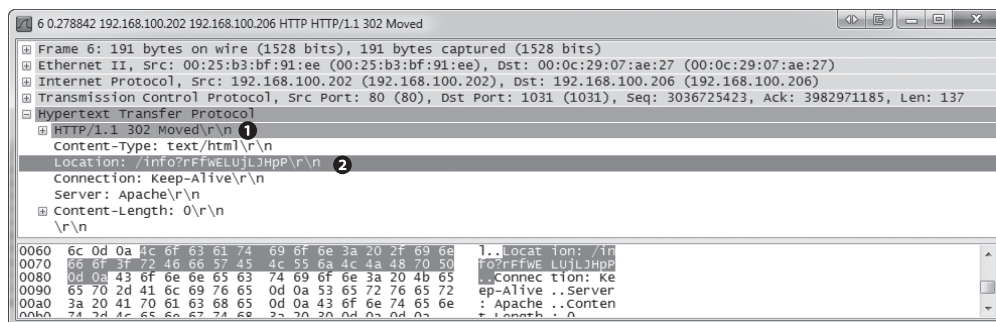


図10-9 このパケットでクライアントのブラウザがリダイレクトされる

HTTPステータスコード302のパケットを受け取ると、クライアントは7番目のパケットで、/info?rFfWELUjLJHpPというURLに別のGETリクエストを送り、8番目のパケットでACKを受け取ります。ACKに続くいくつかのパケットは、攻撃者から被害者へと転送されたデータです。このデータをよく見るため、ストリームの中の9番目のパケットを右クリックして、[Follow TCP Stream] を選択します。このストリームの出力には、最初のGETリクエスト、302のリダイレクト、2番目のGETリクエストが表示されています (図10-10)。

このあとからにわかには状況がおかしくなってきます。攻撃者は非常に奇妙なコンテンツでGETリクエストに応答しているのです。そのコンテンツの最初の部分を図10-11に示しました。

コンテンツは<script>タグに囲まれた、一連のランダムな数字と文字のように見えます❶。高水準スクリプト言語を使っていることを示すためにHTML内で使われるのが<script>タグで、このタグで囲まれた部分には通常さまざまなスクリプトのステートメントが含まれますので、このでたらめな文字と数字から検出を回避するためにエンコードされているらしいことがわかります。ここではこれが攻撃プログラムだとわかっているため、この意味不明なテキストには、パディングや脆弱性を攻撃するためのシェルコードが含まれていると考えてよいでしょう。

攻撃者から送信されたコンテンツの末尾の部分を図10-12に示します。エンコードされたテキストのあとに、ようやく読むことができるテキストを見つけました。高度なプログラミングの知識がなくても、このテキストがいくつかの変数から文字列を復元しようとしているもののように見えます。これが</script>タグで閉じられる前のテキストの最後の部分です。

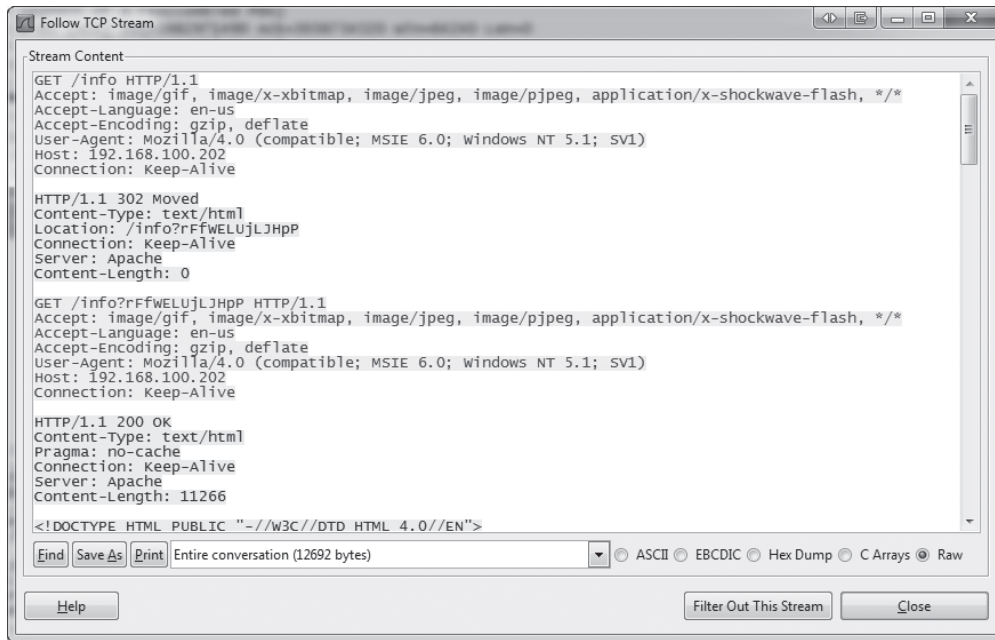


図10-10 クライアントに転送されたデータストリーム



図10-11 <script>タグで囲まれたコンテンツはエンコードされているように見える

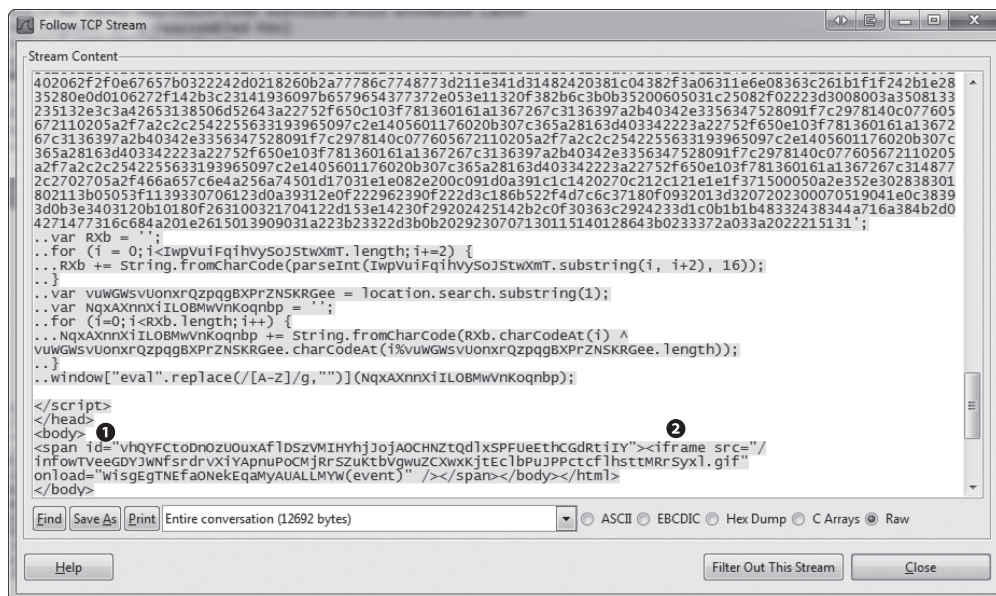


図10-12 サーバから送信されたコンテンツの末尾には、解読できるテキストと怪しいiframeが含まれている

攻撃者がクライアントに送信したデータの最後の部分は2つに分かれています。最初のセクションは^①で、2番目のセクションはタグにはさまれている<iframe src="/infowTVeeGDYJWNfSrdrvXiyApnuPoCmJRrSZuKtbVgwuzCXwxKjtcEclbPuJPPctcfIhsttMRrSyxl.gif" onload="WisgEgTNEfaONekEqMyAUALLMYW(event)" />^②です。繰り返しますが、このコンテンツは不自然に長い、解読不能なランダムな文字列であり、わざとわかりにくくしている可能性がある点からも、悪意ある活動の証拠と考えられます。

タグに囲まれたコードの部分は「iframe」と呼ばれ、攻撃者がHTMLページにコンテンツを埋め込むときに使うよくある手法です。<iframe>タグはユーザーが確認できないインラインフレームを作成します。ここでの<iframe>タグは奇妙な名前のGIFファイルを参照しています。被害者のブラウザがこのファイルへの参照を閲覧しようとする、図10-13のように、21番目のパケットでGETリクエストが送られ^①、すぐにこのGIFが送信されます^②。このGIFはおそらく、すでに被害者のマシンにダウンロードされている攻撃プログラムを起動するのに利用されると思われます。

No.	Time	Source	Destination	Protocol	Info
21	0.455107	192.168.100.206	192.168.100.202	HTTP	① GET /infowTVeeGDYJWNfSrdrvXiyApnuPoCmJRrSZuKtbVgwuzCXwxKjtcEclbPuJPPctcfIhsttMRrSyxl.gif HTTP/1.1 200 OK (GIF89a)
22	0.199959	192.168.100.202	192.168.100.206	TCP	80 > 1031 [ACK] Seq=3036736951 Ack=3982971911 Win=64518 Len=0
23	0.001166	192.168.100.202	192.168.100.206	HTTP	② GET /infowTVeeGDYJWNfSrdrvXiyApnuPoCmJRrSZuKtbVgwuzCXwxKjtcEclbPuJPPctcfIhsttMRrSyxl.gif HTTP/1.1 200 OK (GIF89a)
24	0.161592	192.168.100.206	192.168.100.202	TCP	1031 > 80 [ACK] Seq=3982971911 Ack=3036737098 Win=64093 Len=0

図10-13 被害者はiframeで指定されたGIFをリクエストし、ダウンロードする

このキャプチャで特に奇妙な部分が25番目のパケットで、被害者が攻撃者に4321番ポートでコネ

クションを開始しようとしています。この2番目の通信ストリームを [Packet Details] ペインで見てもほとんど情報がないので、再度TCPストリームを見て、やり取りされているデータをよく調べてみましょう。図10-14は [Follow TCP Stream] ダイアログの出力です。



図10-14 攻撃者が、コネクション上でコマンドシェルを使っている

このWindowsコマンドシェルが確認できたら、即刻警戒が必要です❶。このシェルは被害者から攻撃者のサーバへ送られたもので、攻撃者の侵入が成功し、ペイロードの展開にも成功してしまったことを意味するものだからです。攻撃プログラムが起動すると、被害者はコマンドシェルを攻撃者に返却します。このキャプチャでは、攻撃者がdirコマンドを入力し❷、被害者のホストのディレクトリを見ていることがわかります❸。

コマンドシェルにアクセスした攻撃者は、被害者のホストの管理者権限が得られるので、好き放題できるようになります。たった1度の、わずか数秒のクリックで、被害者のホストの管理権限はすべて攻撃者に渡ってしまうのです。

このような攻撃プログラムは、ネットワーク上のIDSに検出されるのを防ぐため、やり取りされる際はわからないように通常エンコードされています。そのためこの攻撃プログラムについてあらかじめ知っているか、攻撃プログラムのコードの例がない場合は、さらなる解析をしない限り、被害者のシステムで何が起きているかを正確に把握するのは困難です。幸いにもこのパケットキャプチャには、悪意あるコードの明らかな兆候、つまり<script>タグで囲まれた難読化されたテキスト、奇妙なiframe、そして平文のコマンドシェルがありました。

Auroraの攻撃プログラムがどのように攻撃するかをまとめてみましょう。

- 被害者は信頼できると見せかけたメールを攻撃者から受け取り、その中のリンクをクリックすると、攻撃者の悪意あるサイトへGETリクエストが送信されます。
- 攻撃者のWebサーバが被害者に302リダイレクトを送信、被害者のブラウザはリダイレクトされた先のURLに、自動的にGETリクエストを送信します。
- 攻撃者のWebサーバは、攻撃プログラムと、悪意あるGIFイメージへのリンクを含んだiframeを含む怪しいJavaScriptコードの入ったWebページをクライアントに表示します。
- 被害者は悪意あるイメージにGETリクエストを送信し、これをサーバからダウンロードします。
- 先に送信されたJavaScriptコードが悪意あるGIFによって解読され、被害者のマシン上でコードが実行され、Internet Explorerの脆弱性が攻撃されます。
- 脆弱性により侵入されると、コードに隠されたペイロードが実行され、4321番ポートで被害者と攻撃者間の新たなセッションが開始されます。
- ペイロードからコマンドシェルが生成されて、攻撃者へ返却され、攻撃者がコマンドシェルを操れるようになります。

防御側の視点から言うと、このキャプチャファイルからIDSのシグネチャを作成すれば、今後この攻撃が起こるのを防げるかもしれません。たとえば、`<script>`タグで囲まれたテキストの最後にある平文の部分のように、難読化されていない部分でフィルタすることが可能です。あるいはURLに「info」が含まれるサイトへ302でリダイレクトされたすべてのHTTPトラフィックをフィルタするシグネチャを記述することも考えられます。こうしたシグネチャを現場で使うにはさらに調整する必要がありますが、悪くないアイデアです。



悪意あるトラフィックのサンプルをもとにシグネチャを作成するのは、未知の脅威からネットワークを守ろうとする人にとって重要なステップです。ここで説明したようなキャプチャは、シグネチャを作成するスキルを磨くのに最適です。侵入検知と攻撃シグネチャについてさらに知りたければ、Snortプロジェクト<http://www.snort.org/>を参照してください。

10.2.2 ARPキャッシュポイズニング

arppoisson.pcap

ARPキャッシュポイズニングについては、回線に潜入し、パケット解析したい機器からのトラフィックに割り込む方法として2章で説明しました。これはネットワークエンジニアにとって、効果的かつ有益なツールとなります。しかしながら、悪意を持って使用されると、中間者（MITM）攻撃として致命的なものにもなり得るのです。

中間者攻撃では、攻撃者は通信の割り込みや改竄を行うために、2台の機器間のトラフィックをリダイレクトします。中間者攻撃には、セッションハイジャック、DNSスプーフィング、SSLハイジャックを含む、さまざまな方法があります。

ARPキャッシュポイズニングでは、特別に作られたARPパケットにより、2台の機器に互いに通信し

あっていると思い込ませますが、実際には、中間に位置してパケットを転送している第3者と通信させます。

ファイル `arppoisson.pcap` には、ARP キャッシュポイズニングのサンプルが含まれています。ファイルを開くと、一見何の変哲もないように見え、被害者である 172.16.0.107 が Google を閲覧し、検索しているのがわかります。この検索により、DNS クエリが混じった HTTP トラフィックが生じています。

ARP キャッシュポイズニングは第2層で使われるテクニックなので、[Packet List] ペインでパケットを調べてみることもできますが、不正行為を見つけるのは難しそうです。[Packet List] ペインにいくつかカラムを追加して、わかりやすくしてみましょう。

1. メニューから [Edit] → [Preferences] を選択します。
2. [Preferences] ダイアログの左側の [Columns] をクリックします。
3. [Add] ボタンをクリックします。
4. 「Source MAC」と入力し、Enter キーを押します[†]。
5. [Field type] のドロップダウンリストで、[Hw src addr (resolved)] を選択します。
6. 追加したばかりのエントリをクリックして、[Source] カラムのすぐあとにくるようドラッグします^{††}。
7. [Add] ボタンをクリックします。
8. 「Dest MAC」と入力し、Enter キーを押します。
9. [Field type] のドロップダウンリストで、[Hw dest addr (resolved)] を選択します。
10. 追加したばかりのエントリをクリックし、[Destination] カラムのすぐあとにくるようドラッグします。
11. [OK] ボタンをクリックします。

この一連の作業を完了すると[‡]、図10-15のような画面になるはずです。これでパケットの送信元と宛先の MAC アドレスを示す2つのカラムが追加できました。

[†] 監訳注：追加された行にある「New Column」という名称の部分をクリックして、名称を変更可能にしてから左記の作業を行います。

^{††} 監訳注：各行の [Field type] カラムの部分をクリックする必要があります。

[‡] 監訳注：正確には手順11の [OK] ボタンをクリックする前の時点で。

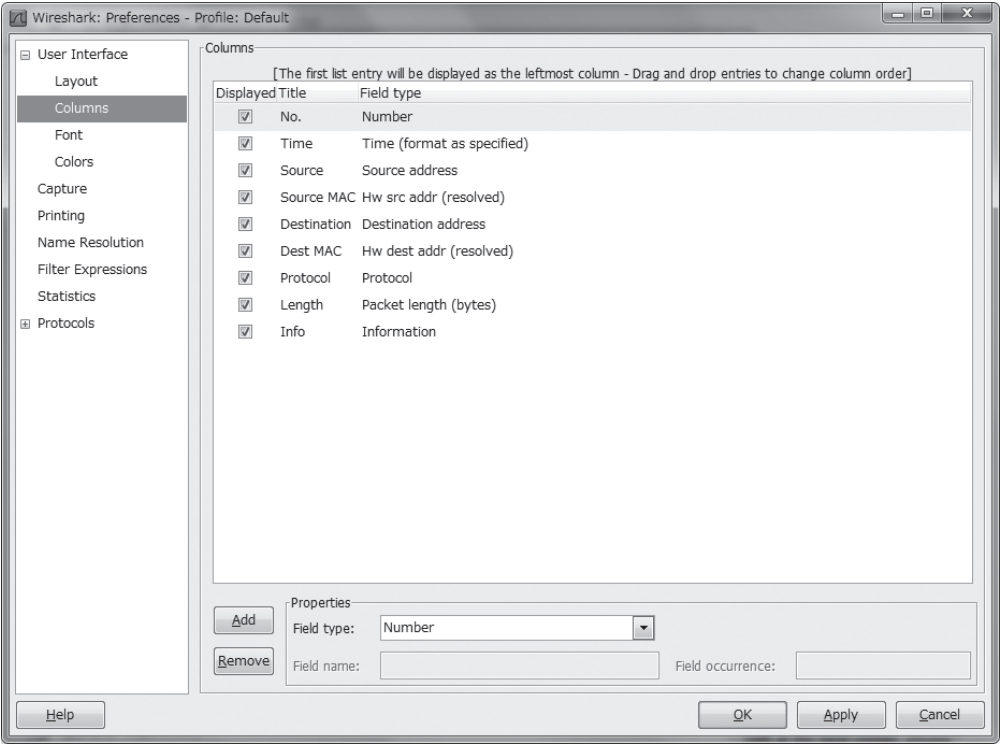


図10-15 送信元と宛先のMACアドレスを示す新たなカラムが追加されたカラム設定画面（バージョン1.8.0）

MACアドレスの名前解決が有効になっていれば、通信している機器がDellとCiscoのものだとわかるはずです。これは非常に重要なので覚えておきましょう。なぜかという、キャプチャをスクロールしていくと、54番目のパケットで、Dellの機器（被害者）と新たに入ってきたHPの機器（攻撃者）との間に、奇妙なARPトラフィックが発生しているからです（図10-16）。

No.	Time	Source	Source MAC	Destination	Dest MAC	Protocol	Info
54	4.171500	HewlettP_bf:91:ee	HewlettP_bf:91:ee	Dell_c0:56:f0	1 Dell_c0:56:f0	ARP	who has 172.16.0.107? tell 172.16.0.1
55	0.000053	Dell_c0:56:f0	Dell_c0:56:f0	HewlettP_bf:91:ee	HewlettP_bf:91:ee	ARP	172.16.0.107 is at 00:21:70:c0:56:f0
56	0.000013	HewlettP_bf:91:ee	HewlettP_bf:91:ee	Dell_c0:56:f0	Dell_c0:56:f0	ARP	3 172.16.0.1 is at 00:25:b3:bf:91:ee

図10-16 Dellの機器とHPの機器間の奇妙なARPトラフィック

先に進む前に、この通信にかかわっているエンドポイントを見ておきましょう。表10-3にまとめました。

表10-3 関連するエンドポイント

役割	ベンダー	IPアドレス	MACアドレス
被害者	Dell	172.16.0.107	00:21:70:c0:56:f0
ルータ	Cisco	172.16.0.1	00:26:0b:21:07:33
攻撃者	HP	不明	00:25:b3:bf:91:ee

では何がこのトラフィックを奇妙にしているのでしょうか。6章のARPについての説明を思い出し
てほしいのですが、ARPパケットには2つのタイプ、つまりリクエストとレスポンスがあります。リク
エストパケットは、特定のIPアドレスに対応付けられたMACアドレスを持つマシンを見つけるため、
ネットワーク上のすべての機器にブロードキャストとして送られます。これによりリクエストを送信し
た通信機器に応答する機器からパケットが送信されます。こうしたことから考えると、この通信には
奇妙な点がいくつかあります。

ひとつは、54番目のパケットがMACアドレス00:25:b3:bf:91:eeの攻撃者から、MACアドレス
00:21:70:c0:56:f0の被害者に直接送信されたARPリクエストであることです❶。この種のリクエスト
は、ネットワーク上のすべてのホストにブロードキャストされるべきなのに、被害者を直接ターゲット
にしています。またこのパケットは攻撃者から送信されたもので、ARPヘッダには攻撃者のMACアド
レスが含まれているにも関わらず、IPアドレスはルータのものとなっています。

このパケットのあとには、MACアドレス情報を含んだ被害者からのレスポンスが攻撃者に送られて
います❷。恐ろしいことが起きているのは56番目のパケットです。攻撃者が送ったこのARPレスポ
ンスは、IPアドレス172.16.0.1のMACアドレスが00:25:b3:bf:91:eeだと伝えているのです❸。問題なの
は、172.16.0.1のMACアドレスは00:25:b3:bf:91:eeではなく、00:26:0b:31:07:33だということです。
172.16.0.1のルータがパケットキャプチャの最初のほうで被害者とやり取りしていたので、この事実が
わかっています。ARPプロトコルはセキュアでないため（一方的に送りつけられたARPリクエストを
ARPテーブルに反映させる）、被害者はルータに行くべきトラフィックを、攻撃者に送ってしまうよう
になりました。



このパケットキャプチャは被害者のホスト上のものなので、全体像は見られません。実
際に攻撃を仕掛けるには、攻撃者の機器が被害者の機器であるとルータに思い込ませる
ために、一連のパケットをルータに送信しなければなりません。しかしこれらのパケッ
トを確認するには、ルータ（あるいは攻撃者）上のパケットキャプチャが必要です。

両者をうまく欺くと、図10-17のように、被害者とルータ間の通信が攻撃者へと流れます。

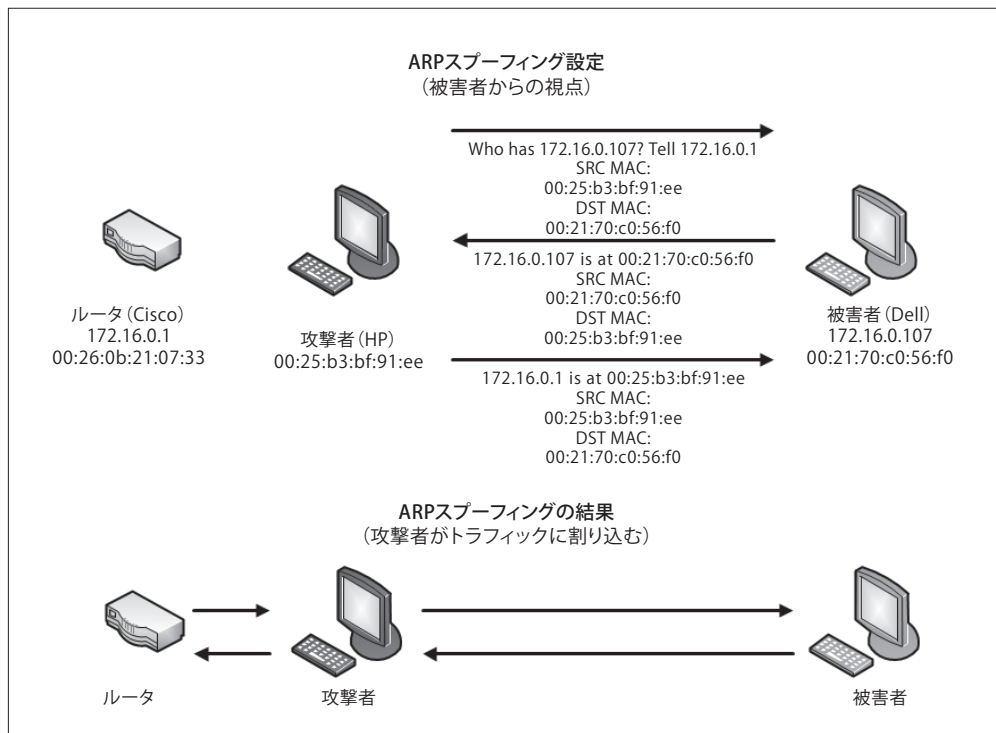


図10-17 ARPキャッシュポイズニングによる中間者攻撃

57番目のパケットにより攻撃の成功が確認できます。このパケットを、奇妙なARPトラフィック前に送ったパケット(40番目など)と比べると(図10-18)、リモートのサーバ(Google)のIPアドレスは同じですが①、宛先のMACアドレスが変わっています②。MACアドレスが変わったことから、現在トラフィックはルータを通過する前に、攻撃者を経由していることがわかります。

この攻撃は本当に目立たないので、検出が非常に難しくなります。こうした攻撃専用設定されたIDSの助けを借りるか、ARPテーブルエントリの急な変更を検出するよう設計されたソフトウェアが必要です。解析しているネットワーク上のパケットをキャプチャするためにARPキャッシュポイズニングを利用することが多くなるでしょうが、諸刃の剣だということを知っておくべきです。

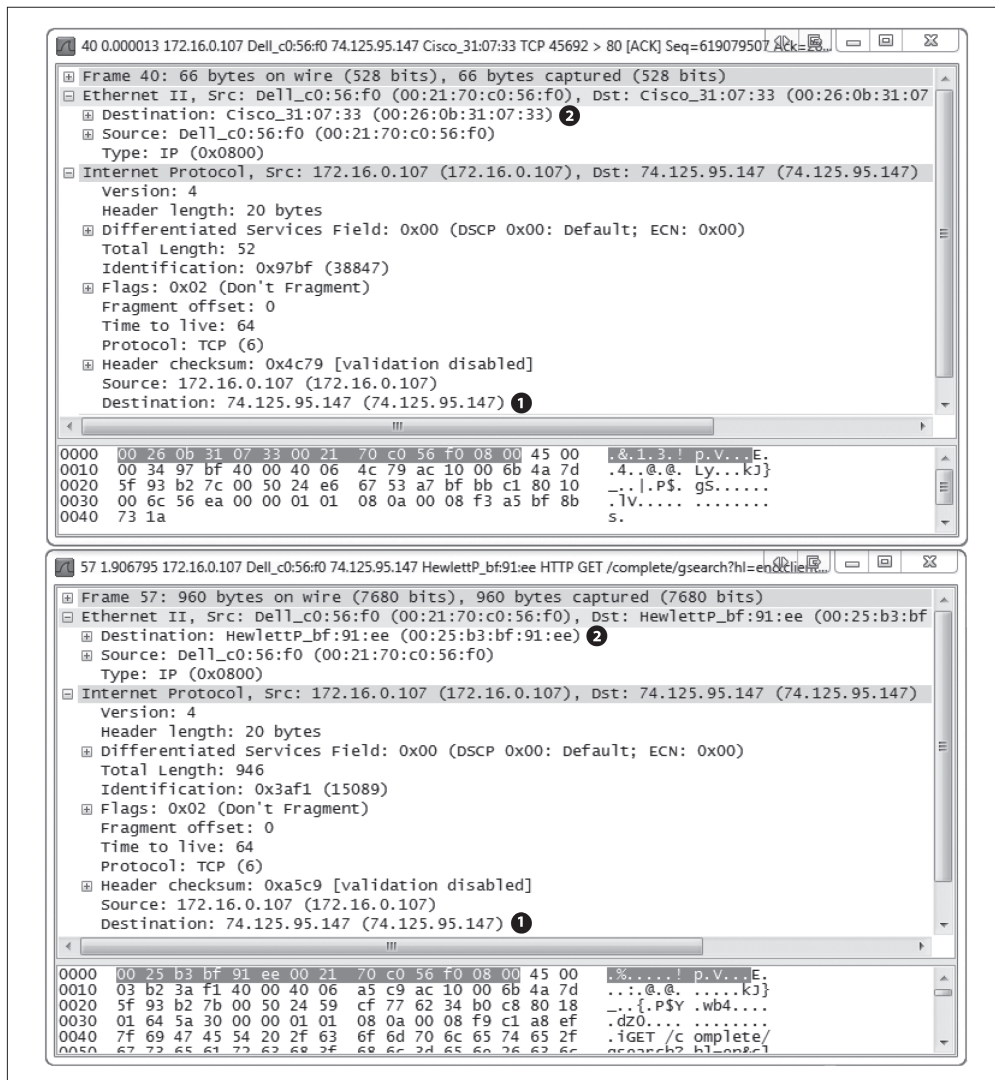


図10-18 宛先MACアドレスの変化が攻撃の成功を示す

10.2.3 リモートアクセス型のトロイの木馬

ratinfected.pcap

ここまでは、キャプチャを調べる前に、何が起きているのかがわかっているセキュリティのトラブルを見てきました。攻撃がどのように行われるかを学ぶには良い方法ですが、あまり現実的ではありません。実際のシナリオでは、ネットワーク防御の仕事を行う人々が、ネットワークを行き来するすべてのパケットを調べたりすることはありません。その代わりに、何らかのIDSを使ってネットワークトラフィックに異常があれば警告が発生するようにすることで、あらかじめ定義した攻撃シグネチャをも

とに、さらなる調査が行えるようにしています。

次のシナリオでは、実際の解析のように、簡単な警告から始めましょう。ここではIDS (Snort) が次のような警告を出しています。

```
[**] [1:132456789:2] CyberEYE RAT Session Establishment [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/18-12:45:04.656854 172.16.0.111:4433 -> 172.16.0.114:6641
TCP TTL:128 TOS:0x0 ID:6526 IpLen:20 DgmLen:54 DF
***AP*** Seq: 0x53BAEB5E Ack: 0x18874922 Win: 0xFAF0 TcpLen: 20
```

次の段階として、この警告を引き起こしたシグネチャルールを確認します。

```
alert tcp any any -> $HOME_NET any (msg:"CyberEYE RAT Session Establishment";
content:"|41 4E 41 42 49 4C 47 49 7C|"; classtype:trojan-activity;
sid:132456789; rev:2;)
```

このルールは、16進数のコンテンツに41 4E 41 42 49 4C 47 49 7Cが含まれるパケットが内部ネットワークに侵入してきたら、警告を発するよう設定されています。このコンテンツは可読可能なASCIIだと「ANA BILGI」となります。これが検出されると警告が発せられ、CyberEYEによるリモートアクセス型のトロイの木馬 (RAT/Remote-access Trojan) が存在する可能性を知らせます。RATは被害者のホスト上で秘密裏に実行され、攻撃者へと接続する悪意あるプログラムで、これにより攻撃者は被害者のマシンをリモート操作することが可能となります。



CyberEYEはRAT実行ファイルを作成し、感染したホストを操るための、トルコ生まれの有名なツールです。皮肉にも、Snortのルールで発見したら警告を発するよう設定した「ANA BILGI」とは、トルコ語で「基本情報」という意味です。

今度はファイルratinfected.pcapの警告に関連するトラフィックを見てみましょう。このSnortの警告は、通常は警告の契機となったパケット1つだけをキャプチャしますが、幸いなことに、ここではホスト間の通信シーケンス全体を確認できます。次のようにSnortのルールで定めた16進数文字列を検索します。

1. メニューから [Edit] → [Find Packet] を選択します。
2. [Hex Value] ラジオボタンをクリックします。
3. テキストの部分に値「41 4E 41 42 49 4C 47 49 7C」を入力します。
4. [Find] ボタンをクリックします。

まずは4番目のパケットのデータ部分で上記の文字列が見つかるはずです❶ (図10-19)。

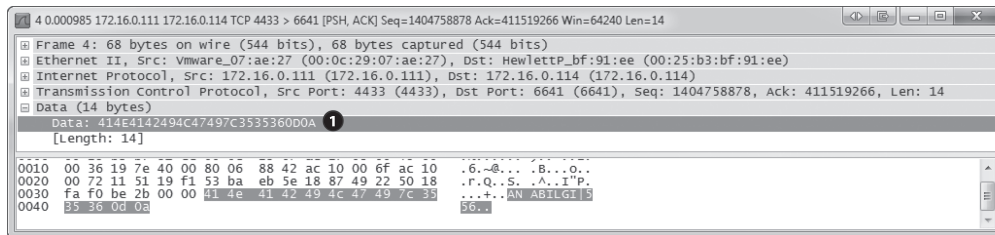
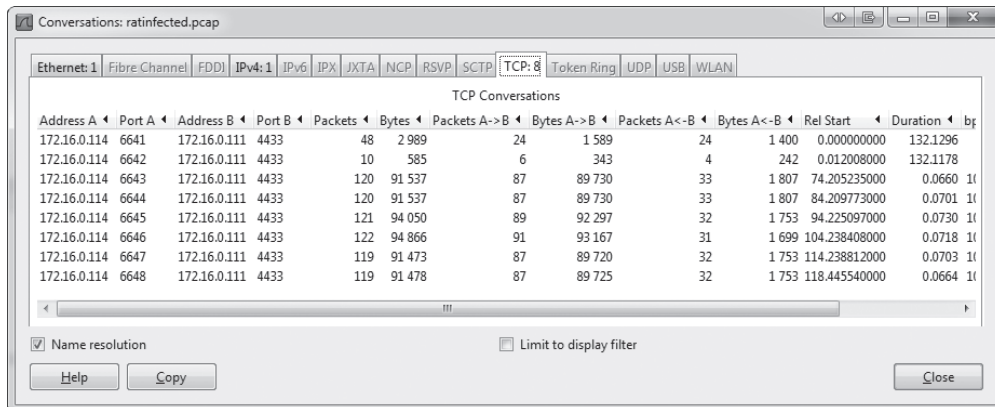


図10-19 4番目のパケットでSnortの警告にあった文字列が見つかる

[Edit] → [Find Next] メニューコマンドを何度か繰り返すと、5、10、32、156、280、405、531、652番目のパケットでもこの文字列が見つかります。このキャプチャのすべての通信が攻撃者（172.16.0.111）と被害者（172.16.0.114）との間のものですが、文字列があったパケットは複数の対話にまたがっているようです。4番目と5番目のパケットは4433番と6641番のポートを使っていますが、ほかのパケットのほとんどが4433番ポートとランダムに選択されたエフェメラルポートを使っています。[Conversations] ダイアログのTCPタブを見れば、複数の対話の存在が確認できます（図10-20）。

図10-20 攻撃者と被害者の間に3つの対話が存在している[†]

色分けすることによって、それぞれの対話を分類することができます。

1. [Packet List] ペインの上にある [Filter] ボックスで、「(tcp.flags.syn == 1) && (tcp.flags.ack == 0)」というフィルタを入力し、[Apply] ボタンをクリックします。これでトラフィックの各対話の最初のSYNパケットが選択できます。
2. 最初のパケットを右クリックし、[Colorize Conversation] を選択します。
3. [TCP] を選択し、色を選びます。
4. 残りのSYNパケットについても同じプロセスを繰り返し、それぞれについて違う色を選びます。

[†] 監訳注：対話の数は8つの誤りだと思われます。

5. 終了したら [Clear] をクリックし、フィルタを削除します。

対話を色分けすると、互いがどう関連しているかが見てわかるようになるので、2つのホスト間の通信処理を追跡しやすくなります。最初の対話（6641番／4433番ポート）で2つのホストが通信が始まっているので、ここから始めるのがよいでしょう。対話内のパケットのどれかを右クリックし、[Follow TCP Stream] を選択して、やり取りされているデータを参照します（図10-21）。

まず攻撃者から被害者に、ANABILGI|556というテキスト文字列が送信されているのがわかります①。さらに被害者は、コンピュータ名（CSANDERS-6F7F77）、使用しているOS（Windows XP Service Pack 3）などを含む基本的なシステム情報をレスポンスとして送ってから②、攻撃者にBAGLIMI?という文字列を送信し続けます③。攻撃者から返却された通信はCAPSCREEN60という文字列④のみで、これは6回現れています。

攻撃者から返却されたCAPSCREEN60という文字列が気になるので、何を意味するものか、ちょっと見てみましょう。再度検索ダイアログを使い、[String] オプションを指定して、パケット内でのテキスト文字列を検索します。

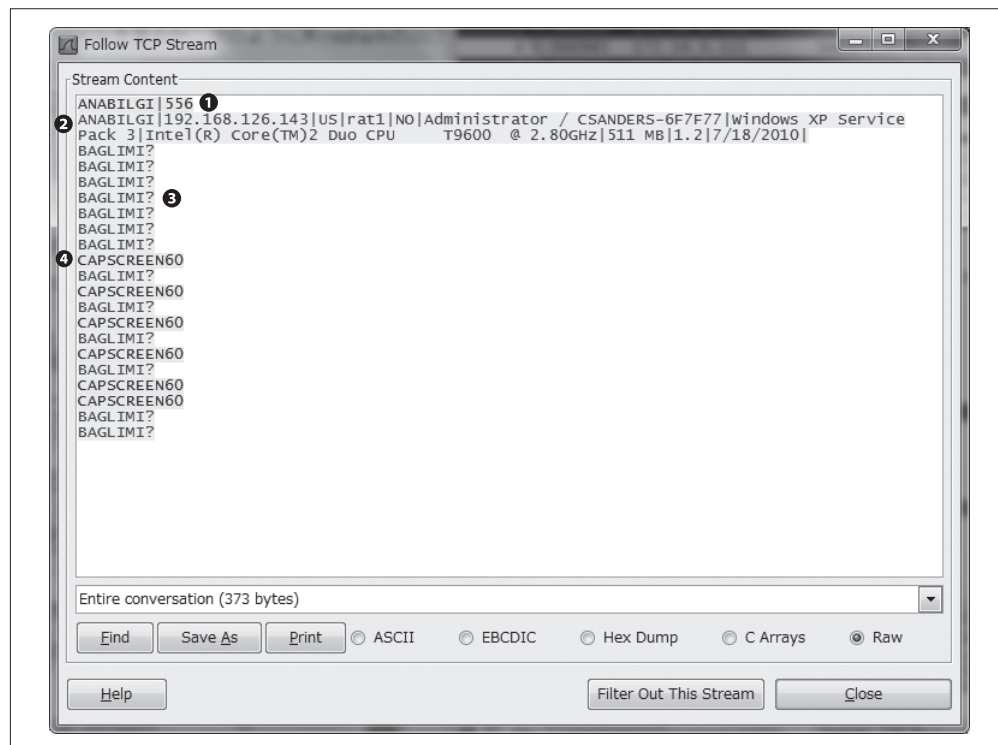


図10-21 最初の対話から面白い結果が出た（バージョン1.8.0）

この検索を実行すると、最初に27番目のパケットで文字列が見つかります。この情報が興味深いの

は、文字列が攻撃者からクライアントに送られるとすぐ、クライアントがパケットの受け取りを確認して、29番目のパケットで新しい対話が始まっていることです。

この新しい対話のTCPストリームの出力を追跡すると(図10-22)、見慣れた文字列であるANABILGI|556が目に入り、そのあとにSH|556という文字列、最後にCAPSCREEN|C:\WINDOWS\jpgevhook.dat|84972と続いています①。CAPSCREENのあとにファイルパスが指定されていて、そのあとに解読不能なテキストがあることに気づくでしょう。ここで面白いのは、解読不能なテキストがJFIFという文字列の先頭に追加されていることで②、これはGoogle検索をかけるとJPGファイルの先頭に存在するものでした。

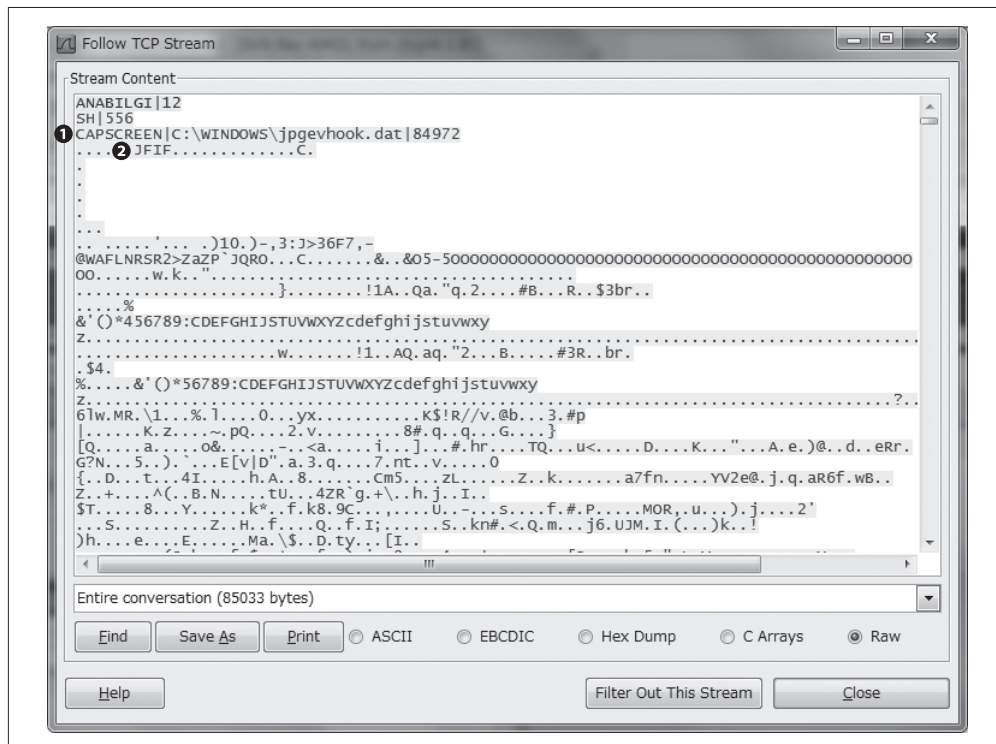


図10-22 攻撃者がJPGファイルのリクエストを開始しようとしている (バージョン1.8.0)

この時点では、攻撃者はこのJPGイメージを転送するために対話を開始したと判断していいでしょう。しかしさらに重要なのは、このトラフィックからコマンドの構造が見えることです。CAPSCREENは攻撃者がJPGイメージを転送するために起動したコマンドのようです。実際CAPSCREENコマンドが送信されると、いつも結果は同じです。これを検証するには、各対話のストリームを見るか、次のようにWiresharkのIOグラフ機能を使います。

1. [Statistics] → [IO Graphs] を選択します。

2. フィルタ `tcp.stream eq 2`、`tcp.stream eq 3`、`tcp.stream eq 4`、`tcp.stream eq 5`、`tcp.stream eq 6` を、5つのフィルタダイアログにそれぞれ挿入します。
3. [Graph 1] [Graph 2] [Graph 3] [Graph 4] [Graph 5] のボタンをクリックし、指定したフィルタのデータポイントを有効にします。
4. Y軸のスケールを [Byte/Tick] に変更します。

図10-23がそのグラフです。

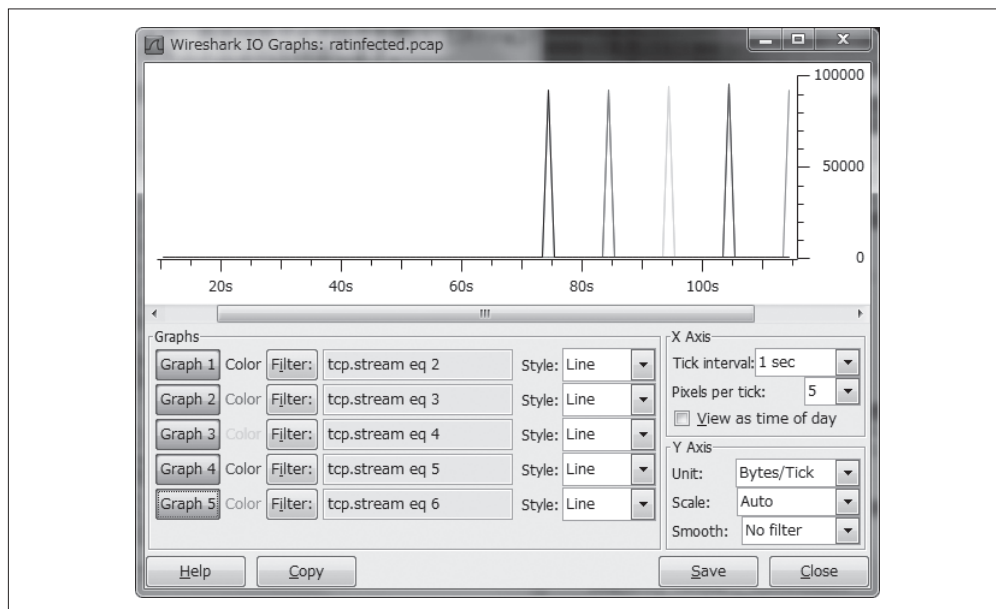


図10-23 同じ活動が繰り返されていることを示すグラフ (バージョン1.8.0)

このグラフによれば、各対話に同じ量のデータが含まれていて、同じ長さのようです。これでこの活動が数回繰り返されていると判断できます。

転送されたJPGイメージのコンテンツについては想像がつくでしょうが、これらJPGファイルの中身が実際に見られるかどうか試してみましょう。WiresharkからJPGデータを抽出するには、次のステップを実行します。

1. まず図10-22の前の文章で説明したように、パケットのTCPストリームを追跡します。
2. 通信を分割して、被害者から攻撃者へ送られたデータストリームのみを参照することができます。これを行うには「Entire Conversation (85033 bytes)」と表示されているドロップダウンの横の矢印を選択し、`172.16.0.114:6643 --> 172.16.0.111:4433 (85020 bytes)` となるようにします。トラフィック (矢印) の向きに注意してください。

3. [Save As] ボタンでデータを保存します。拡張子を .jpg とするのを忘れないように。

このイメージファイルを開こうとしても開かないのであわてるかもしれませんが、もう1段階残っています。8章でFTPトラフィックからファイルを抽出したときとは異なり、このトラフィックは本来の内容にデータが追加されています。TCPストリームの最初の2行はトロイの木馬のコマンドシーケンスの一部で、JPGを構成しているデータではありません(図10-24)。ストリームを保存すると、この外部データも保存されます。その結果、JPGファイルヘッダを探すファイルビューワは、探しているヘッダとは一致しない内容を参照することとなり、画像が開かないのです。

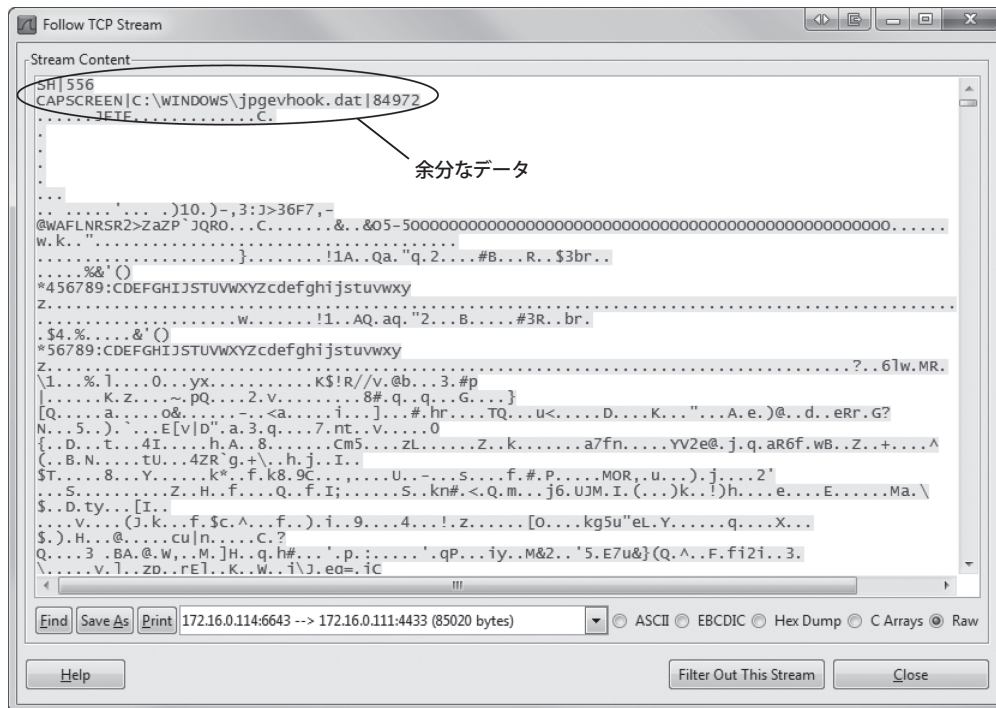


図10-24 トロイの木馬が追加した外部データが、ファイルを開くことを妨げている

この問題を修正するのは簡単な作業ですが、バイナリエディタを使ったちょっとした操作が必要です。この作業をファイルカービング (file carving) と呼びます。図10-25では、WinHexを使ってJPGファイルの先頭のバイト列をハイライトしています。バイナリエディタを使ってこれらのバイトを削除し、画像ファイルを保存しましょう。

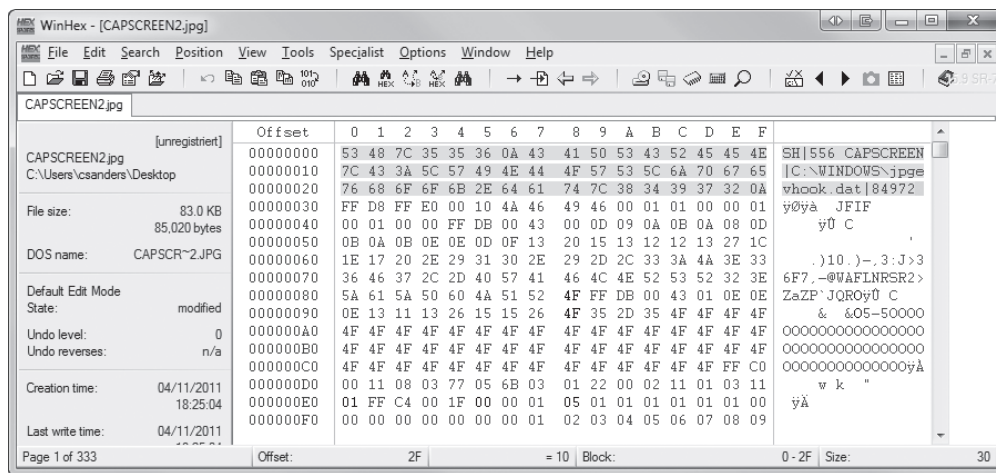


図10-25 JPGファイルから余分なバイト列を取り除く

余分なデータを削除すれば、ファイルが開くはずです。トロイの木馬が被害者のデスクトップのスクリーンキャプチャを乗っ取り、攻撃者へと返却していたことがこれではっきりしました(図10-26)。



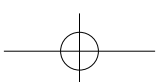
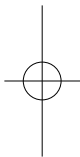
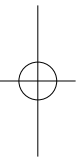
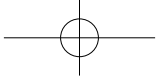
図10-26 転送されたJPGは被害者のホストのスクリーンキャプチャだった

これらの通信シーケンスが完了すると、通信は、通常のTCPティアダウンによって終了します。このシナリオは、IDSによる警告をもとにトラフィックを解析する際に、侵入アナリストがたどる作業の一例です。

- 警告と警告を引き起こしたシグネチャを調べます。
- シグネチャが実際にトラフィックに存在することを確認します。
- トラフィックを調査し、侵入されたホストで攻撃者が何を行ったかを見つけます。
- 被害者からさらに重要な情報が漏れる前に、問題の対処を開始します。

10.3 まとめ

セキュリティに関するシナリオにおけるパケットキャプチャの絞り込み、一般的な攻撃の解析、IDSによる警告への対応だけで、1冊の本が書けます。この章では一般的なスキャンと列举の方法、中間者攻撃、システムへの侵入に関する2つの実例、ホストが攻撃されて乗っ取られてしまった場合に何が起こるかについて説明しました。



11章

無線LANのパケット解析

無線LANの世界は、伝統的な有線ネットワークとは少々違うものです。TCPやIPといった一般的な通信プロトコルを使う点は同じですが、OSI参照モデルの下層レベルへ行くと話が少々変わってきます。無線という特性のため、データリンク層が特に重要になってくるのです。この層の特性のため、アクセスできるデータとキャプチャする方法が変わってきます。

そう考えると、まるまる1章を無線LANでのパケットキャプチャと解析にあてても不思議ではないでしょう。この章では、パケット解析の観点で、なぜ無線LANが特別なのか、またこの課題をどう乗り越えるかについて、もちろん無線LANでのパケットキャプチャの実例を通じて、説明していきます。

11.1 物理面での考察

無線LANで転送されているデータをキャプチャし、解析するときにまず考えなければいけないのは、物理的な転送媒体です。これまではLANケーブルで通信してきたため、物理層については考えてきませんでした。しかし無線LANでは目に見えない電波で通信を行い、パケットが空中を飛び交うのです。

11.1.1 一度に1つのチャンネルをキャプチャする

無線LANのトラフィックのキャプチャで一番特徴的なのは、無線の周波数帯域（スペクトラム）が共有されている媒体だということです。各クライアントが個別のLANケーブルでスイッチに接続している有線LANと異なり、無線LANの通信媒体はクライアント間で共有される帯域で、限りがあります。もっとも、ある無線LANが802.11の周波数帯域に占める割合はほんのわずかなので、複数の機器が周波数帯域の異なる部分を使い、同じ物理的空間において動作できるのです。



無線LANは、IEEE（米国電気電子技術者協会）が策定した802.11規格に基づいています。この章に登場する無線LANは802.11規格に準拠したものを前提としています。

帯域の分割は、周波数帯域をチャンネルに分割することで実現しています。チャンネルとは802.11の無線周波数帯域を単純に分割したものです。米国には11のチャンネルがあります（国によってはそれ以上のチャンネルがある場合もあります）[†]。無線LANは一度に1つのチャンネルで通信するので、同時に1つのチャンネルだけをキャプチャすることが可能です（図11-1）。そのため、たとえばチャンネル6の無線LANをトラブルシューティングするときには、チャンネル6でのトラフィックをキャプチャするように機器を設定する必要があります。

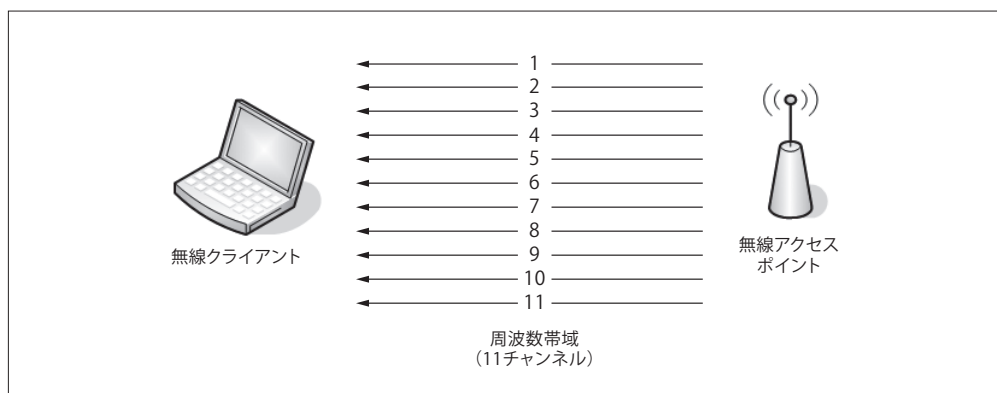


図11-1 同時に1つのチャンネルしかキャプチャできないので、無線LANのキャプチャは面倒



伝統的な無線LANのキャプチャは同時に1つのチャンネルでしか行えませんが、例外が1つだけあります。一部の無線LANスキャナアプリケーションは、データを収集するために素早くチャンネルを切り替える「チャンネルホッピング」というテクニックを採用しています。なかでも特に有名なツールがKismet (<http://www.kismetwireless.net/>)で、1秒間に最高10チャンネルまで切り替えることができ、複数のチャンネルを同時にキャプチャできます。

11.1.2 無線LANの電波干渉

無線による通信では、しばしば空中を伝送されてくるデータの整合性を期待できない場合があります。さまざまな方法で電波へ干渉することが可能です。無線LANには干渉に対処する機能が備わっていますが、それがうまく動作しないときもあります。したがって無線LANでパケットをキャプチャするときには、電波を反射するもの、硬くて大きなもの、電子レンジ、2.4GHzのコードレス電話、厚い壁、高密度のものといった干渉元が近くにないことを確認する必要があります。これらはパケット消失、重複パケット、不正な形式のパケットなどの原因になります。

チャンネル間の干渉も考慮しましょう。同時に1つのチャンネルしかキャプチャできないとはいえ、

[†] 監訳注：日本では14のチャンネルが使用できます。

これには若干の注意が必要です。無線LANの周波数帯域では複数の異なるチャンネルが利用可能ですが、帯域が限られているため、図11-2のようにチャンネル同士でやや重複しています。つまりチャンネル4とチャンネル5にトラフィックがあるときに、どちらかのチャンネルをキャプチャしているとすると、もう一方のチャンネルの packets をキャプチャしてしまう場合があります。一般には、同じ領域に共存する無線LANは、互いに重複しないようにチャンネル1、6、11を使うよう設計されているため、このような問題は起こりにくいのですが、万一の場合に備え、なぜこうした事態が生じるかを理解しておきましょう。

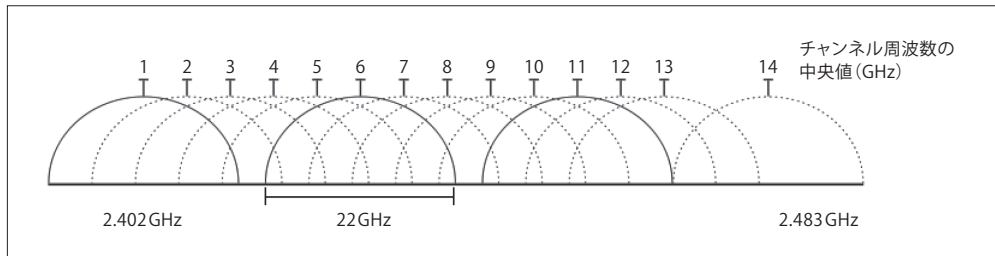


図11-2 周波数帯域が限られているためにチャンネル同士が重複している

11.1.3 電波干渉を検出、解析する

電波干渉のトラブルシューティングは、Wiresharkでパケットを見るだけでできるようなものではありません。無線LANのトラブルシューティングで経験を積むためには、定常的に電波干渉を確認する必要があります。これは、電波の生データや周波数帯域の干渉を表示するツール、スペクトラムアナライザを使えば可能です。

商業用のスペクトラムアナライザは数千ドルはしますが、日常的に使えるソリューションもあります。MetaGeekのWi-Spyという製品は、802.11周波数全体の干渉を監視するUSB機器です。MetaGeekのChanalyzerというソフトウェアと組み合わせると、Wi-Spyは周波数帯域を出力しグラフ化してくれます。その例を図11-3に示します。

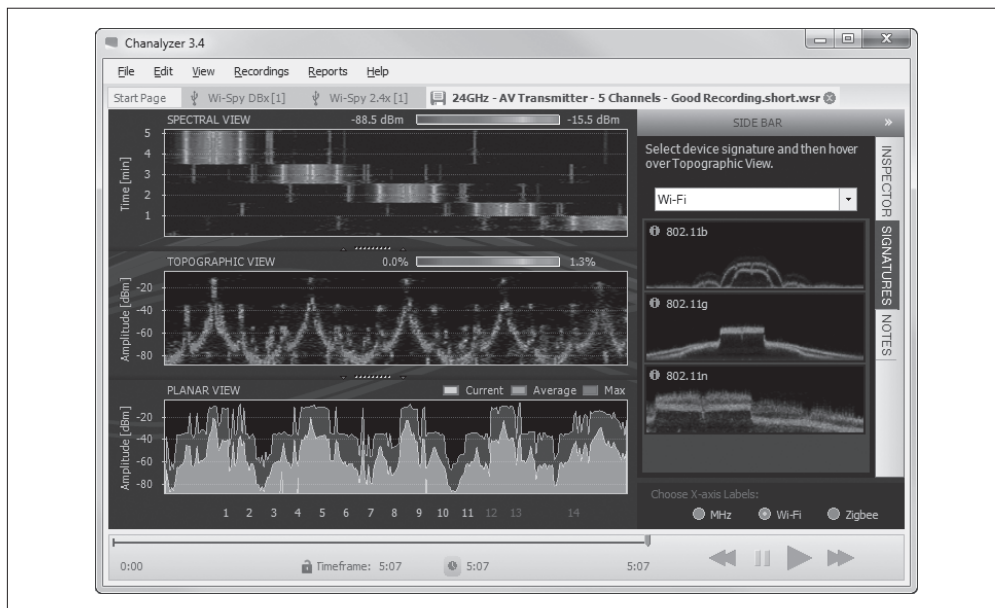


図11-3 Chanalyzerの出力によって同じ領域で複数の無線LANが動作しているのがわかる

11.2 無線LANカードのモード

無線LANのパケットをキャプチャする前に、無線LANカードのモードについて知っておきましょう。無線LANカードには4種類のモードがあります。

マネージドモード[†]

マネージドモードでは、クライアントはアクセスポイント（Wireless Access Point : WAP）に直接接続します。このモードでは、クライアントはアクセスポイントに通信の制御を任せます。

アドホックモード

アドホックモードは、クライアント同士が直接無線を介して通信するときに使います。このモードでは通信を行う2つのクライアントが、アクセスポイントの代わりに通信を制御します。

マスターモード

ハイエンドな無線LANカードは、マスターモードもサポートしています。このモードでは、特別なドライバソフトウェアにより、クライアントがアクセスポイントのような役割を担うことができます。

[†] 監訳注：無線LANの世界ではインフラストラクチャモードと呼ぶことが多いです。

モニターモード

これがもっとも重要なモードです。モニターモードの無線LANカードは、データの送受信を行わずに、飛び交うパケットを監視したいときに用います。Wiresharkで無線LANのパケットをキャプチャする場合は、キャプチャするコンピュータの無線LANカードがモニターモードをサポートする必要があります。パケットキャプチャするために無線LANカードを買うときは、モニターモード（RFMONモードとも呼ばれます）をサポートしていることを確認してください。

無線LANカードのほとんどはマネージドモードかアドホックモードになっています。各モードの動作を図11-14に示します。

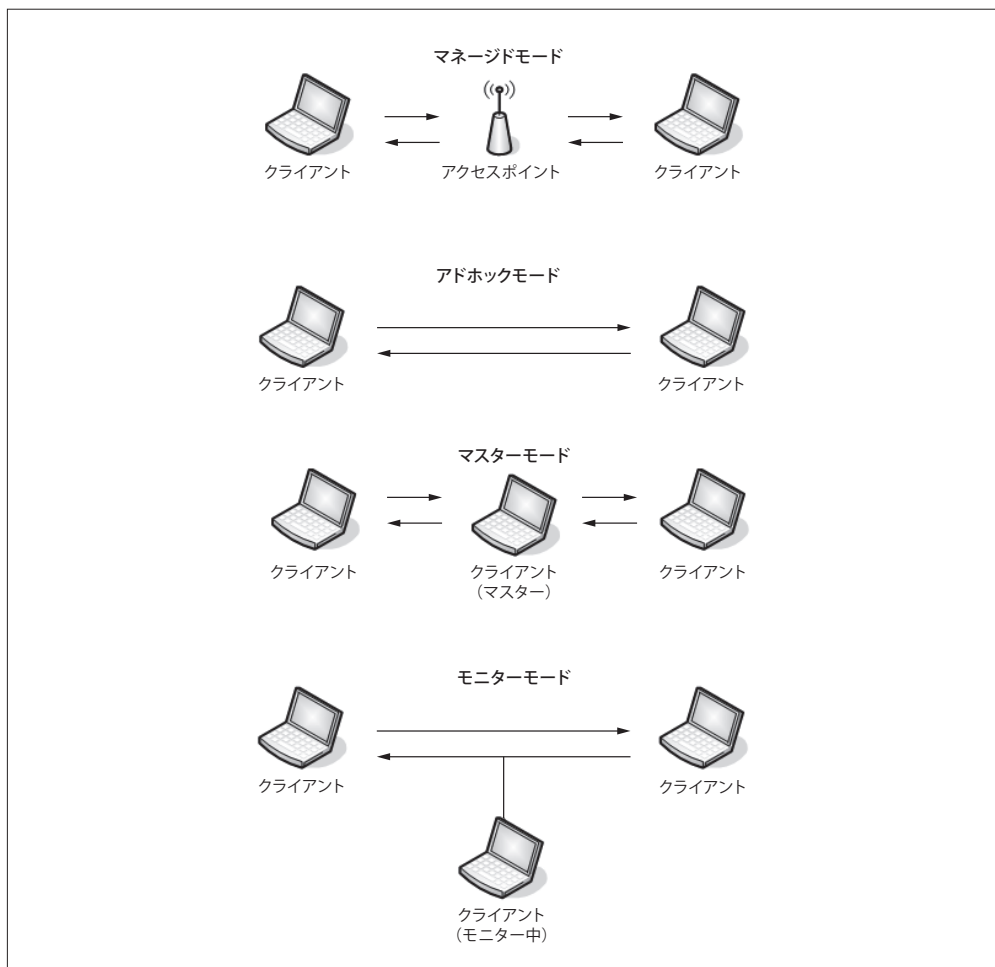


図11-4 無線LANカードのモード



無線LANのパケット解析にはどの無線LANカードがよいかとよく聞かれます。私自身が使っていて、一番お勧めしたいのが「ALFA 1000mW USB」無線アダプタです。どのパケットでも確実にキャプチャできる、もっとも優れた製品として高く評価されています。インターネット上のコンピュータハードウェアサイトの大半で入手可能です。

11.3 Windows上での無線LANのパケットキャプチャ

モニターモードをサポートしている無線LANカードを使っている、WindowsのLANカードドライバではそのモードを使うことができません (WinPcapもサポートしていません)。キャプチャを行うには追加のハードウェアが必要です。

11.3.1 AirPcapの設定

AirPcap (CACE Technologies、現在はRiverbed傘下。 <http://www.cacotech.com/>) はWindows上で無線LANのパケット解析を行うために設計されたものです。AirPcapは無線LANでのパケットキャプチャのために設計されたUSBフラッシュドライブです (図11-5)。AirPcapは3章で説明したWinPcapドライバを使っており、専用の設定画面があります。



図11-5 AirPcapはコンパクトでノートPCと一緒に簡単に持ち運びが可能

AirPcapの設定はオプションが少ないので非常に簡単です。図11-6の [AirPcap Control Panel] から以下のオプションが設定可能です。

[Interface]

キャプチャに使うデバイスを選択できます。高度な解析を行う場合は、複数のAirPcapを使って複数のチャンネルを同時にキャプチャする必要がある場合もあります。

[Blink Led]

AirPcapのLEDを点滅させます。この機能は主に、複数のAirPcapを使っているときに、どれを使っているかを示すためのものです。

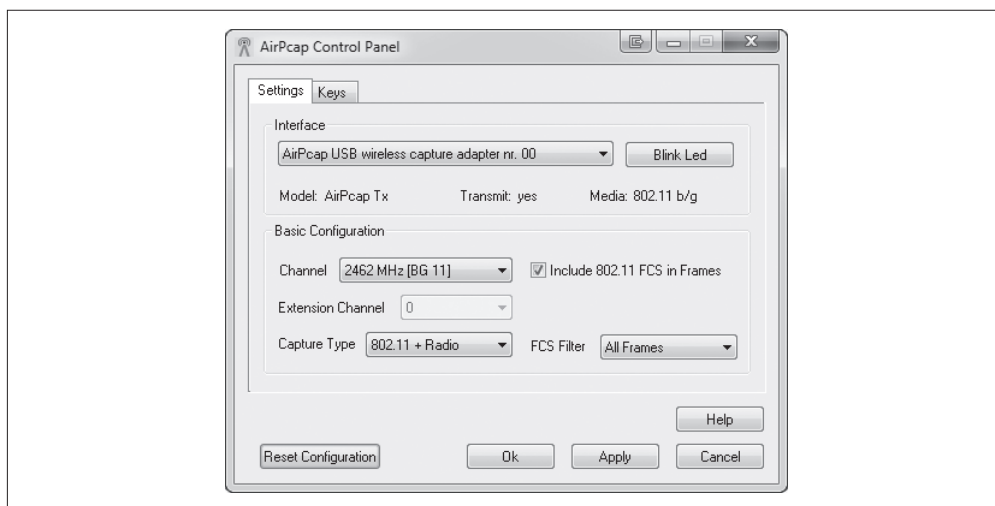


図11-6 AirPcapの設定用プログラム

[Channel]

ここでは、AirPcapを使ってキャプチャするチャンネルを選択します。

[Include 802.11 FCS in Frames]

OSによっては、デフォルトで無線LANパケットのチェックサムの最後の4ビットを取り除いてしまうことがあります。このチェックサムはFCS (Frame Check Sequences) と呼ばれており、転送している間にデータが破損していないことを保証するために使われています。特に理由がなければ、チェックボックスをオンにしてFCSチェックサムを削除しないようにしましょう。

[Capture Type]

[802.11 Only] と [802.11 + Radio] という2つのオプションがあります。[802.11 Only] というオプションは、標準的な802.11のパケットのヘッダをキャプチャするということです。[802.11 + Radio] は、データの転送速度、周波数、信号レベルやノイズレベルを含むラジオタップヘッダもキャプチャします。入手可能なすべての情報を見られるようにするため、[802.11 + Radio] を選択しましょう。

[FCS Filter]

[Include 802.11 FCS in Frames] のチェックボックスをオンにしていなくても、このオプションを有効にしておけばFCSのチェックによりデータが破損していると判断されればパケットはフィルタされます。[Valid Frames] オプションをオンにすれば、FCSのチェックによりデータが正しく受信されたと判断されたものだけが表示されます。

[WEP Configuration]

この画面 (AirPcap Control Panelの [keys] タブから参照可能) では、キャプチャしたい無線LANのWEPキーを入力し、WEPによって暗号化されたデータを解釈できるようにします。WEPキーについては、「11.8 無線LANのセキュリティ」で説明します。

11.3.2 AirPcapを使ったパケットキャプチャ

AirPcapをインストールして設定したら、Wiresharkを起動して [Capture] → [Options] を選択し、[Interface] からAirPcapデバイスを選択します❶ (図11-7) [†]。

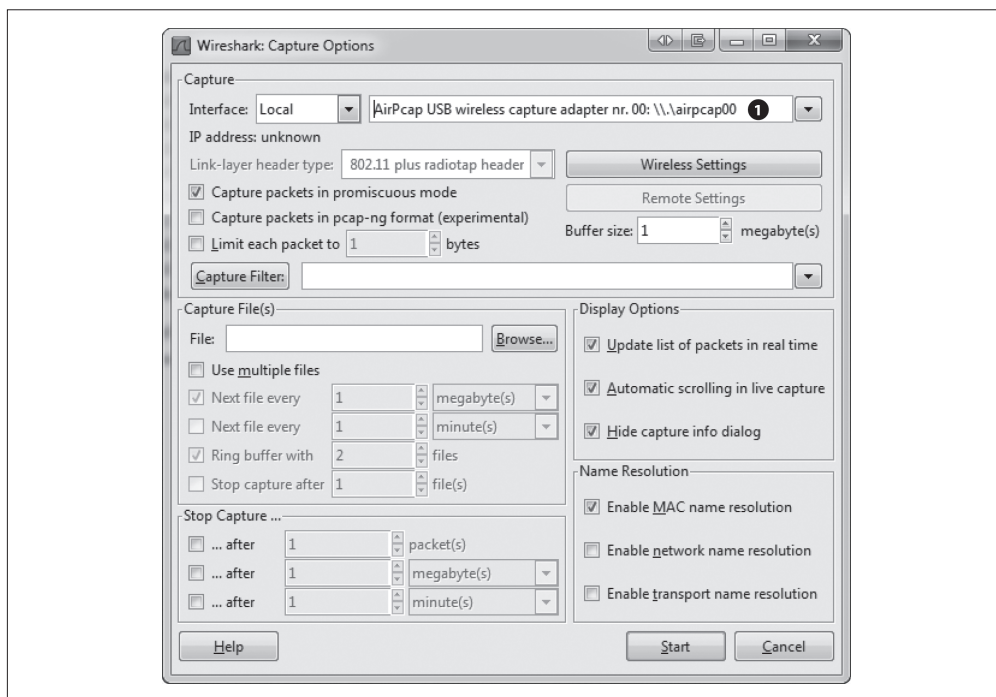


図11-7 キャプチャするインターフェイスとして、AirPcapデバイスを選択する

[Wireless Settings] というボタン以外は見慣れた画面だと思います。このボタンをクリックするとAirPcapの設定用プログラムと同じオプションが表示されます (図11-8)。WiresharkはAirPcapと完全に統合されているため、AirPcapで設定できることはWiresharkでも設定できます。

[†] 監訳注：本書執筆時点で最新版のバージョン1.8.0では、3章の脚注 (図3-5) で説明したように、UIが異なっており、[Wireless Settings] というボタンも図3-5の画面で選択したデバイスをダブルクリックすると表示される図4-11の [Edit Interface Settings] ダイアログに存在します。

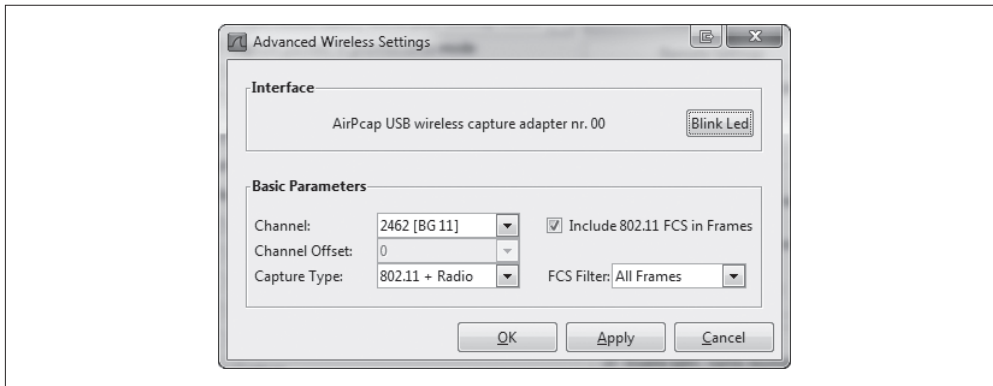


図11-8 [Advanced Wireless Settings] ダイアログではWiresharkからAirPcapの設定ができる

設定がすべて終わったら、[Start] ボタンを押してパケットキャプチャを開始してください。

11.4 Linux上での無線LANのパケットキャプチャ

Linuxでのパケットキャプチャに必要なのは、無線LANカードをモニターモードにすることだけです。残念ながらモニターモードに変更する手順は無線LANカードごとに異なるため、ここでそのやり方を説明することはできません。無線LANカードによっては変更が不要なものもあります。ご自分の無線LANカードについて、Googleで検索してみてください。

Linux上で無線LANカードをモニターモードに変更するもっとも一般的な方法は、Linuxに内蔵されている機能を使うことです。iwconfigコマンドを使えば、無線LANカードを設定できます。コンソール上でiwconfigを実行すると、以下のような結果になります。

```
$ iwconfig
eth0 no wireless extensions
lo0 no wireless extensions
eth1 IEEE 802.11g ESSID: "Tesla Wireless Network"
    Mode: Managed Frequency: 2.462 GHz Access Point: 00:02:2D:8B:70:2E
    Bit Rate: 54 Mb/s Tx-Power-20 dBm Sensitivity=8/0
    Retry Limit: 7 RTS thr: off Fragment thr: off
    Power Management: off
    Link Quality=75/100 Signal level=-71 dBm Noise level=-86 dBm
    Rx invalid nwid: 0 Rx invalid crypt: 0 Rx invalid frag: 0
    Tx excessive retries: 0 Invalid misc: 0 Missed beacon: 2
```

iwconfigコマンドの結果から、802.11gという無線LANプロトコルについての情報が表示されているeth1が無線LANインターフェイスであることがわかります。eth0とlo0では無線LANは使えません。

eth1と表示されている行の下の方を見てください。iwconfigコマンドを実行して得られる無線LANカードのESSID (Extended Service Set ID) や周波数などの情報とともに、モードがManaged

である则表示されています。これを変更する必要があります。

eth1をモニターモードに変更するにはroot権限が必要なので、suコマンドでユーザーを変更します。

```
$ su
Password: <rootのパスワードを入力>
```

rootになれば、無線LANカードのオプションを設定するコマンドを実行することができます。eth1をモニターモードにするには、以下のコマンドを実行してください。

```
# iwconfig eth1 mode monitor
```

モニターモードに変更したら、iwconfigをもう一度実行して変更を有効にします。以下のコマンドを実行してください。

```
# iwconfig eth1 up
```

iwconfigコマンドでチャンネルを切り替えることもできます。eth1のチャンネルを3に切り替えるには、以下のコマンドを実行してください。

```
# iwconfig eth1 channel 3
```



パケットキャプチャをしている間にもチャンネルを切り替えることができますので、必要に応じて変更してください。スクリプトを作ってしまうとより簡単に実行することができます。

設定が終わったらWiresharkを起動し、パケットキャプチャを開始してください。

11.5 802.11のパケット構造

80211beacon.pcap

無線LANと有線LANのパケットの違いは、802.11ヘッダの有無です。この第2層ヘッダにはデータ転送に使う媒体の情報が含まれています。802.11パケットには3つのタイプがあります。

マネジメント

第2層でホスト間のコネクションを確立するために使われるパケットです。マネジメントパケットのサブタイプには、認証、アソシエーション、ビーコンパケットがあります。

コントロール

マネジメントパケットとデータパケットを配送し、パケットの輻輳管理を行います。一般的なサブタイプとしてRTS (Request-to-send) とCTS (Clear-to-send) パケットがあります。

データ

実際のデータを含んだパケットで、また無線LANから有線LANへ転送が可能な唯一のパ

ケットタイプです。

802.11パケットの構造は、パケットのタイプとサブタイプの組み合わせによって決まります。組み合わせはかなりの数になりますが、ここでは80211beacon.pcapファイルのパケットで、そのひとつを見てみましょう。このファイルには、ビーコンというマネジメントパケットのサンプルが含まれています(図11-9)。

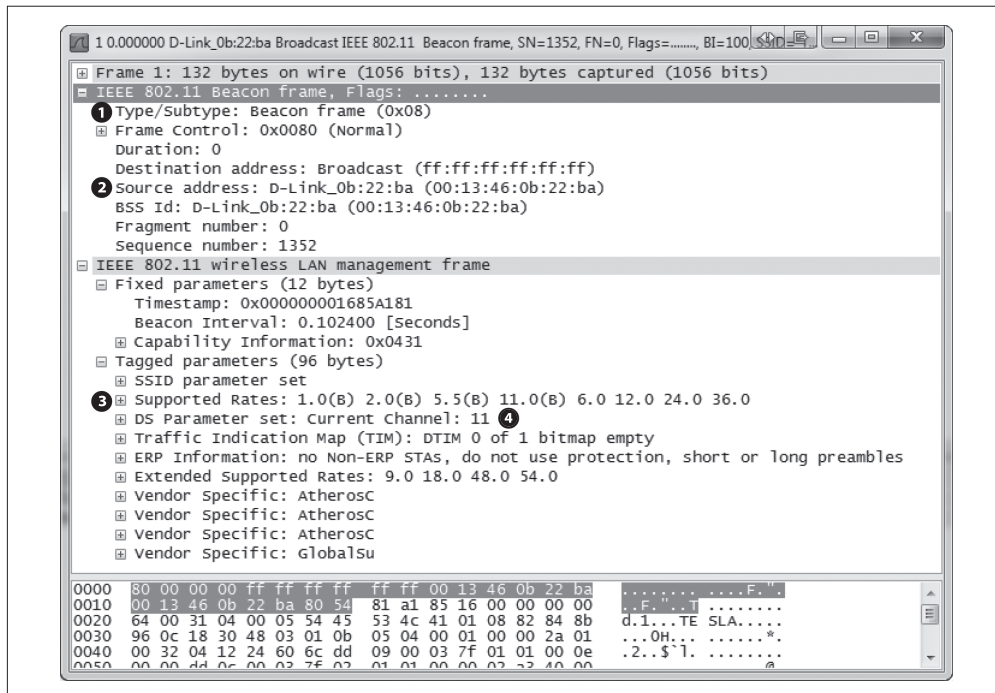


図11-9 802.11ビーコンパケット

ビーコンは無線LANでの通信においてもっとも有益なパケットの1つです。ビーコンはアクセスポイントからチャンネルをまたがるブロードキャストパケットとして送信されるパケットで、アクセスポイントに接続可能なクライアントに対して、接続に必要なパラメータを提示するために送信されます。サンプルでは、このパケットは802.11ヘッダのType/subtypeでビーコンと定義されています①。

802.11 wireless LAN management frameヘッダには、以下の情報を含む多くの情報が含まれています。

Timestamp

パケットが送信された時刻。

Beacon Interval

ビーコンパケットが再送されるまでの間隔。

Capabilities Information

アクセスポイントのハードウェア性能についての情報。

SSID Parameter Set

アクセスポイントがブロードキャストしているSSID（ネットワーク名）。

Supported Rates

アクセスポイントがサポートしているデータの転送速度。

DS Parameter

アクセスポイントがブロードキャストしているチャンネル。

ヘッダには、送信元と宛先のアドレスやベンダー特有の情報も含まれています。

これらの情報をもとに、ビーコンを送信しているアクセスポイントについてかなりの情報が得られます。これは802.11b規格(B) ❸によるD-Link社のデバイス❷で、チャンネル11❹を使っていることがわかります。

802.11 マネジメントパケットの中身や目的はいろいろありますが、一般的な構成はこのサンプルと同じです。

11.6 [Packet List] ペインに無線LANの情報を追加する

Wiresharkの[Packet List] ペインには通常6つのカラムがあります。無線LANの解析を先へ進める前に、[Packet List] ペインに新たに3つのカラムを追加しておきましょう。

RSSI (Received Signal Strength Indication : 受信信号強度) カラム

キャプチャしたパケットの無線周波数 (RF) 信号の強度を示します。

TX Rate (Transmission Rate) カラム

キャプチャしたパケットのデータ転送速度を示します。

Frequency/Channelカラム

周波数と、パケットが収集されたチャンネルを示します。

これらの情報は無線LANでのトラブルシューティングにおいて大きな助けになるでしょう。たとえばクライアントが信号の強度が強いと示しているときに、これらのカラムがあれば本当かどうかを確認することができます。

これらのカラムを[Packet List] ペインに表示させるには、以下の手順に従ってください。

1. メニューから[Edit] → [Preference] を選択します。
2. [Columns] セクションを選択して [Add] ボタンをクリックします。

3. [Title] カラムのテキストボックスに RSSI と入力し、下部にある [Field type] ドロップダウンリストで [IEEE 802.11 RSSI] を選択します。
4. TX Rate と Frequency/Channel カラムについても同じ手順を繰り返し、[Field type] ドロップダウンリストでそれぞれ [IEEE 802.11 TX Rate] と [Channel/Frequency] を選択します。図 11-10 は以上の手順を終えたあとの [Preferences] ダイアログです。
5. [OK] ボタンをクリックして変更を保存します。
6. Wireshark を再起動して新しいカラムを表示します。

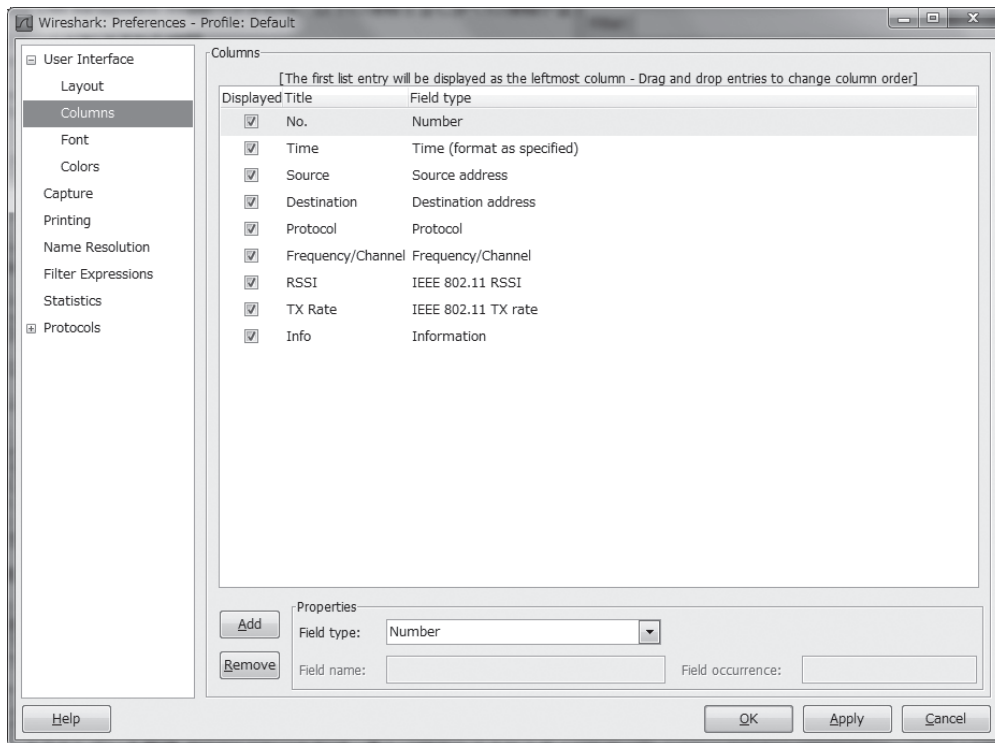


図11-10 無線LAN特有の情報を表示するカラムを [Packet List] ペインに追加する (バージョン1.8.0)

11.7 無線LAN特有のフィルタ

フィルタの有用性については4章で説明しました。有線LANでは各通信機器にLANケーブルが伸びているため、キャプチャしたいパケットのみをキャプチャするフィルタは簡単に作ることができました。しかしながら、無線LANでは機器によって発生するすべてのトラフィックがチャンネル上に共存しており、1つのチャンネルをキャプチャするとさまざまな機器のトラフィックが混在した形で記録されます。ここでは、自分が求めるパケットのみをキャプチャできるようなフィルタの作り方を学びます。

11.7.1 特定のBSSIDでフィルタリング

無線LAN上の各アクセスポイントには、BSSID (Basic Service Set Identifier) と呼ばれる固有の識別子が割り当てられています。アクセスポイントが発信する無線LANのマネジメントパケットとデータパケットの中には、この名前が含まれています。

解析しようとしているBSSIDがわかれば、あとはそのアクセスポイントから送信されるパケットを見つけるだけです。Wiresharkでは、[Packet List] ペインの [Info] カラムでパケットを送信しているアクセスポイントを表示してくれますので、目的のパケットを見つけ出すのは簡単でしょう。

解析したい無線LANのアクセスポイントから送信されているパケットを見つけたら、802.11 ヘッダからBSSIDを確認しましょう。これがフィルタの基本となるアドレスです。BSSIDのMACアドレスが確認できたら、次のようなフィルタを使えます。

```
wlan.bssid.eq 00:11:22:33:44:55:66
```

これでこのアクセスポイントを経由するトラフィックのみがキャプチャされるようになります。

11.7.2 パケット別のフィルタリング

この章の最初で、無線LANのパケットにはいくつかのタイプがあるということを説明しました。これらのタイプやサブタイプによってパケットをフィルタリングすることが必要となることも多いでしょう。タイプについてはwlan.fc.type、タイプとサブタイプを組み合わせる場合はwc.fc.type_subtype というフィルタを使うことができます。たとえばNULLデータパケット (16進数でタイプ2、サブタイプ4のパケット) をフィルタしたい場合、wlan.fc.type_subtype eq 0x24 というフィルタを用いることができます。無線LANのパケットをタイプとサブタイプでフィルタする際に必要になると思われるフィルタを表11-1に示します。

11.7.3 周波数によるフィルタ

複数のチャンネルからのパケットを含むトラフィックを調査するときは、チャンネルに基づくフィルタが非常に役立ちます。たとえばチャンネル1と6以外のトラフィックがあってほしくないという場合は、まず11チャンネルすべてのトラフィックを表示するようにフィルタを設定し、この2つのチャンネル以外でトラフィックが見つければ、設定ミスか通信機器の障害など、何か問題があるということになります。周波数別にフィルタを行うには、次のような構文を使います。

```
radiotap.channel.freq == 2412
```

これはチャンネル1のすべてのトラフィックを表示します。2422の値を別の周波数に変えれば、他のチャンネルに対するフィルタを行えます。表11-2は各チャンネルの周波数の一覧です。

表11-1 無線LANのタイプ／サブタイプと対応するフィルタ一覧

タイプ／サブタイプ	フィルタ構文
マネジメントフレーム	wlan.fc.type eq 0
コントロールフレーム	wlan.fc.type eq 1
データフレーム	wlan.fc.type eq 2
アソシエーション要求 (Association request)	wlan.fc.type_subtype eq 0x00
アソシエーション応答 (Association response)	wlan.fc.type_subtype eq 0x01
再アソシエーション要求 (Reassociation request)	wlan.fc.type_subtype eq 0x02
再アソシエーション応答 (Reassociation response)	wlan.fc.type_subtype eq 0x03
プローブ要求 (Probe request)	wlan.fc.type_subtype eq 0x04
プローブ応答 (Probe response)	wlan.fc.type_subtype eq 0x05
ビーコン	wlan.fc.type_subtype eq 0x08
ディスアソシエート (Disassociate)	wlan.fc.type_subtype eq 0x0A
オーセンティケーション (Authentication)	wlan.fc.type_subtype eq 0x0B
デオーセンティケーション (Deauthentication)	wlan.fc.type_subtype eq 0x0C
アクションフレーム (Action frames)	wlan.fc.type_subtype eq 0x0D
ブロック ACK 要求 (Block ACK requests)	wlan.fc.type_subtype eq 0x18
ブロック ACK (Block ACK)	wlan.fc.type_subtype eq 0x19
PS-Poll (Power save poll)	wlan.fc.type_subtype eq 0x1A
RTS (Request to send)	wlan.fc.type_subtype eq 0x1B
CTS (Clear to send)	wlan.fc.type_subtype eq 0x1C
ACK	wlan.fc.type_subtype eq 0x1D
CF-End (Contention free period end)	wlan.fc.type_subtype eq 0x1E
NULL データ (NULL data)	wlan.fc.type_subtype eq 0x24
QoS データ (QoS data)	wlan.fc.type_subtype eq 0x28
Null QoS データ (Null QoS data)	wlan.fc.type_subtype eq 0x2C

表11-2 無線チャンネルと周波数

チャンネル	周波数
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462

無線LANのトラフィックに対する有用なフィルタは何百とあります。キャプチャフィルタのサンプルは<http://wiki.wireshark.org/>のWireshark Wikiを参照してください。

11.8 無線LANのセキュリティ

無線LANを展開、管理するときの最大の懸念が、送信するデータのセキュリティです。データは空中を飛んでいくため、やり方さえ知っていれば誰でも自由に横取りできますので、データの暗号化が必須です。暗号化されていない場合、WiresharkとAirPcapカードさえあれば、誰でもデータが見られるのです。



SSLやSSHなどの別の層の暗号化を使うと、トラフィックはその層では暗号化されるため、ユーザーの通信内容をパケットキャプチャツールで読むことはできません。

無線LANでセキュアにデータを送信する方法として当初よく使われていたのが、**WEP** (Wired Equivalent Privacy) 規格です。WEPは暗号キーの管理方法にいくつかの弱点が発見されるまで、数年にわたってある程度の広がりを見せていましたが、セキュリティ向上のために、新たな規格が策定されました。それが**WPA** (Wi-Fi Protected Access) と**WPA2**規格です。WPAや、よりセキュアになったWPA2にも欠点はありますが、WEPよりはるかに安全で、実用に耐え得ると考えられています。

この項ではWEPおよびWPAのトラフィックや、認証の失敗例を見ていきます。

11.8.1 WEP認証の成功

80211-WEPauth.pcap

ファイル80211-WEPauth.pcapには、WEPが有効な無線LANへの接続の成功例が含まれています。この無線LANのセキュリティは、WEPキーで保護されています。認証に成功して、暗号化されたデータを受信するために、アクセスポイントに渡す必要があるのがこのWEPキーです。無線LANのパスワードと考えればいいでしょう。

図11-11に示しているように、キャプチャファイルはアクセスポイント(00:11:88:6b:68:30) からクライアント(00:14:a5:30:b0:af)に、4番目のパケットでチャレンジが送信されるところから始まっています❶。クライアントのWEPキーが正しいかどうかを確認するのがチャレンジの目的です。チャレンジは、802.11ヘッダを展開してTagged parametersを参照すると確認できます。

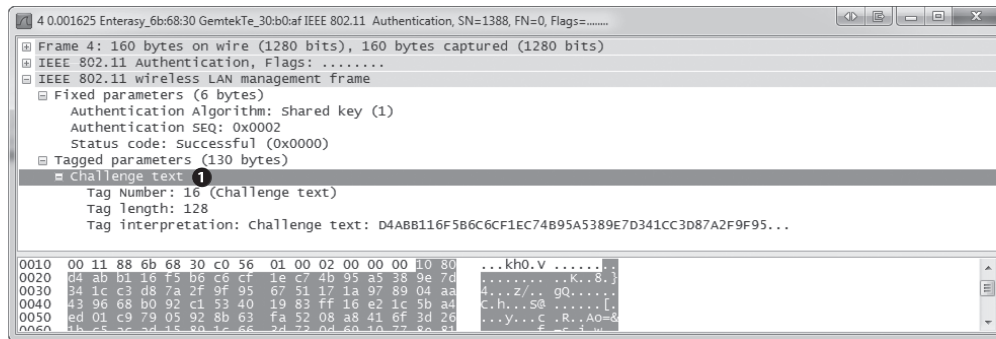


図11-11 アクセスポイントからクライアントにチャレンジが送信されている

チャレンジは5番目のパケットでACKが行われます。クライアントは、WEPキーを使って暗号化されたチャレンジのテキストを復号し、図11-12のようにレスポンスとしてアクセスポイントに返却します①。

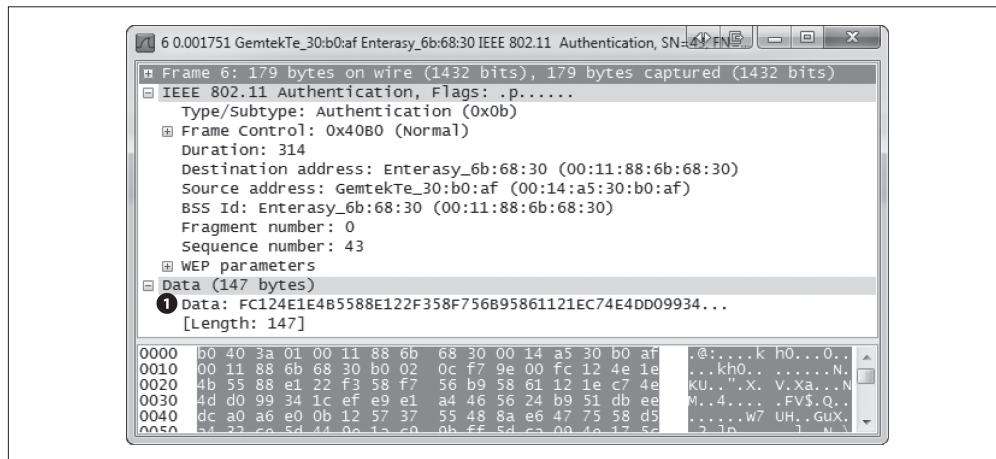


図11-12 クライアントは復号したチャレンジをアクセスポイントに返却する

このパケットは7番目のパケットでACKが行われ、アクセスポイントは図11-13のように8番目のパケットでクライアントにレスポンスを返却します。レスポンスには認証処理が成功したという通知が含まれています①。

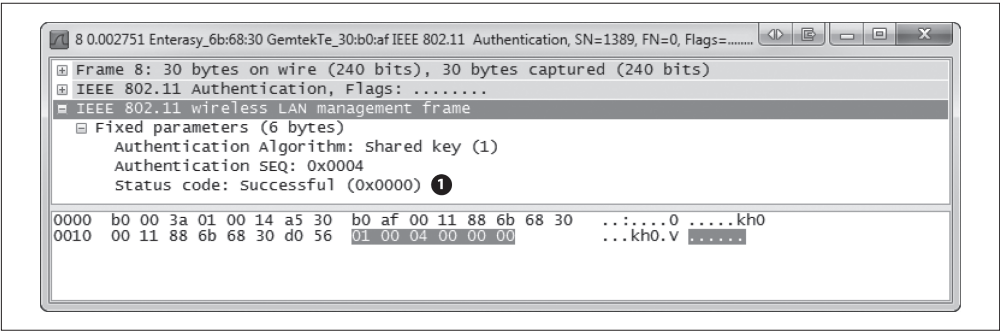


図11-13 アクセスポイントがクライアントに認証が成功したことを伝えている

認証が成功すると、クライアントはアソシエーション要求を送信し、ACKの受信後、ついに無線LANに接続します (図11-14)。

No.	Time	Source	Destination	Protocol	Channel	Info
10	0.000876	GemtekTe_30:b0:af	Enterasy_6b:68:30	IEEE 802.11		Association Request, SN=44, FN=0, Flags=....., SSID="DENVEROFFICE"
11	0.000374			IEEE 802.11		Acknowledgement, Flags=.....
12	0.002627	Enterasy_6b:67:28	Broadcast	IEEE 802.11		Data, SN=1390, FN=0, Flags=p....F.
13	0.000624	Enterasy_6b:68:30	GemtekTe_30:b0:af	IEEE 802.11		Association Response, SN=1391, FN=0, Flags=.....
14	0.000374			IEEE 802.11		Acknowledgement, Flags=.....
15	0.683813	GemtekTe_30:b0:af	Enterasy_6b:68:30	IEEE 802.11		Null function (No data), SN=45, FN=0, Flags=.....T
16	0.000098			IEEE 802.11		Acknowledgement, Flags=.....
17	0.000053	GemtekTe_30:b0:af	Broadcast	IEEE 802.11		Data, SN=46, FN=0, Flags=p....T

図11-14 認証の処理は単純なアソシエーション要求とアソシエーション応答によって行われる

11.8.2 WEP認証の失敗

80211-WEPauthfail.pcap

次のサンプルでは、ユーザーがアクセスポイントに接続するためにWEPキーを入力したものの、数秒後にクライアントのユーティリティが理由はわからないが接続できなかったと報告してきています。そのときのファイルが80211-WEPauthfail.pcapです。

成功した接続と同様、ここでもアクセスポイントが3番目のパケットでクライアントにチャレンジを送信するところから始まっています。ここでACKが行われ、クライアントは5番目のパケットで、WEPキーを使ってレスポンスを返却しています。

本来はここで認証に成功したという通知を受け取るはずですが、7番目のパケットには違う内容が見えます❶ (図11-15)。

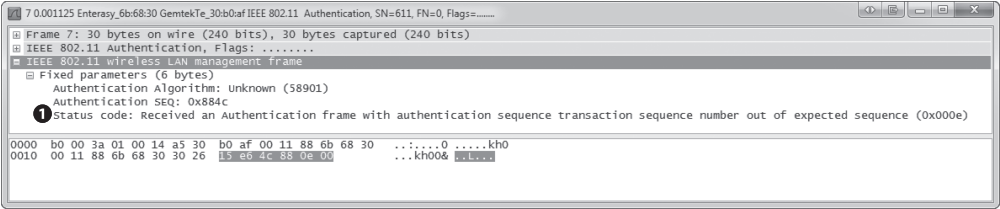


図11-15 認証に失敗したというメッセージ

メッセージには、チャレンジに対するクライアントのレスポンスが正しくないとあります。これはつまり、クライアントがチャレンジテキストの復号に使ったWEPキーが間違っていたため接続に失敗したことを意味します。正しいWEPキーで再度接続を試みなければなりません。

11.8.3 WPA認証の成功

80211-WPAauth.pcap

WPAはWEPとはまったく異なる認証機構を用いていますが、接続にキー入力を求める点は同じです。ファイル 80211-WPAauth.pcap に WPA 認証の成功例があります。

ファイルの最初のパケットは、アクセスポイントからのビーコンです。このパケットの 802.11 ヘッダを展開して、図 11-16 のように、[Tagged parameters] 行の下にある [Vendor Specific] 行を展開してみましょう。アクセスポイントの WPA の属性に関する項目が参照できるはずです❶。これによってアクセスポイントが WPA をサポートしていることや、サポートしているバージョン、実装がわかります。

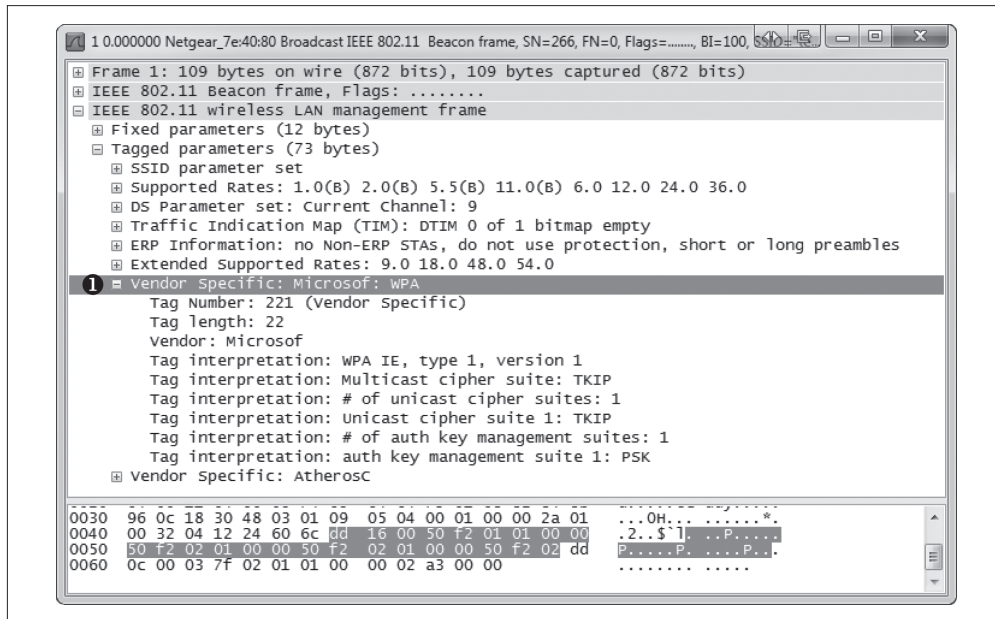


図11-16 ビーコンによってアクセスポイントがWPA認証をサポートしていることが確認できる

ビーコンを受信すると、クライアント (00:14:6c:7e:40:80) はアクセスポイント (00:0f:b5:88:ac:82) にプローブ要求を送信し、それに対してアクセスポイントがプローブ応答を行います。4番目から7番目のパケットで、認証、アソシエーション要求と応答が行われます。

8番目のパケットからいろいろなことが始まります。ここでWPAハンドシェイクが始まり、11番目のパケットまで続きます。図11-17でおわかりのように、WPAのチャレンジレスポンスがこのハンドシェイク処理で行われます。

No.	Time	Source	Destination	Protocol	Channel	Info
8	0.004096	Netgear_7e:40:80	Netgear_88:ac:82	EAPOL		Key
9	0.004101	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key
10	0.003580	Netgear_7e:40:80	Netgear_88:ac:82	EAPOL		Key
11	0.000004	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key

図11-17 これらのパケットはWPAハンドシェイクを構成する

図を見ると、チャレンジとレスポンスが2つずつありますが、802.1x Authenticationヘッダの下の[Replay Counter]フィールドによって、チャレンジとレスポンスが対になっています(図11-18)。最初の2つのハンドシェイクパケットのReplay Counter値は1❶、次の2つのハンドシェイクパケットの値は2❷になっています。

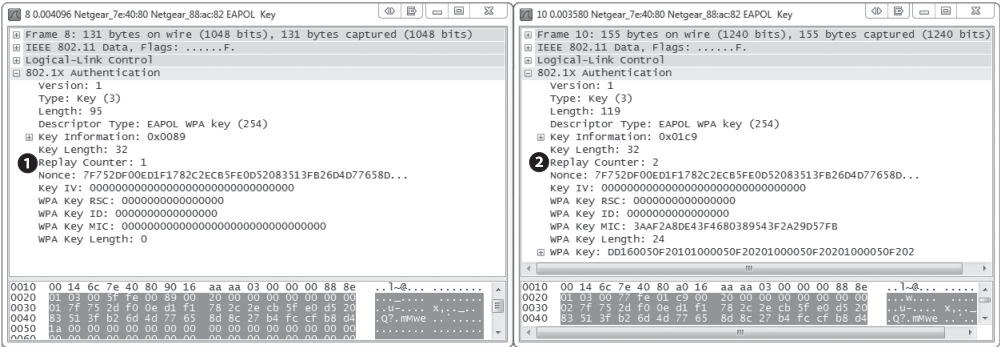


図11-18 [Replay Counter] フィールドによってチャレンジとレスポンスを対にできる

WPAハンドシェイクが終了し認証が成功すると、クライアントとアクセスポイント間でのデータの送受信が始まります。

11.8.4 WPA認証の失敗

80211-WPAauthfail.pcap

WEP同様、ユーザーがWPAキーを入力したにも関わらず、クライアントのユーティリティが接続できないというレスポンスを行ったところから見ていきましょう。このファイルが80211-WPAauthfail.pcapです。

先ほどと同じく、キャプチャファイルはWPA認証に成功した場合と同じように始まっています。ファイルにはプローブ、認証、アソシエーション要求が含まれています。WPAハンドシェイクは8番目のパケットで始まっていますが、なぜか認証に成功した場合の4個ではなく、8個のハンドシェイクパケットが存在しています。

WPAハンドシェイクの最初の2つのパケットが、8番目と9番目のパケットです。しかしここでは、クライアントからアクセスポイントへ送信されたチャレンジが間違っています。その結果同じ処理が10番目と11番目、12番目と13番目、そして14番目と15番目で繰り返されているのです(図11-19)。

[Replay Counter] 値を使えば、チャレンジとレスポンスを対にすることができます。

No.	Time	Source	Destination	Protocol	Channel	Info
9	0.003547	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key
10	1.000549	Netgear_7e:40:80	Netgear_88:ac:82	EAPOL		Key
11	0.000476	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key
12	0.999489	Netgear_7e:40:80	Netgear_88:ac:82	EAPOL		Key
13	0.000511	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key
14	0.999013	Netgear_7e:40:80	Netgear_88:ac:82	EAPOL		Key
15	-0.000037	Netgear_88:ac:82	Netgear_7e:40:80	EAPOL		Key

図11-19 EAPOLパケットの数がWPA認証の失敗を示している

ハンドシェイクプロセスが4回試行されると、通信が切断されます。図11-20のように、16番目のパケットでアクセスポイントから認証の失敗を通知されました。

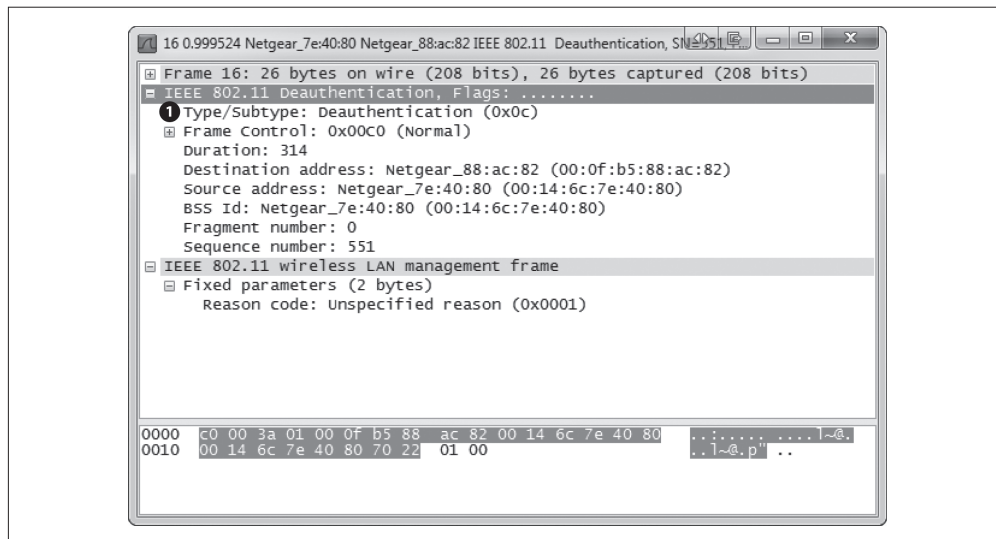
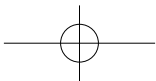
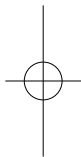
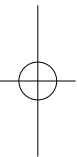
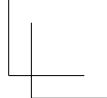
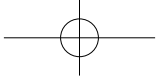


図11-20 WPAハンドシェイクが失敗し、認証が失敗した

11.9 まとめ

無線LANは現在もあまりセキュアではないと考えられていますが、さまざまな組織への普及が進んでいきます。無線LANへの移行とともに、有線LANと同様にデータのキャプチャと解析が行えるようになることが非常に重要になってきています。この章で学んだスキルと概念は完全なものではありませんが、パケット解析による無線LANのトラブルシューティングの難しさを理解するのに役立つはずです。



付録 A

USB ポートの通信キャプチャ

宮本 久仁男

付録Aは日本語版オリジナルの記事で、Wireshark を用いて USB デバイスとホスト間の通信をキャプチャする方法について解説します。「Wireshark = ネットワークパケットキャプチャ&解析ツール」だけではないのですが、Wireshark がそれ以外の用途でも使えることは意外に知られていません。本稿では、「それ以外」の用途のひとつとして、Wireshark による USB ポートの通信キャプチャを解説します。OS 環境は、Ubuntu 11.10 Desktop を使います。

A.1 Linux マシンにつなげた USB デバイスとの通信をキャプチャする

Wireshark では、Linux カーネルで usbmon が有効になっている場合に、当該カーネルが動作しているコンピュータの USB ポートにつなげられたデバイスの I/O 情報のキャプチャが可能です。

ここでは、入手が容易な USB メモリ (USB Mass Storage Device) をキャプチャ対象として、手順を説明します。

A.1.1 Wireshark をインストールする

Ubuntu に普通に Wireshark をインストールします。これで必要なソフトウェアが準備できます。

A.1.2 USB デバイスをつなげる

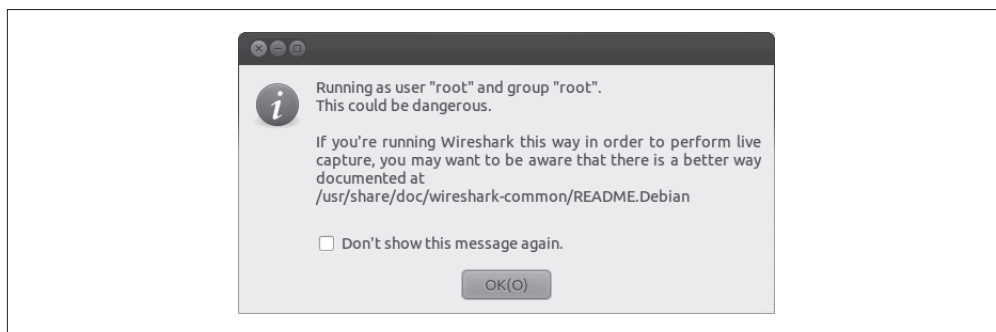
まず lsusb コマンドを実行して、デバイスの認識状況を確認します。今回の環境では以下のようになりました。基本的に Bus 001 の下に USB デバイスがぶらさがります。

```
wakatono@packman:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 002: ID 0483:2016 SGS Thomson Microelectronics Fingerprint Reader
Bus 001 Device 011: ID 0718:0638 Imation Corp.
```

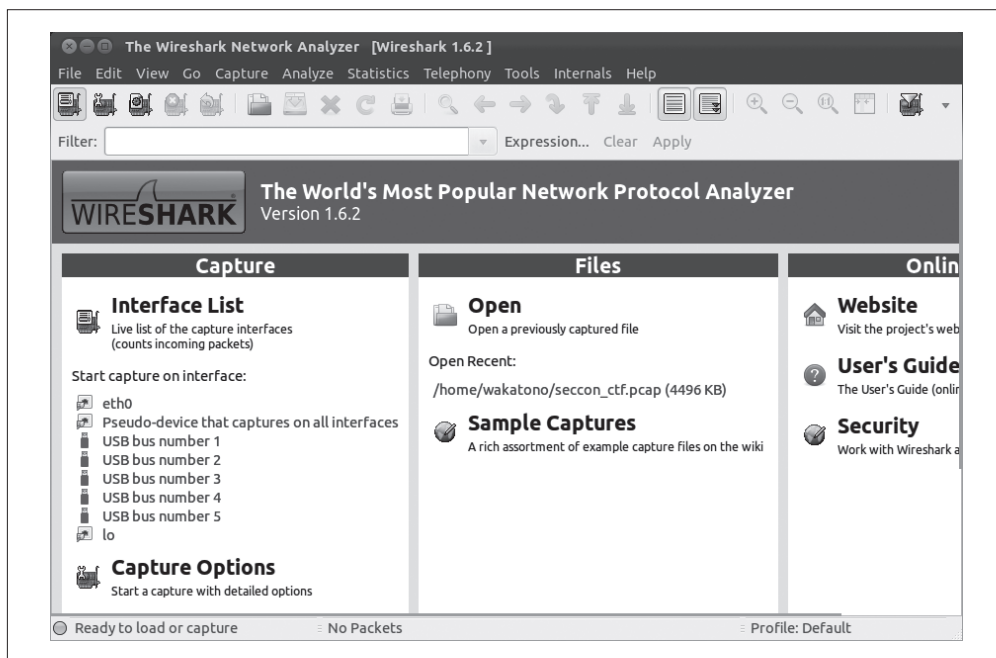
USBデバイスをつなげるとどのバスに接続されるのかについては、あらかじめ調べて把握しておいてください。

A.1.3 Wiresharkを起動し、USBインターフェイスを選択する

root権限でWiresharkを起動します。いくつか警告が表示されるかもしれませんが(図A-1)、気にせず起動しましょう。Wiresharkのウィンドウの左の部分に表示されている[Interface List]を見れば、キャプチャ可能なインターフェイスやバスなどがわかります(図A-2)。キャプチャ対象のデバイスが接続されているバスをクリックしてください。今回の場合は「USB bus number 1」です。



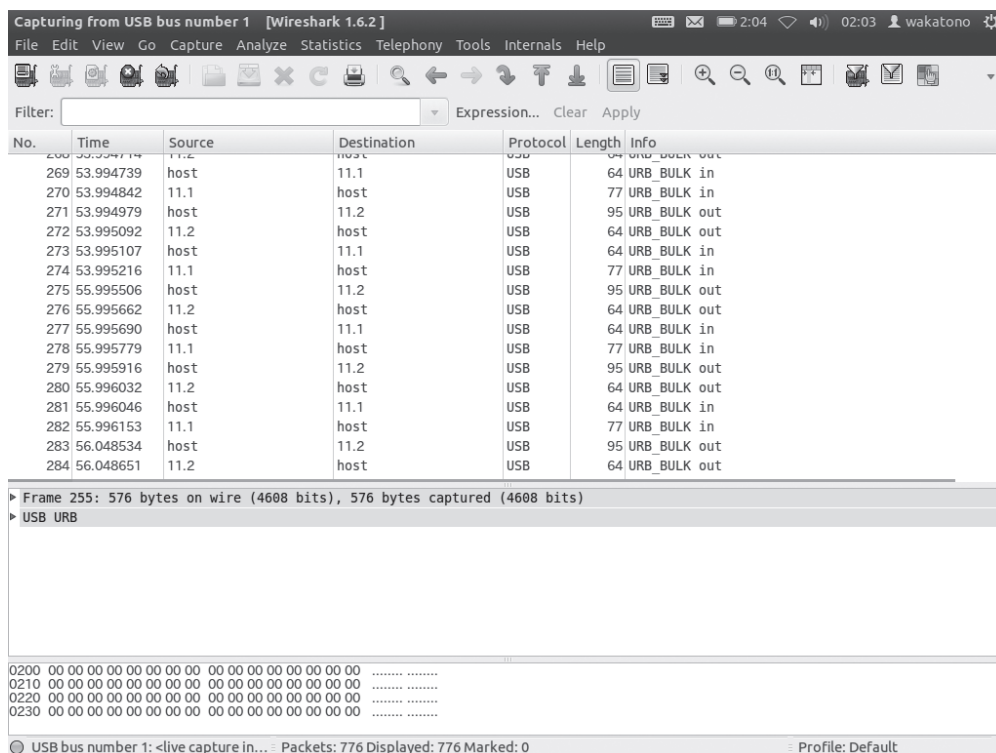
図A-1 root権限での起動に対する警告画面



図A-2 キャプチャ可能なインターフェース一覧

A.1.4 キャプチャ結果の確認

Wiresharkを起動し、キャプチャの開始後、USBメモリへのデータのコピーを開始します。コピーが完了したら、Wiresharkでのキャプチャを停止してキャプチャデータをファイルに書き出します。キャプチャファイルを開けば、USBデバイスとホストの間のI/O情報(コマンド類)を確認できます(図A-3)。ただし、取得されるのはファイルシステムの管理データとコマンドのやり取りのみです。ホストとデバイスの間でやり取りされた実際のコピーデータは、キャプチャファイルに含まれません。



図A-3 USBデバイスとホスト間のやりとり

A.1.5 他のLinux環境でキャプチャするには

USB通信のキャプチャ機能は、Wiresharkの機能というよりは、Wiresharkが用いているライブラリlibpcapの機能です。また、libpcapのバージョンによってはUSBキャプチャのためのパッチが必要なこともあります。このため、Linux環境でUSBデバイスのキャプチャを行うには、以下の条件を満たす必要があります。

1. usbmonが有効になったLinuxカーネルを用いる
2. USBキャプチャのためのパッチが適用されたlibpcapを準備する

3. 上記2で準備したlibpcapを用いる Wireshark を準備する

Ubuntu 11.10 Desktop は、この条件を標準カーネルおよび標準パッケージで満たしています。

A.2 Windowsで認識したUSBデバイスのI/Oをモニタする

Windows 版の Wireshark では、USB デバイスの I/O を直接モニタすることができません。しかし、Windows で認識した USB デバイスの I/O をモニタする方法はあります。

- VirtualBox 上で Windows を起動し、対象の USB デバイスをつなげる
- VirtualBox 上の Windows につながれた USB デバイスの USB ポートを、Linux で USB デバイスの I/O 情報をキャプチャするのと同じ方法で取得する

この場合、VirtualBox で物理的な USB デバイスを認識できるようにする必要があります。以降では、USB デバイスを VirtualBox で認識できるようにし、認識されたバスの I/O を Linux 上でキャプチャする方法について解説します。環境によって取得できる情報に若干違いはありますが、仮想マシン上とはいえ Windows の USB 通信を追跡することができるというのは有用です。

使用する環境は前節と同じです。VirtualBox 上にインストールするゲスト OS は Windows XP SP3、デバイスは今回も USB メモリを使用しました。

A.2.1 VirtualBoxをインストールする

Ubuntu のパッケージマネージャに任せてインストールします。

A.2.2 ユーザーを追加する

Linux 上のグループに、VirtualBox を動作させるユーザーを追加します。これを行わないと、USB ポートにつながれたデバイスを認識させることができません。今回は `/etc/group` を編集して、グループ (`vboxusers`) にユーザー (`wakatono`) を追加しました。

```
vboxusers:x:125:wakatono
```

A.2.3 ホストOSを再起動する

いったん OS を再起動して、ユーザー情報の変更を反映します。

A.2.4 VirtualBoxから認識させたいUSBデバイスをつなげる

USB デバイスを物理的につなげてください。ただし、USB メモリを挿入すると自動マウントされてしまうので、念のため USB メモリのマウントを解除しておきます。以下の例では、`umount /dev/sdb1` コマンドを実行しています (`/dev/sdb1` が USB メモリ)。

```
root@packman:~# mount
```

```

/dev/sda1 on / type ext4 (rw,errors=remount-ro,commit=600)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,nodev)
gvfs-fuse-daemon on /home/wakatono/.gvfs type fuse.gvfs-fuse-daemon
(rw,nosuid,nodev,user=wakatono)
/dev/sdb1 on /media/5017-E8DB type vfat (rw,nosuid,nodev,uid=1000,gid=1000,shortn
ame=mixed,dmask=0077,utf8=1,showexec,flush,uhelper=udisks)

```

A.2.5 VirtualBoxを起動し、当該デバイスをVirtualBox上から使えるようにする

VirtualBoxを起動します。管理しているOSの各種情報がOracle VM VirtualBoxマネージャ上に表示されるので、そこから「USB」を選択してデバイスフィルタを作成します。USBの設定画面で、「+」アイコンをクリックしてフィルタを設定してください（図A-4）。USBメモリがつながっていれば、デバイスが表示がされるのでチェックしてください（図A-5）。



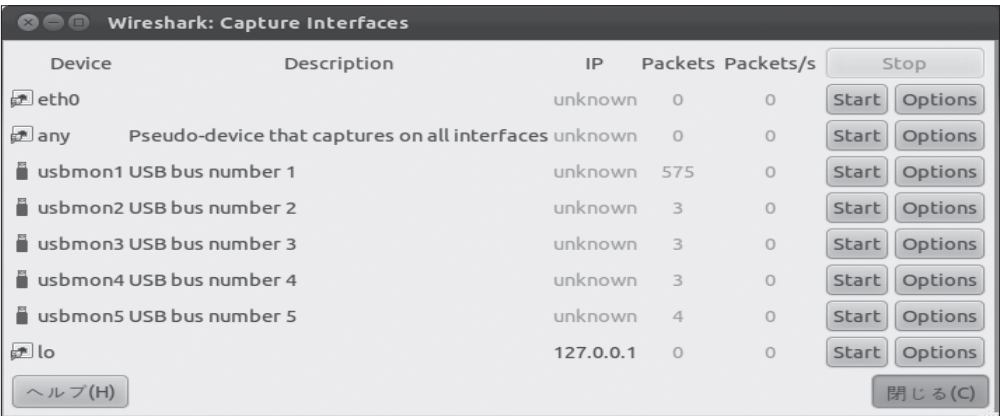
図A-4 USBの設定画面。「+」アイコンをクリックしてフィルタを設定する



図A-5 USBデバイスが追加された設定画面

A.2.6 WindowsとWiresharkを起動し、対象となるポートを選択してキャプチャを行う

Windowsを起動する以外は、前節とほとんど同じです。Wiresharkのメニューから [Capture] → [Interfaces] を選択し、キャプチャ可能なインターフェイス (図A-6) を表示し、キャプチャ対象のデバイスを一覧から選択してください。USBデバイスが接続されているバスは「usbmon1 USB bus number 1」です。

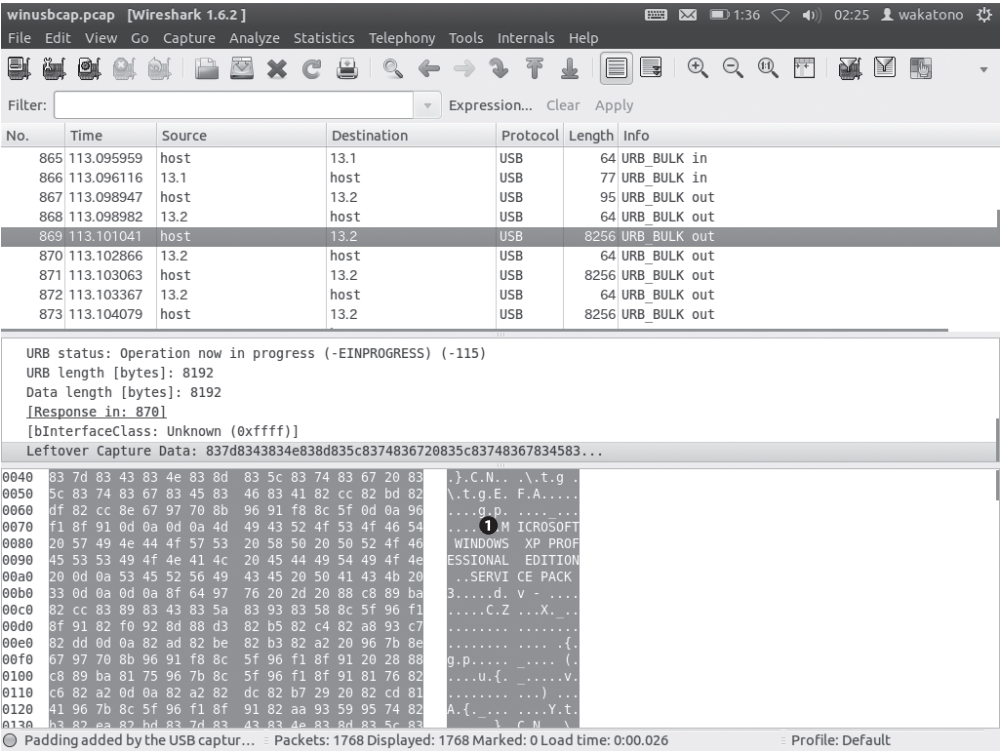


図A-6 キャプチャ可能なインターフェイスの一覧

この後、VirtualBox上で動いているWindowsからUSBメモリにファイルをコピーします。今回はWindowsのEULA.txt (C:¥Windows¥system32¥EULA.txt) をUSBメモリにコピーしてみました。

A.2.7 キャプチャ結果

図A-7に、Wiresharkでの確認結果を示します。EULA.txtに含まれている日本語（非ASCII文字）の部分は、Packet Bytesペインでうまく表示できていませんが、MICROSOFTから始まる文字列①から、USBメモリに送ったデータを取得できていることがわかります。



図A-7 WiresharkでキャプチャしたUSBデバイスとの通信

A.3 まとめ

本稿ではUSBバスを流れる情報のキャプチャ方法について解説し、どのようなデータが取得できるのかというのを例示しました。しかし、ここであげたのはあくまで一例にすぎません。どのようなデバイスを扱うかは読者次第です。

A.4 参考文献

CaptureSetup/USB - The Wireshark Wiki

<http://wiki.wireshark.org/CaptureSetup/USB>

付録B

pcap-ng 形式 ↔ pcap 形式 のデータ変換

宮本 久仁男

付録Bは日本語版オリジナルの記事です。Wireshark 1.8以降で標準のデータ形式となったpcap-ngですが、このデータ形式は以前から使われてきたpcapとは互換がありません。このため、pcap形式にしか対応していないツールでは、pcap-ng形式のデータをそのまま解析することはできません。本稿では、pcap-ng形式で保存されたデータをpcap形式に変換する方法、pcap形式で保存されたデータをpcap-ng形式に変換する方法を紹介します。

B.1 pcap形式とpcap-ng形式の概要

pcap形式は、tcpdumpで取得したパケットを取り扱うデータ形式として以前から使われてきました。以下は、pcap.hで宣言されているヘッダ構造体です。caplenとlenの2つがあるのは、実際に取得したデータ長と本来のパケット長が異なることがあるためです。

```
struct pcap_pkthdr {
    struct timeval ts;          /* time stamp */
    bpf_u_int32 caplen;        /* length of portion present */
    bpf_u_int32 len;           /* length this packet (off wire) */
};
```

以下は、Wireshark 1.2.11で宣言されている、パケットごとに付くヘッダ形式のひとつです。このヘッダが付加されるブロックはEnhanced Packet Blockと呼ばれます。実際にはこのヘッダにさらにブロック形式（32ビット値）と総ブロック長（32ビット値）が付加されますが、ブロック形式のところに0x00000006が入ると、パケットデータはEnhanced Packet Blockとして扱う必要があります。

```
typedef struct pcapng_enhanced_packet_block_s {
    guint32 interface_id;
    guint32 timestamp_high;
    guint32 timestamp_low;
    guint32 captured_len;
    guint32 packet_len;
    /* ... Packet Data ... */
    /* ... Padding ... */
};
```

```
/* ... Options ... */  
} pcapng_enhanced_packet_block_t;
```

以下は、同じくパケットごとに付くヘッダ形式の1つであり、このヘッダが付加されるブロックは Simple Packet Block と呼ばれます。ブロック形式のところに 0x00000003 が入ると、パケットデータは Simple Packet Block として扱う必要があります。

```
typedef struct pcapng_simple_packet_block_s {  
    guint32 packet_len;  
    /* ... Packet Data ... */  
    /* ... Padding ... */  
} pcapng_simple_packet_block_t;
```

pcap形式では、タイムスタンプの持ち方が `struct timeval` と宣言されています。一方 pcap-ng 形式の Enhanced Packet Block ヘッダでは、`timestamp_high` と `timestamp_low` の2つの値を用いて、1970年1月1日 (UTC) からマイクロ秒単位で経過した時間を64ビット値として保持するようになっています。前者はいわゆる2038年問題の影響を受けますが、後者は膨大な時間を扱えます ($(2^{64}-1)/(365(\text{day/year}) * 24(\text{hours/day}) * 60(\text{minutes/hour}) * 60(\text{seconds/minute}) * 1000(\text{milliseconds/second}) * 1000(\text{microseconds/millisecond}))$ を計算すればわかりますが、この形式では60万年近い時間を扱えます)。

B.2 pcapとpcap-ngでデータ変換が必要な理由

pcap-ng形式は最近になって使われ始めたデータ形式です。そのため、以前から使っているツールが対応していない場合もあります [文献1]。もちろん、それらのツールが pcap-ng に対応してくればよいのですが、使い慣れたツールから乗り換えるのは面倒です。また、snort [文献2] や ngrep [文献3] などのツールで、pcap-ng形式で取得したパケットデータを扱いたいということもあるでしょう。幸い、どちらの形式もパケットデータそのものはその中に保持しているので、ヘッダの変更を行うことで対応可能です。以下では、パケットデータを保持しつつ pcap形式と pcap-ng形式の間でデータ変換を行う方法について説明します。

B.3 変換方法

editcapはキャプチャデータを操作するための簡単なコマンドラインツールですが、このツールには形式を変換する機能もあります。本来ならば、pcap-ng形式から pcap形式への変換を最初に扱うべきですが、本稿では説明の都合上、pcap形式から pcap-ng形式への変換を最初に扱います。より詳細な情報が Wireshark Wiki [文献4] に記述されているので参照してください。

B.3.1 pcap形式 → pcap-ng形式

以下のように editcap コマンドを実行することで、pcap形式の `file.pcap` を、pcapng形式の

file.pcapngに変換できます。

```
editcap -F pcapng file.pcap file.pcapng
```

Wiresharkのコマンドライン版ともいえるtsharkコマンドでも、同様の変換処理を行えます。

```
tshark -F pcapng -r file.pcap -w file.pcapng
```

B.3.2 pcap-ng形式 → pcap形式

以下のようにeditcapコマンドを実行することで、pcapng形式のfile.pcapngを、pcap形式のfile.pcapに変換できます。

```
editcap -F libpcap -T ether file.pcapng file.pcap
```

pcap形式からpcap-ng形式への変換にはtsharkコマンドも使えますが、pcap-ng形式をpcap形式に変換する際にtsharkコマンドを使うことはできません。実行しようとしても、「tshark: The capture file being read can't be written in that format.」というメッセージが出力され、変換処理が行われずに終了してしまいます。

B.3.3 pcapやpcap-ngのファイルサイズが巨大になる場合

Wiresharkは、pcap形式やpcap-ng形式のファイルに保存されたパケットデータをすべてメモリに展開してから処理を行おうとします。そのため、ファイルが巨大になると、そのファイルをすべて読み込むだけのメモリが必要になります。メモリが不足するような場合にはプログラムが異常終了することもあります。一方でeditcapなどのプログラムは、逐次処理結果をファイルに書き出すように作られています。tsharkで試したところ、500MBのpcapファイルをWiresharkで読み込むだけで1分近い時間がかかり、変換結果をファイルに書き出すのに1分以上の時間がかかりました。一方で、editcapを同じ環境で試したところ、10秒かからないくらいの時間で変換が完了しました。

B.4 参考文献

[文献1] Wireshark and Pcap-ng

<https://blog.wireshark.org/2012/03/wireshark-and-pcap-ng/>

[文献2] Snort :: Home Page

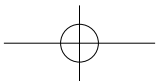
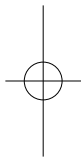
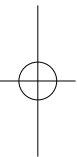
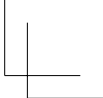
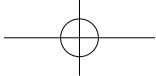
<http://www.snort.org/>

[文献3] ngrep - network grep

<http://ngrep.sourceforge.net/>

[文献4] Development/PcapNg (Wireshark Wiki)

<http://wiki.wireshark.org/Development/PcapNg>



付録C

推薦文献

本書で主に使用したツールはWiresharkですが、パケット解析を実行する場合は、一般的なトラブルシューティング、ネットワーク遅延、セキュリティの問題、無線ネットワークのいずれであっても、ここに示すツールが大いに役立つでしょう。この章では有用なパケット解析ツールと、パケット解析の学習に役立つ情報源とをご紹介します。

C.1 パケット解析ツール

Wiresharkに加え、パケット解析に有用なツールがいくつかあります。ここでは実際に使ってみて非常に役に立ったものをいくつか紹介します。

C.1.1 tcpdumpとWindump

Wiresharkは非常に人気がありますが、tcpdumpほどは普及していないでしょう。tcpdumpはCUIベースのツールですが、多くの人々の間でパケットキャプチャおよび解析ツールの業界標準とみなされています。

tcpdumpにグラフィック機能はありませんが、たとえばLinuxのsedやawkといったコマンドで出力をパイプするなど、大量のデータを扱う場合には非常に便利です。パケット解析を探求していくにつれ、Wiresharkとtcpdumpの両方を使い分けることになるでしょう。tcpdumpは<http://www.tcpdump.org/>でダウンロードできます。

WindumpはtcpdumpのWindows版で、<http://www.winpcap.org/windump/>でダウンロード可能です。

C.1.2 Cain & Abel

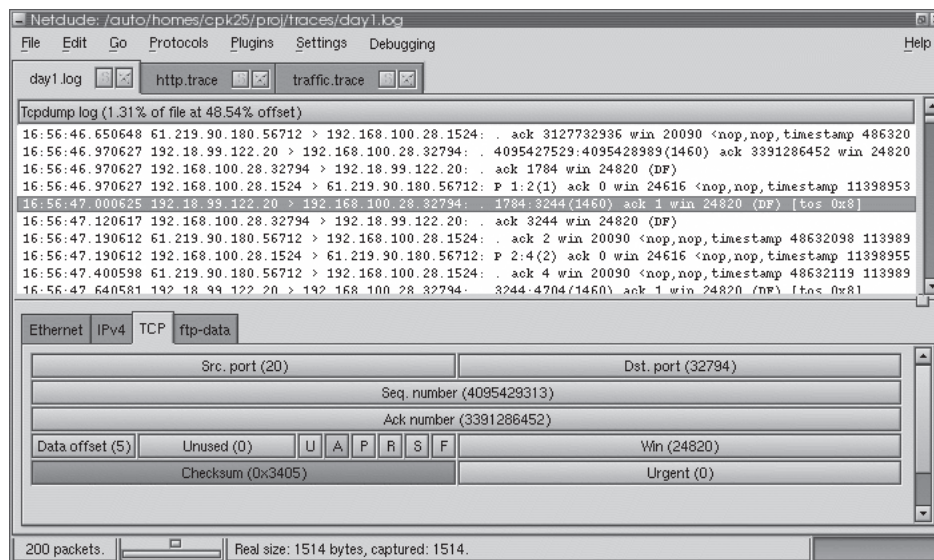
2章で説明したように、Cain & AbelはARPキャッシュポイズニングを実行するためのWindows用ツールです。Cain & Abelは非常に堅牢であり、ほかの用途でも利用できるでしょう。<http://www.oxid.it/cain.html>で入手できます。

C.1.3 Scapy

ScapyはPythonライブラリで、CUIのスクリプトを使ってパケットを作成、操作することができます。言ってみればScapyは、もっとも強力かつ柔軟性のあるパケット作成アプリケーションです。<http://www.secdev.org/projects/scapy/>でScapyの詳細やスクリプトのサンプル、またScapyそのものをダウンロードできます。

C.1.4 Netdude

Scapyほど高機能なものが必要でなければ、Netdudeが便利です (Linux向け)。Netdudeの機能は限られていますが、調査用のパケットを作成、修正する際に、非常に使いやすいGUIを備えています。図A-1はNetdudeの使用例です。Netdudeは<http://netdude.sourceforge.net/>でダウンロード可能です。



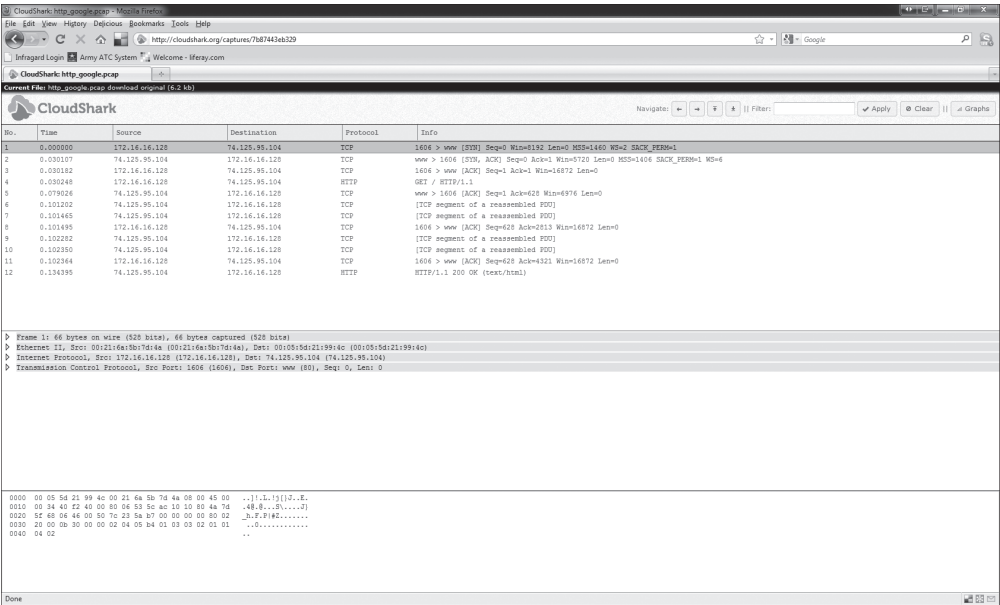
図A-1 Netdudeでパケットを修正する

C.1.5 Colasoft Packet Builder

WindowsユーザーでNetdudeのようなGUIが欲しい場合は、Colasoft Packet Builderがよいでしょう。Colasoftでも扱いやすいGUIでパケットの作成や修正が可能です。Colasoftは無料ソフトで、http://www.colasoft.com/packet_builder/から入手可能です。

C.1.6 CloudShark

QA Cafeが開発したCloudSharkは、パケットキャプチャをインターネット上で共有できるサイトです。ブラウザ内で、キャプチャファイルをWireshark風に表示することができます(図A-2)。キャプチャファイルをアップロードして、共有したい相手にリンクを送ることで、一緒に解析ができます。



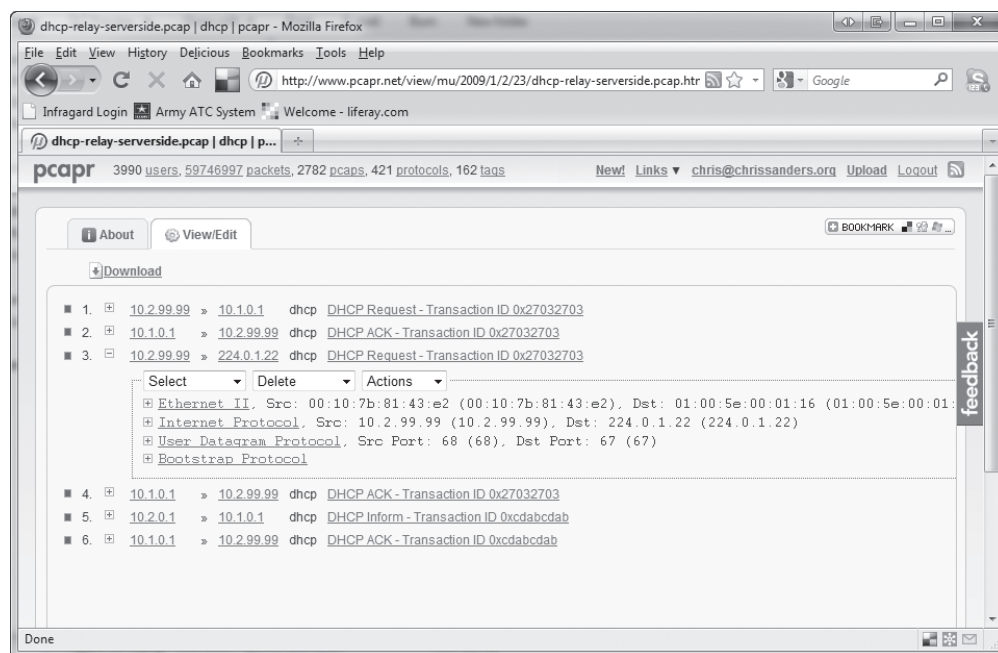
図A-2 CloudSharkで見たキャプチャファイルのサンプル

CloudSharkの気に入っているところは、登録が不要で、URLへのリンクから直接アクセスできるからです。つまり自分のブログにPCAPファイルへのリンクを投稿すると、誰かがそれをクリックするだけでパケットを見られるので、ファイルのダウンロードやWiresharkで開く手間が省けます。

CloudSharkのサイトは<http://www.cloudshark.org/>です。

C.1.7 pcapr

pcaprはMu Dynamicsが開発した、PCAPファイル共有のための非常に堅牢なWeb 2.0プラットフォームです。本書執筆時点で、pcaprには約3,000のPCAPファイルがあり、400以上のさまざまなプロトコルのサンプルが含まれています。図A-3はpcaprにおけるDHCPトラフィックキャプチャのサンプルです。



図A-3 pcaprでDHCPトラフィックのキャプチャを見る

通信のサンプルを探すときは、まずpcaprで探します。たくさんのキャプチャファイルを持っているなら、<http://www.pcapr.net/>にアップロードして共有しましょう。

C.1.8 NetworkMiner

NetworkMinerは、主にネットワークフォレンジックに使われるツールですが、それ以外のさまざまな場面でも役に立ちます。パケットキャプチャにも使えますが、PCAPファイルの解析に本領を発揮します。NetworkMinerはPCAPファイルを取り出し、OSごと、またホスト間のセッションごとに分割します。キャプチャから直接ファイルを抽出することもできます。NetworkMinerは無料ソフトで、<http://networkminer.sourceforge.net/>で入手できます。

C.1.9 Tcpreplay

ネットワーク上にパケットを再転送し、デバイスがどう反応するかを調べたいときには、いつもTcpreplayを使っています。TcpreplayはPCAPファイルを取り出し、その中に含まれているパケットを再転送するために設計されています。<http://tcpreplay.synfin.net/>からダウンロード可能です。

C.1.10 ngrep

Linuxに慣れているなら、データの検索には普通grepを使うでしょう。ngrepはgrepとよく似てい

ますが、PCAPデータに対して非常に的を絞った検索ができます。私はフィルタがうまく機能しない場合や、複雑になりすぎたときに使っています。http://ngrep.sourceforge.net/にngrepの詳細が掲載されています。

C.1.11 libpcap

高度なパケット解析、あるいはパケットを処理するアプリケーションを開発したいなら、libpcapに詳しくなることです。簡単に言えば、libpcapはネットワークトラフィックのキャプチャのための、ポータブルなC/C++ライブラリです。Wireshark、tcpdumpをはじめとする多くのパケット解析アプリケーションが、libpcapライブラリを使っています。libpcapについてはhttp://www.tcpdump.org/を参照してください。

C.1.12 hping

hpingは、多用途に使えるツールで、コマンドラインでパケットの作成、転送が可能です。さまざまなプロトコルをサポートしており、簡単に使うことができます。http://www.hping.org/でダウンロード可能です。

C.1.13 Domain Dossier

ドメインやIPアドレスの登録情報を検索する必要があるときに利用するのが、Domain Dossierです。http://www.centralops.net/co/DomainDossier.aspxでアクセスすることができます。

C.1.14 PerlとPython

PerlとPythonはツールではなく、スクリプト言語です。パケット解析に熟練してくると、ニーズを満たす自動化ツールが存在しないケースに出くわします。こうした場合のツールを作成できる言語が、PerlとPythonなのです。私はたいがいのアプリケーションにPythonを使っていますが、どちらを選ぶかは好みの問題です。

C.2 パケット解析に役立つ情報源

Wiresharkのホームページ、学習コース、ブログに至るまで、パケット解析に役立つ情報源は数多くあります。そのうちで私のお気に入りをご紹介します。

C.2.1 Wiresharkホームページ

Wiresharkに関連するすべてが見つかるのがこのWebサイト、http://www.wireshark.org/です。ここにはソフトウェアのドキュメント、キャプチャファイルのサンプルを含む非常に有益なWikiや、Wiresharkメーリングリストへの登録に関する情報があります。

C.2.2 SANS Security Intrusion Detection In-Depth Course

SANSのメンターである私の意見は少々偏っているかもしれませんが、SANS SEC 503、Intrusion Detection In-Depthよりも優れたパケット解析のコースは存在しないと思います。セキュリティ専門でなくても、コースの最初の2日間は、パケット解析とtcpdumpの入門編として、経験上最適です。

コースを指導しているのは、パケット解析のヒーロー的な存在であるMike PoorとJudy Novakの2人で、年に数回ライブイベント形式で行われています。旅費が限られているなら、オンラインやWebのオンデマンド形式でも受講が可能です。

SEC 503とその他のSANSコースについては、<http://www.sans.org/>を参照してください。

C.2.3 Chris Sandersのブログ

十分な量とは言えませんが、パケット解析に関する記事を、ときどき自分のブログ<http://www.chrissanders.org/>に投稿しています。私が執筆したほかの記事や書籍のポータルとしても役立つだけでなく、私と連絡を取る方法も掲載しています。

C.2.4 Packetstanブログ

パケット関連のブログで現在一番のお気に入りには、Mike PoorとJudy Novakのブログです。<http://www.packetstan.com/>には、興味深いトラフィックの分析が載っており、またここにあるコンテンツのすべてが秀逸です。

C.2.5 Wireshark University

Laura Chappellはもっとも熱心なWireshark伝道者のひとりです。彼女のサイトにはWiresharkを使う際のヒントや、彼女が執筆した書籍「Wireshark Network Analysis」に関する情報、また指導するコースなどが掲載されています。<http://www.wiresharktraining.com/>を参照してください。

C.2.6 IANA

IANA (Internet Assigned Numbers Authority) <http://www.iana.org/>は、北米のIPアドレスとプロトコル番号を管理している組織です。Webサイトではポート番号の検索やトップレベルドメインに関する情報、RFCの検索や閲覧ができるサイトの一覧など、貴重な情報が掲載されています。

C.2.7 TCP/IP Illustrated (Addison-Wesley) [†]

多くの人々にTCP/IPの聖書と考えられているのが、Richard Stevens博士によるこのシリーズ本で、パケットを専門とする人なら必ず本棚にあるはずです。私の一番のお気に入りのTCP/IP関連書籍であり、本書執筆時にも参考にしました。

[†] 監訳注：邦訳『詳解TCP/IP』ピアソン・エデュケーション刊。

C.2.8 The TCP/IP Guide (No Starch Press)

TCP/IP関連でもうひとつのお気に入りですが、Charles Kozierokによるこの本です。1000ページ以上にわたり、非常に詳しく、たくさんの図表が掲載されています。

索引

数字

0x90	125
16進数からバイナリへの変換	216

A

ActiveX コントロール	207
add_group_user	100, 311
add_user コマンド	100, 311
Adobe Flash のゼロデイ脆弱性	124
Adobe の Collab.collectEmailInfo 脆弱性	153
airbase-ng コンポーネント	202
Aircrack-ng の Web サイト	202
airmon-ng start wlan0 コマンド	202
APACHE_SERVER オプション	150
Apache Tomcat への攻撃	288
API	
Meterpreter	267
Windows	111
Armitage	12
armitage コマンド	12
ARP ポイズニング	196
Autopwn ツール	61
autorun.inf	175
auxiliary モジュール	8, 137-148
~の使用	140
~の分析	144
Auxiliary::Scanner ミックスイン	35
AVG Anti-Virus	115

B

back コマンド	64
background コマンド	275, 311, 315
Back Track	297, 298

～の更新	304, 305
------------	----------

Base64	115, 213, 217-219
Binary-to-Hex	187, 194
bind_tcp 形式	128
Blowfish 暗号化アルゴリズム	178
browser_autopwn サーバー	201
Burp Suite	281
bypassuac	315

C

check コマンド	308
CIDR (Classless Inter-Domain Routing)	25
cld	320
clearev コマンド	311
client.framework.payloads.create(payload) 関数 ..	264
cmd 変数	212
cmd_exec() 関数	268
cnt カウンタ	218
covert (秘密)	4
Collab.collectEmailInfo 脆弱性	153
Conficker ワーム	65
Convert::ToByte	217
CRLF	217
CVE (Common Vulnerabilities and Exposures) ...	47

D

data/templates/template.exe テンプレート	118
db_autopwn コマンド	61, 62,
db_connect コマンド	22, 26, 46, 53, 61, 309
db_import コマンド	23
db_nmap コマンド	26, 309
db_status コマンド	23
debug コマンド	216
Defcon 18 Hacking Conference	209

def exploit 行	215
def inject 関数	263
def powershell_upload_exec 関数	216
DEP (Data Execution Prevention)	71
DHCP サーバー	199
dhcpcd.conf	200
DistCC	291
DNS (Domain Name System)	18, 199
download コマンド	310
drop_token コマンド	310

E

egg hunter	228
EHLO コマンド	243
EIP レジスタ	239, 241, 243, 244
Encase	294
-EncodedCommand	217
ESP レジスタ	239, 240
ESSID	202
Ettercap	196
eventlog_clear() 関数	268
eventlog_list() 関数	268
event_manager ツール	294
evil 文字列	231
Excellent	62
exe コマンド	216
execute -f コマンド	310
execute_upload.rb	270
exploit コマンド	74, 77, 103, 107, 111, 275, 308
exploit モジュール	8
exploit -e コマンド	308
exploit -h コマンド	308
exploit -j コマンド	308
exploit -z コマンド	308

F

Fast-Track	181-198
file_local_digestmd5() 関数	268
file_local_digestsha1() 関数	268
file_local_digestsha2() 関数	268
file_local_write() 関数	268
Framework の機能の実装	244
[FTP (File Transfer Protocol)] サービス	299
FTP スキャン	32
ftp_version モジュール	32
fuzzed 変数	223

G

generate_seh_payload 関数	255
generic/debug_trap ペイロード	231, 243

getgui スクリプト	285
getprivs コマンド	311
getsystem コマンド	97, 310, 314
getuid コマンド	97

H

h2b 変換機能	217
hashdump コマンド	92, 93, 311
hashdump ポストエクスプロイトモジュール	92
help コマンド	9, 309
hex-blob	209
host_process.memory.allocate 関数	263
host_process.memory.write 関数	263
host_process.thread.create 関数	263
hosts コマンド	23
HTTP GET リクエスト	40
HTTP PUT メソッド	289

I

ICMP	21
IDS (Intrusion Detection System)	14, 254, 255
idx カウンタ	218
iexplore.exe	127, 131, 260
IFRAME インジェクション	164
IFRAME リプレースメント	167
IIS (Internet Information Services)	299
IMAP ファザー	222
Immunity Debugger	126, 127, 129
impersonate_token コマンド	310
INC ECX 命令	233
include Msf::Exploit::Remote::BrowserAutopwn: ディレクティブ	202
Incognito	315
incognito コマンド	99
Infectious Media Generator	174
info コマンド	68, 307
'INJECTHERE'	183
insecure.org	285
INT3	246, 247
Intel x86 アーキテクチャ	125
Internet Explorer Aurora エクスプロイト	129
IP アドレスの検索	19
ipidseq スキャン	25
IPS (Intrusion Prevention System)	20, 123
irb シェル	111, 267
is_admin?() 関数	268
is_uac_enabled?() 関数	269

J

Java アプレット攻撃	150, 156
JavaScript の出力フォーマット	13
JMP 命令セット	240
JMP ESP 命令セット	240

K

KARMA	199
karma.rc	200
Karmetasploit	199-207
Kerberos トークン	98, 100
keylog_recorder モジュール	91
keyscan_dump コマンド	310
keyscan_start コマンド	310
keyscan_stop コマンド	310
killav スクリプト	105

L

LAN Manager (LM) ハッシュ	93
LHOST	67, 73, 95, 103, 272, 307
lib/msf/core/exploit/http.rb	145
Linux 仮想マシンの起動	298
Linux システム	
～に対する攻撃	102
～上でハッシュをダンプする	315
LIST コマンド	221, 223
list_tokens -g コマンド	310
list_tokens -u コマンド	99, 309
LPORT	67, 97, 103, 272
ls コマンド	309

M

MailCarrier エクスプロイト	241
MailCarrier 2.51 SMTP	240
mailcarrier_book.rb	243
make_nops() 関数	247
Man-Left-in-the-Middle 攻撃	167
McAfee アンチウイルスソフト	89
MD5 チェックサム	268
MessageBoxA 関数	111
Metasploit	
Nmap の結果をインポートする	23
～インターフェイス	8
スクリプティング	259-277
データベース利用	22
～モジュールの調査	211
ユーティリティ	13
～を使ったポートスキャン	27

Metasploitable システムのスキャン	286
Metasploit Auxiliary モジュール	137-148
Metasploit Express	15
Metasploit Framework (MSF)	7
～へのエクスプロイトの移植	239-257
Metasploit Pro	15
Meterpreter	83
Microsoft SQL Server への攻撃	85
Microsoft SQL Server へのブルートフォース ...	86
Windows XP マシンの侵害	83
xp_cmdshell	88
キーストロークのキャプチャ	90
偽装トークン	98
基本コマンド	89
権限昇格	95
パスワードのダンプ	91
パスワードハッシュのダンプ	92
パスワードハッシュの抽出	92
ピボットリング	101
ポストエクスプロイトモジュール	108
ユーザー名のダンプ	91
Meterpreter コマンド	309
Meterpreter スクリプト	104, 270
Meterpreter ポストエクスプロイトコマンド	314
Meterpreter ミックスイン	268
Meterpreter API	267
Microsoft SQL Server	
～でコマンド実行	210
～の探索	30
～へのブルートフォース	86
～への攻撃	85
Microsoft SQL Server インジェクション	182
migrate コマンド	91
migrate モジュール	105
migrate PID コマンド	309
MS08-067	28, 65-67, 70, 71, 73, 109
ms08_067_netapi モジュール	10, 65
MS11-006	134
MSF → Metasploit Framework	
msf プロンプト	65
Msf::Auxiliary::Scanner ミックスイン	35
msfbook データベース	22, 26
msf.doc	134, 135
Msf::Exploit::Remote::Seh ミックスイン	252, 253
Msf::Exploit::Remote::Tcp ミックスイン	35, 242
Msf::Exploit::Remote::Udp ミックスイン	252, 253
MSFcli	9
コマンド	313
msfcli コマンド	10

msfcli -h コマンド	10	スキャンを実行する	52
MSFconsole	8	設定	49
～からNmapを実行する	26	レポート	52
コマンド	307	netcat	36, 39
～のカスタマイズ	283	Netcraft	19
msfconsole コマンド	9	net localgroup administrators metasploit /ADD コマンド	211
MSFencode	14	netstat -an コマンド	129
コマンド	312	net user コマンド	95
～でエンコードする	115	NeXpose	40
msfencode コマンド	115	MSFconsole内で実行する	47
msfencode -h コマンド	116, 312	New Report ウィザード	44
msfencode -l コマンド	14, 312	New Site ウィザード	42
MSFpayload	13	Scan Now ウィザード	43
コマンド	311	設定	41
～でスタンドアロンバイナリを作成する	114	レポートをMetasploit Frameworkに インポートする	46
msfpayload -h コマンド	14	nexpose_connect -h コマンド	48
msfpayload -l コマンド	311	nexpose_scan コマンド	48
msfpescan -p コマンド	230	Nmap	
msfupdate コマンド	304, 305	～でのポートスキャン	20
::Msfrpc::Util::EXE.to_win32pe	272	～によるポートスキャン	83
Msfrpc::Util::EXE.to_win32pe(framework,payload. encoded)	216	～の結果をMetasploitにインポートする	23
MSFvenom	122, 313	～をMSFconsoleから実行する	26
MSSQL Bruter	186, 188	Nmap スキャン	24
攻撃オプション	188	nmap -A コマンド	21
mssql_commands.rb	212	nmap -Pn コマンド	21
mssql_exec モジュール	211	nmap -sS コマンド	21
MSSQL Injector	182	NOP	240
mssql_login モジュール	87	～シェルコードを解析する	126
mssql_payload エクスプロイト	213	～を確認する	125
mssql_payload モジュール	88, 89	NOPスライド	125, 240
mssql_ping モジュール	85, 86	～の削除	247
mssql_powershell モジュール	209	notepad.exe	174
mssql_powershell.rb	213, 215, 219	NSEH (Next SE Handler)	253
mssql.rb	212, 216, 219	nslookup	19
Multi-Attack Web Method	169	NT AUTHORITY\SYSTEM	97
multi/handler モジュール	107, 275	NTLM (NT LAN Manager)	92
multi/http/tomcat_mgr_deploy エクスプロイト	289	NTLMv2 (NT LAN Manager v2)	92
multi_meter_inject スクリプト	259	NX (No Execute)	69
Muts	251		
N		O	
nasm_shell.rbユーティリティ	14	Offset	247
NASM シェル	14	oledlg.dll	254
NAT (Network Address Translation)	28	OpenSSH	32, 287
Nessus	49	open_x11 スキャナ	60
Metasploit内からスキャンする	54	OSINT (Open Source Intelligence)	18
結果をMetasploit Frameworkに インポートする	53	overt (公開)	4
スキャンポリシーを作成する	50		

P

packetrecorder コマンド	106
pattern_offset.rb	227
Pass-the-Hash 攻撃	94
payload.encoded 関数	248
payload.exe	96
.pcap ファイル	106
.pde ファイル	176
PDF ファイル	151
PE (Portable Executable) 形式	114
persistence スクリプト	106
PID (プロセス ID)	260
ping コマンド	21
PoC (Proof of Concept)	13
POP3 サービス	204
POP-POP-RETN 命令シーケンス	228, 230, 232, 253
portscan/syn モジュール	28
POST パラメータ攻撃	184
Postfix	288
PostgreSQL	22
PowerShell	213
powershell_upload_exec 関数	216
print_error() 関数	267
print_good() 関数	267
print_line() 関数	267
print_status() 関数	267
priv 拡張機能	97
ProFTPD	287
ps コマンド	90, 91, 98, 99, 309
PTES (Penetration Testing Execution Standard) ...	1
PureBasic	59
PUT メソッド	289

Q

Quick TFTP Pro 2.1	250, 251
--------------------------	----------

R

Railgun アドオン	111
rand_text_alpha_upper バッファ	247
Rapid7	41
RATTE (Remote Administration Tool Tommy Edition)	178
RDP (Remote Desktop Protocol)	285
reboot コマンド	311
reg コマンド	310
regedit	108
registry_createkey() 関数	269
registry_deleteval() 関数	269

registry_delkey() 関数	269
registry_enumkeys() 関数	269
registry_enumvals() 関数	269
registry_getvaldata() 関数	269
registry_getvalinfo() 関数	269
registry_setvaldata() 関数	269
resource コマンド	79
resource karma.rc コマンド	203
resource.rc ファイル	79
rev2self コマンド	98, 310
reverse_tcp ペイロード	67, 73, 74, 312
RHOST	67, 73, 95, 103, 139, 307
RO コミュニティ文字列	33
robots.txt	142
route コマンド	102
route add コマンド	103
route print コマンド	102
RPC サービス	65
RSA	124
RT73 チップセット	202
Ruby	209
Ruby シェル	111
run_batch(batch) メソッド	35
run checkvm コマンド	315
run getcountermeasure コマンド	315
run getcountermeasure -h コマンド	315
run getcountermeasure -d -k コマンド	315
run get_local_subnets コマンド	101
run hashdump コマンド	105
run_host(ip) メソッド	35
run killav コマンド	314
run migrate コマンド	314
run packetrecorder コマンド	106
run persistence コマンド	108
run post/	109
run_range(range) メソッド	35
run scraper コマンド	106
run screen_unlock コマンド	104
run scriptname コマンド	104, 309
run vnc コマンド	104, 315
RW コミュニティ文字列	33

S

sa アカウント	86, 88, 89, 187, 210
SAM データベース	96, 314
Samba エクスプロイト	75
save コマンド	70
scanner ミックスイン	35
scraper スクリプト	106

screenshot コマンド	89, 310
search コマンド	64
search <i>name</i> コマンド	307
search scanner/http コマンド	140
SEH → 構造化例外ハンドラ	
service_change_startup() 関数	269
service_create() 関数	269
service_delete() 関数	269
service_info() 関数	269
service_list() 関数	269
service_start() 関数	270
service_stop() 関数	270
sessions -c <i>cmd</i> コマンド	309
sessions -i 1 コマンド	74, 197
sessions -i <i>sessionid</i> コマンド	97
sessions -K コマンド	309
sessions -l コマンド	74, 97, 197, 308
sessions -l -v コマンド	74, 308
sessions -s <i>script</i> コマンド	309
sessions -u コマンド	109
sessions -u 1 コマンド	110
sessions -u <i>sessionID</i> コマンド	309
SET → Social-Engineer Toolkit	
set コマンド	68
set autorunscript migrate -f コマンド	308
setdesktop コマンド	310
set <i>function</i> コマンド	307
setg コマンド	70
setg <i>function</i> コマンド	308
SET Interactive Shell	178
set LHOST コマンド	73
set payload <i>payload</i> コマンド	308
set target <i>num</i> コマンド	308
SET Web-GUI	178
SHA-1 チェックサム	268
SHA-256 チェックサム	268
shell コマンド	310, 315
SHELL32.DLL	244
shell_reverse_tcp ベイロード	114
shikata_ga_nai エンコーダ	14, 117, 308
show advanced コマンド	308
show auxiliary コマンド	64, 138, 307
show exploits コマンド	63, 307
show options コマンド	64, 215, 308
show payloads コマンド	66, 307
show targets コマンド	67, 308
simple_tcp.rb	36
[Site Cloner] オプション	158, 162, 165, 167, 171
SMB スキャン	29
SMB ログイン検証	56
smb_login モジュール	56
SMBPass 変数	95
smb_version モジュール	29
SMTP	151, 240, 288
sniffer_dump	311
sniffer_interfaces	311
sniffer_start	311
sniffer_stats	311
sniffer_stop	311
SNMP	33, 299
SNMP スイープ	33
Social-Engineer.org	149
Social-Engineer Toolkit (SET)	149-179
Web 攻撃ベクター	156
～の設定	150
標的型フィッシング攻撃ベクター	151
sock.put コマンド	242, 243
'Space'	229
SQL インジェクション攻撃	182
SQL インジェクター	182, 184
SQLPwnage	191
SQL Server 認証	186
SRVHOST	131
SRVPORT	131
SSL (Secure Socket Layer)	35
SSH (Secure Shell)	31
ssh_version モジュール	31
steal_token コマンド	99, 310
sudo	274
surgemail サービス	225
surgemail.exe	228, 230
SVN の証明書	304
svn update コマンド	150, 304
SYN ポートスキャナ	28
sysadmin の SQL ロール	210
sysinfo コマンド	90, 309

T

TCP アイドルスキャン	24
TCP ポート	30, 40, 85, 86, 243, 300
Teensy USB HID 攻撃ベクター	175
TFTP (Trivial File Transfer Protocol)	250-256
THREADS	25, 28, 29, 139
timestamp コマンド	293, 311
toor パスワード (PostgreSQL)	298
Total size	229
Trojan バックドア	139
Twitter	144

U

UAC (User Account Control)	274, 2
～のダンプ	92
～の抽出	92
バックカー	121
バックグラウンドに移行	315
ハッシュ値	322
バッファオーバーフローの移植	240
バナグラビング	40
ヒープ	125
ヒープスプレー	124
ヒープベース攻撃	77
ピボットリング	27, 101
秘密ペネトレーションテスト	4
標的型フィッシング攻撃ベクター	151
ファイルフォーマットエクスプロイト	134
ファイルフォーマットのバグ	134
ファジング技術	221
ファジングテスト	221
ファジング用の文字列	223
フィンガープリンティング	5
不正な文字	14, 234-236
ブラウザベースのエクスプロイト	124
ブルートフォース	
Microsoft SQL Serverへの～	86
ブルートフォース攻撃	77
ブレイクポイント	232
プロセスID (PID)	260
プロセス間の移動	105
ペイロード	7
～をステルスに立ち上げる	120
別ネットワークへの攻撃	101
ペネトレーションテスト	1
～のタイプ	4
変数が大文字	69
報告書の作成	3
ポートスキャン	
Metasploitを使った～	27
Nmap	20
ポストエクスプロイト	3, 285
ポストエクスプロイトモジュール	108
～の一覧表示	109
ホワイートハットテスト	4

ま行

マスキラントサイド攻撃	195
マルチエンコーディング	117
ミックスイン	35

Auxiliary::Scanner	35
Meterpreter	268
Msf::Exploit::Remote::Seh	252, 253
Msf::Exploit::Remote::Tcp	35, 242
Msf::Exploit::Remote::Udp	252, 253
scanner	35
無線LANカード	202
無線LAN攻撃ベクター	178
モジュール	8
～の作成	213

や行

ユーザーアカウント制御 → UAC	
ユーザー名の収集	164
ユーザー名のダンプ	91
読み取り／書き込みコミュニティ文字列	33
読み取り専用コミュニティ文字列	33

ら行

ランダム化	246
リスナー	8
リソースファイル	79
リターンアドレスの取得	230
リバースエンジニアリング入	317
リバースシェル	7
リモートGUI (VNC) を取得	315
リモートコード実行	234
レジストリキー	108

●著者紹介

Chris Sanders (クリス・サンダース)

ケンタッキー州の公立学校にネットワーク管理者として勤務。さまざまな技術を駆使しながら日々の業務をこなしている。Windows 2003 Server、Active Directory、グループポリシー、Microsoft Virtual Server、無線LAN管理、ネットワークセキュリティについての専門知識を有する。ユニークビジターが一日に61万5千人以上いる人気サイトThe TechGenix networkのWindowsNetworking.comにネットワーク管理者向けのTIPS記事を寄稿している。

ブログ：<http://www.chrissanders.org/>

メール：chris@chrissanders.org

ツイッター：[@chrissanders88](https://twitter.com/chrissanders88)

●監訳者紹介

高橋 基信 (たかはし もとのぶ)

1970年生まれ。1993年早稲田大学第一文学部哲学科卒。同年NTTデータ通信株式会社(現・株式会社NTTデータ)に入社。入社後数年間Unix上でのプログラム開発に携わった後、オープン系システム全般に関するシステム基盤の技術支援業務に長く従事する。Unix、Windows両OSやインターネットなどを中心とした技術支援業務を行う中で、接点ともいべきネットワーク分野のトラブルシューティングに不可欠な技術であるパケット解析についての造詣を深める。現在はNTTデータ先端技術株式会社にてWindows関連の基盤システム構築業務を行いながらオープンソースのSambaを中心とした出版および講演活動を行っている。こうした出版や講演の傍ら、長年の趣味である音楽を楽しんでいる。

宮本 久仁男 (みやもと くにお)

株式会社NTTデータ 技術開発本部 セキュリティ技術センター所属。NTTDATA-CERTメンバー。1991年電気通信大学卒、同年NTTデータ通信株式会社(現・株式会社NTTデータ)へ入社。各種開発やシステム運用および支援業務、セキュリティ推進等のスタッフ業務を経て、現在はセキュリティに関する研究開発に従事。2011年3月に、情報セキュリティ大学院大学博士後期課程修了。博士(情報学)。クライアントセキュリティ技術や仮想マシン技術、ネットワーク技術に強い興味を持つが、技術的に面白いと感じれば何でも興味の対象になり得る。主な著訳書として『WebDAV システム構築ガイド』(共著、技術評論社)、『実用SSH 第2版』(共訳、オライリー・ジャパン)、『実用 Subversion 第2版』『実践 Metasploit』(監訳、オライリー・ジャパン)、『欠陥ソフトウェアの経済学 ―その高すぎる代償―』(監訳、オーム社)があるほか、雑誌等への寄稿は多数。セキュリティ&プログラミングキャンプ講師(2004～2011)、同実行委員(2008～2011)、Microsoft MVP for Enterprise Security(2004～2012)、SECCON-CTF実行委員。

●訳者紹介

岡 真由美 (おか まゆみ)

フリーランスのライター兼翻訳家。電波新聞社で雑誌編集、海外特派員を経験後、フリーに。主にIT分野関連の取材・執筆、翻訳業に携わる。訳書に『技術とイノベーションの戦略的マネジメント』(共訳、翔泳社)、『POWER+UP ― 米国オタクゲーマーの記したニッポンTVゲーム興隆の軌跡』(コンピュータエージ社)、『「ヒットする」のゲームデザイン ― ユーザーモデルによるマーケット主導型デザイン』『実践 Metasploit』(オライリー・ジャパン)などがある。

実践 パケット解析 第2版

—— Wiresharkを使ったトラブルシューティング

2012年11月21日 初版第1刷発行

著 者	Chris Sanders (クリス・サンダース)
監 訳 者	高橋 基信 (たかはし もとのぶ)、宮本 久仁男 (みやもと くにお)
訳 者	岡 真由美 (おか まゆみ)
発 行 人	ティム・オライリー
編 集 協 力	有限会社風工舎
印 刷・製 本	株式会社平河工業社★★★★未定★★★★
発 行 所	株式会社オライリー・ジャパン 〒160-0002 東京都新宿区坂町26番地27 インテリジェントプラザビル1F Tel (03)3356-5227 Fax (03)3356-5263 電子メール japan@oreilly.co.jp
発 売 元	株式会社オーム社 〒101-8460 東京都千代田区神田錦町3-1 Tel (03)3233-0641 (代表) Fax (03)3233-3440

Printed in Japan (ISBN978-4-87311-569-6)

乱丁本、落丁本はお取り替え致します。

本書は著作権上の保護を受けています。本書の一部あるいは全部について、株式会社オライリー・ジャパンから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。