=== ch08.docx

[chap]8章　基本的な現実世界のシナリオ

この章からは、Wiresharkを使って現実のネットワーク問題を分析しつつ、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。また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、実際の問題を取り上げます。それぞれの問題を取り巻く状況を解説するとともに、その時点でアナリストが利用可能だった情報を提供します。基本を提示したら、適切なパケットをキャプチャするための手法を説明し、解析プロセスを段階を踏んで解説します。解析が完了したら、問題の解決方法、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。

解析は非常にダイナミックなプロセスであり、それぞれのシナリオの解析に用いる手法は、読者の手法とは異なるかも知れません。解析手法は人それぞれです。一番大切なのは、解析の結果問題が解決するか、その経験から学習することなのです。また本章で説明する問題の大半は、パケットスニッファを使わずに解決できるかも知れません。初めてパケット解析の方法を学んだとき、パケット解析テクニックを使って一般的な問題に取り組むことが役に立ったので、こうしたシナリオを提示しているのです。

パケットレベルでのソーシャルネットワーキング

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。各サービスの認証プロセスを調べ、2つの非常によく似たSNSがどのように異なる方法を使い、同じタスクを実行しているかを見ていきます。またそれぞれのサービスの主要な機能を見て、日々の活動から生じているトラフィックのさらなる理解を深めていきましょう。

Twitterトラフィックをキャプチャする

テクノロジーコミュニティの最新ニュースを得るため、あるいは彼女の愚痴をこぼすためにしても、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。

注　Webサイトのコードは頻繁に変わります。そのため次の項のキャプチャを再度作成しようとすると、ここに示したものとは結果が異なる場合があります。

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、そのログインプロセスからのトラフィックのキャプチャです。これには2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、ネット上を行きかうプレインテキストのパスワードから、日々の活動がどれだけ安全でないかに気付いてもらうためです。

幸いにもTwitterの認証プロセスにはそれほど問題はありません。図8-1から分かるように、最初の3つのパケットは、ローカルのパソコン（172.16.16.128）①とリモートサーバ（168.143.162.68）②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しています③。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、SSLトラフィックであると推測できます。

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2つのパーティー間での通信を暗号化し、解読するのに使う文字列を使用しています。ハンドシェイクプロセスはホスト間での鍵の正式なやりとりであると同時に、さまざまな接続と暗号文字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。

データ交換を扱う暗号化されたパケットを見つけるには、パケット詳細のペインのInfoカラムで[Application Data]とされ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部分を開くと、[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①が含まれているはずです。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　暗号化された信用情報が転送される

16番目のパケットでFIN/ACKによる接続のティアダウンプロセスが始まるまで、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で分かるように、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番ポートで新たな接続をセットアップ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（/）を求めるHTTP GETリクエストが行われます②。サーバは24番目のパケットでリクエストを確認し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。

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

ホームページに関連する画像やその他のファイルを検索するため、さらに何回かのGETリクエストがキャプチャファイルで実行されます。

ツイートでのデータ送信

ログインしたら、今度はみなさんが何を考えているかを伝える番です。私は本を執筆中ですので、こんなふうにツイートします。「これは実践パケット解析第2版のためのツイートです」。そしてポスティングから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てからすぐ始まっています。ローカルのワークステーション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から始まります。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パケットで構成されています。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番目のパケットのパケット詳細のペインに置いています。

このHTTPヘッダを見るため、図8-4のように5番目のパケットのパケット詳細のペインのHTTP部分を開くと、URL /status/updateでPOSTメソッドが使われているのが分かります①。ホストフィールドに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アップデートのための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のに気づくでしょう。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ステータスフィールドが見えます。

This+is+a+tweet+for+practical+packet+analysis%2c+second+edition

ステータスフィールドの値は、暗号化されていないプレインテキスト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るので、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　プレインテキストのツイート

Twitterのダイレクトメッセージ

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、ユーザーが個人的なメッセージを共有できるものです。ファイルtwitter\_dm.pcap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からも分かるように、ダイレクトメッセージはプライベート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プレインテキスト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。

ここで分かるのは、Twitterは周囲を遮断するものではないので、プライベートなTwitterメッセージを使って重要なデータを送信するのは考え直した方がいい、ということです。

Facebookトラフィックをキャプチャする

今度はWiresharkを使ってFacebookのトラフィックをキャプチャし、解析してみましょう。

Facebookのログインプロセス

ファイルfacebook\_login.pcapにキャプチャした、ログインプロセスから始めましょう。図8-7のように、信用情報が送信されるとすぐにキャプチャが始まります。Twitterのログインプロセスと同様に、443番ポートでTCPハンドシェイクが行われます①。ワークステーション172.16.0.122②が、認証プロセスを扱うサーバ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SLLハンドシェイクが起こり④、ログインのための信用情報が送信されます。

図8-7　ログインの信用情報がHTTPSによってセキュアに送信される

Twitterの認証プロセスとの違いは、ログイン信用情報が送信されたあとに、認証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でハイライトしているように、12番目のパケットのHTTPヘッダに/home.phpのGETリクエストがあるのが見えます①。

図8-8　認証の後、/home.phpのGETリクエストが行われる

認証に使われる接続は、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に見られるように、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。

図8-9　HTTP接続、次いでHTTPS接続がティアダウンされる

Facebookのプライベートメッセージ

ログイン認証プロセスのあとは、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。ファイルfacebook\_message.pca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トラフィックから構成されています。図8-10のように2番目のパケットのHTTPヘッダを開くと、長いURLストリングと一緒にPOSTメソッドが使われています①。見てのとおり、ストリングにはAJAX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検索する、インタラクティブな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アプローチです。プライベートなメッセージをクライアントのブラウザへ送信すれば、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）と思うかも知れませんが、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インタラクティブ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]部分を開けば、プライベートメッセージの中身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。

図8-11　Facebookのメッセージのコンテンツもプレインテキスト

TwitterとFacebookの比較

TwitterとFacebookという2つのWebサービスの認証とメッセージの手法が、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。プログラマーなら、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。一方セキュリティリサーチャーは、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知れません。また認証接続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2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人のプログラマーが同じタスクを実行するルーチンを開発した場合、違いが起こりう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、実際に比較が可能なトラフィックを実体験することです。こうした基本は、同じ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。

ESPN.comのトラフィックをキャプチャする

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となっています。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サイトをブラウズ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。それぞれの接続と異常を識別するのに、ファイル全体を手動でスクロールし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プロセスを簡単にしてみましょう。

[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、データのソースを知っていると便利ですが、さらに重要なのは、ソースがひとつなのか複数なのかを把握しておくことです。Wiresharkの[Conversations]ウィンドウ（[Statistics]から[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対話、25のTCP接続、14のUDP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、ずいぶんたくさんのことが行われています。

図8-12　複数の接続を表示している[Conversations]ウィンドウ

[Protocol Hierarchy Statistics]ウィンドウの利用

この状況をさらによく見るため、TCP接続とUDP接続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。図8-13のように、[Statitics]から[Protocol Hierarchy]を選択、[Protocol Hierarchy Statistics]ウィンドウを開きます。

図8-13　プロトコルを表示する[Protocol Hierarchy Statistics]ウィンドウ

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知れませんが、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（データ転送とコントロールパケット）。UDPヘディング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]ウィンドウのUDPヘディングの下を見ると、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クエリは独自に存在しないので、キャプチャ内にあるその他のトラフィックはHTTPトラフィックのみです。ESPNサイト内のHTMLコードはDNS名で他のドメインまたはサブドメインを参照しているため、複数のクエリを実行していると推測できます。

この論理を裏付ける証拠を探してみましょう。

DNSトラフィックを見る

フィルタを作成すれば、DNSトラフィックを簡単に見ることができます。図8-14のように、DNSトラフィックをすべて表示しているWiresharkウィンドウのフィルタ部分に「dns」と入力しましょう。

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トラックは、クエリとレスポンスのように見えます。検索されたDNS名をよく見るには、そのクエリのみを表示するようフィルタを作成します。フィルタを作成するには、パケット一覧のペインでクエリを選択、パケット詳細のペインでパケットのDNSヘッダを開きます。次に[Flags：0x0100（標準クエリ）]フィールドを右クリックし、[Apply as Filter]にマウスを持っていき、[Selected]を選択します。

これでdns.flags == 0x0100フィルタが起動してクエリだけが表示されるようになるので、解析している記録がずっと読みやすくなります。また図8-14からも分かるようにクエリは14個あり（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、またはホームページに表示されるコンテンツに関連しているのが分かります。

HTTPリクエストを見る

最後にHTTPリクエストを調べ、これらのクエリのソースを検証します。そのためには[Statistics]から[HTTP]を開いて[Requests]を選択、[Create Stat]をクリックします（これを実行する前に、作成したばかりのフィルタをクリアにするのを忘れずに）。

図8-15は[HTTP Requests]ウィンドウを示しています。ここに表示されている14の接続は（各行が特定ドメインへの接続を表しています）、DNSクエリによってまとめられています。

図8-15　ウィンドウにまとめられたすべてのHTTPリクエスト。アクセスされたドメインを表す。

これだけ多くの接続が起きていると、関わりの深いプロセスがどれかを迅速に判断するのがベストです。もっとも簡単なのはトラフィックのサマリーをみることです。[Statistics]から[Summary]を選択しましょう。図8-16に示した[Summary]ウィンドウが、プロセス全体を約2秒で表示してくれます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とサブドメインへのリクエストへと分割され、さまざまなサーバと接触、それでもこのプロセス全体に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つつその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面白い練習になります。パケットを見ない限り、データが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]ウィンドウがプロセス全体を2秒以内に表示してくれる

現実での問題

今度は問題のあるトラフィックの例に移ります。さまざまなインターネットアクセス問題と、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。

インターネットに接続できない：設定問題

ユーザーがインターネットに接続できないというのが、最初のシナリオです。他のワークステーションとの共有や、ローカルサーバでホストされているアプリケーションへのアクセスを含む、内部ネットワーク上のリソースにアクセスできることは確認されています。

すべてのクライアントとサーバがスイッチ経由で接続されていることからもわかるように、ネットワークアーキテクチャはかなり単純な作りになっています。インターネット接続はデフォルトのゲートウェイとして機能するひとつ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。

ケーブルにもぐりこむ

問題の原因を判断するため、ユーザーにインターネットをブラウズしてもらっている最中に、スニッファを使います。ページ○○（実際のページ数を入れる）からの情報を使い（図2-15）、スニッファを配置するのに適切な場所を判断します。

われわれの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。テストの実施のためにすでにユーザーの邪魔をしてしまっているので、再度オフラインになるのも我慢してもらいましょう。その結果のファイルがnowebaccess1.pcapです。

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のユーザーコンピュータは、1番目のパケットで、そのデフォルトのゲートウェイである172.16.0.10のIPアドレスに関連する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。

図8-17　デフォルトゲートウェイを求める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、ゲートウェイへのルートが判明しました。

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を使い、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に示したように、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①。

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①。

図8-19　2回目のDNSクエリが4.2.2.1に送信

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。このプロセスは図8-10から分かるように、プライマリのDNSサーバ①とセカンダリ②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。プロセス全体にかかる時間は約8秒で③、これが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かかる時間となります。

図8-20　通信が停止するまでDNSクエリが繰り返される

パケットをもとに、問題の原因を特定しましょう。まず、デフォルトの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、この通信機器はオンラインであり、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クエリは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。つまり問題を正しく解決するには、ルータがDNSクエリを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。しかもWebページそのもののリクエストにHTTPが利用される前にこれらが行われなければなりません。

ネットに接続できないユーザーが他には存在しないので、ネットワークルータとリモートDNSサーバはおそらく問題の原因ではないといえます。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を受け取らずに、手動でアドレス情報が設定されていたため、デフォルトのゲートウェイアドレス設定が不正になっている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正しくなかったため、DNSクエリパケットが転送できなかったのです。

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、ユーザーに与えた影響は多大です。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、相当な時間がかかってしまいます。規模の大きな複雑な問題を解決するだけが、パケット解析ではないのです。

このシナリオでは、ゲートウェイルータのIPアドレスが分かっていなかったので、Wiresharkは問題を正確に識別できませんでしたが、どこをチェック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。ゲートウェイルータを調べ、ISPに連絡を取り、リモートDNSサーバのトラブルシューティングを行うリソースを探すことなく、問題の原因であるパソコンそのものに集中することができたのです。

注　ネットワークのIPアドレススキームに慣れてくると、解析がさらに早く行えるようになります。ARPリクエストが送信されたIPアドレスがゲートウェイルータのアドレスと違うと気づけば、この問題はすぐに識別できたのです。こうした単純な設定ミスはしばしばネットワーク問題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。

インターネットに接続できない：望まないリダイレクト

このシナリオでも、ワークステーション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/に接続できないのです。Google>がホストするドメインにアクセスしようとすると、「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。しかも問題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さなネットワークを使っています。

ケーブルにもぐりこむ

解析を始めるため、ユーザーには<http://www.google.com/>をブラウズしてもらい、タップを使って生成したトラフィックを待ち受けます。その結果のファイルがnowebaccess2.pcapです。

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のユーザーコンピュータは、1番目のパケットで、ホストのIPアドレス172.16.0.102に関連する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。このアドレスをすぐに認識することはできませんでした。

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であると分かりました。先ほどのシナリオ同様、これはゲートウェイルータのアドレスだとし、パケットはこのアドレスを使って外部DNSサーバに再度転送されると推測してみましょう。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つのホスト間で新たなTCP接続を行う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①。

図8-22　内部ホストから別の内部ホストへと送られたTCP SYNパケット

また172.16.0.102③では、通常HTTPトラフィックに使われる80番ポートでTCP接続の試みが行われています②。図8-23に示したように、この接続の試みは、ホスト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を応答として送信した際に、突然停止しています。

6章で、TCP接続を終了するのに、RSTフラグをセットしたパケットを使ったことを思い出してください。しかしながら今回のシナリオでは、172.16.0.8のホストは、172.16.0.102のホストと80番ポートでTCP接続を確立しようとしました。ところがこのホストでは、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いため、TCP RSTパケットが送られて接続が終了してしまったのです。このプロセスは2回繰り返されています。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がセットされた応答レスポンスが戻ってくるからです。ここでユーザーのブラウザには、ページが表示できませんというメッセージが表示されます。

図8-23　TCP SYNへの応答として送られたTCP RSTパケット

図8-24　TCP SYNとRSTパケットは合計3回送られている

きちんと機能しているもうひとつ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に気づきました。ARPリクエストがゲートウェイルータのMACアドレスではなく、ほかの不明な機器に送られていたのです。ARPリクエストとレスポンスを追跡すれば、[www.google.com](http://www.google.com)に関連するIPアドレスを見つけるために、DNSサーバに送信した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クエリの生成が妨げられる理由は2つあります。

・接続を開始した通信機器のドメイン名とIPアドレスがすでにDNSキャッシュにマップされている。

・ドメイン名に接続している通信機器のドメイン名とIPアドレスがそのホストファイルにすでにマッピングされている。

クライアントのコンピュータをさらに調べてみると、コンピュータのホストファイルに、内部IPアドレス172.16.0.102と結びついた[www.google.com](http://www.google.com)のエントリが見つかりました。ユーザーの問題はこの間違ったエントリにあったのです。

コンピュータは一般に、ドメイン名とIPアドレスのマッピングの正式な情報源としてホストファイルを利用し、外部ソースに問い合わせる前にそのファイルをチェックします。今回のシナリオでは、ユーザーのコンピュータはホストファイルをチェック、[www.google.com](http://www.google.com)のエントリを見つけ、[www.google.com](http://www.google.com)はローカルネットワーク上にあると判断しました。次にARPリクエストをホストに送ってレスポンスを受けとり、172.16.0.102と80番ポートでTCP接続しようとしたのです。しかしリモートシステムがWebサーバとして設定されていなかったため、接続が成立しませんでした。

ホストファイルのエントリを削除すると、ユーザーのコンピュータは正しく通信を開始し、[www.google.com](http://www.google.com)に接続できるようになりました。

注　Windowsシステムでホストファイルを調べるには、C:\Windows\System32\drivers\etc\hostsを開きます。Liuxの場合はview/etc/hostsを開きます。

こういった問題は非常によく起こります。悪意あるコードが埋め込まれたWebサイトにユーザーをリダイレクトするのに何年もの間使われてきたマルウェアもこれを悪用したものです。攻撃者がホストファイルを操作し、オンラインバンキングへアクセスすると、毎回口座情報を盗みだす偽サイトへリダイレクトされているとしたらどうでしょう！

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機能と、その遮断の仕方の両方を学びます。今回のシナリオでは、外部からの制限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問題を調べると、IPアドレスが不明なこと、また通信プロセスの鍵となるDNSが見当たらないことも分かります。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。

インターネットに接続できない：アップストリーム問題

先の2つのシナリオ同様、今回のシナリオでも、ユーザーがワークステーション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。

ケーブルにもぐりこむ

問題のトラブルシューティングのために、まず<http://www.google.com>をブラウズしてトラフィックを生成します。問題はネットワーク全体に広がっている、つま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スニッフィングすることはできません。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。サービスがいったん中断されたのち、完全に受動的な状態になれるためです。タップによるキャプチャのファイルはnowebaccess3.pcapです。

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トラフィックで始めます。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応答が含まれているので、ARPプロセスはすでに始まっていて、ゲートウェイルータのMACとIPアドレスマッピングはすでにホスト172.16.0.8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、ホスト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①で、これはDNSパケットです②。パケットのコンテンツを調べると、[www.google.com](http://www.google.com)のAレコードのクエリであることがわかります③。

図8-25　[www.google.comのAレコードのDNS](http://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。パケット詳細のペインを調べると、このリクエストに応答したネームサーバが、クエリに対し複数の回答を提供していることが分かります①。この時点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レコードとDNSレスポンス

ユーザーのワークステーションがWebサーバのIPアドレスを確認したので、サーバとの通信の試み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①。2番目のパケットに、DNSクエリレスポンスで提供された、最初のAレコードからの宛先アドレスが含まれています。TCPパケットにはSYNフラグがセットされており②、リモートサーバと80番ポートでの通信を試みています③。

図8-27　80番ポートでの接続開始を試みる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。図8-28に示したように、このプロセスは約1秒後に再度起こっており、ここで通信が停止し、ブラウザがWebサイトが見つからないと報告していたのです。

図8-28　応答がないため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、ネットワーク内のワークステーションが外部に接続できるのは分かっています。DNSサーバからの応答に含まれるアドレスも有効で、ホストはそのアドレスのひとつへの接続を試みています。また接続に用いているローカルワークステーションも機能しているようです。

問題なのは、リモートサーバが接続リクエストに答えない、つまりTCP RSTパケットが送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。ローカルのパケットフィルタはないと仮定すると、ほかの問題の可能性はリモートネットワークにあるということになり、お手上げとなります。この場合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。

学んだこと

今回のシナリオの問題は、こちら側では修正できないものでした。解析の結果、問題はネットワーク上のホストでも、ルータでも、名前解決サービスを提供する外部DNSサーバでもないと判明したのです。

問題の原因が自分たちにないと分かると、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に体面が救われます。自分のせいじゃないと主張するISPやベンダ、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。

不安定なプリンタ

ITヘルプデスク管理が、印刷問題にてこずっています。セール部門のユーザーは、セールスで使っているプリンタに問題があると報告しています。大量のプリンタジョブを送ると、数ページ印刷しただけで停止してしまうといいます。ドライバ設定を何度も変えてみましたが、うまくいきませんでした。ヘルプデスクスタッフはこれがネットワーク問題ではないかどうか、確認してほしいと言っています。

ケーブルにもぐりこむ

これはプリンタの問題なので、できるかぎりプリンタに近いところにスニッファを設定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レイヤ3スイッチなので、ポートミラーリングが使えます。プリンタが接続し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、プリンタに大量のプリンタ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。

解析

図8-29からわかるように、ネットワークワークステーション間のTCPハンドシェイクがプリンタジョブを送り（172.16.0.8）、プリンタ（172.16.0.253）がキャプチャファイルの頭で接続を始めています。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①。データ量は、パケット一覧のペインのInfoカラムの一番右端か、パケット詳細のペインの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もうひとつのパケットが送られ①、プリンタによって認識されました②。

図8-30　正常なデータ転送とTCPによる認識

最後の2個のパケットまで、データのフローが続きます。121番目のパケットはTCP再送パケットで、図8-31が示すように、これがトラブルの最初のサインです。

図8-31　TCP再送パケットは問題の兆し

ある通信機器がリモート機器へとTCPパケットを転送したのに、そのリモートパケットが転送を認識しない場合、TCP再送パケットが送られます。再送の閾値に達すると、送信元はリモート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。このプロセスは通信が停止するまで数回繰り返されます。

今回のシナリオでは、プリンタが送信されたデータを認識できなかったために、クライアントのワークステーションがプリンタに再送パケットを送っています。TCPヘッダのSEQ/ACK解析部分を開き、その下の追加情報を見れば（図8-31）①、これがなぜ再送なの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③。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②。

パケット間の遅延を解析するとき、状況に合わせて時間表示フォーマットを変えることができます。ここでは先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、[View]から[Time Display Format]を選び、[Second Since Previous Captured Packet]を選びます（図8-31）。するとオリジナルのパケット（120番目のパケット）が送信されてから5.5秒後に121番目のパケットが再送されているのがはっきり分かります①。

図8-32　パケットの送信間隔を知るとトラブルシューティングに役立つ

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。パケット一覧のペインの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分かります。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停止しています。

今回のシナリオでは、自分のネットワーク内のワークステーション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、プリンタがワークステーションへの応答を停止しているのに気づきます。再送からも分かるように、ワークステーションはデータを送信しようと最善の努力をしていますが、プリンタが応答をやめてしまっているのです。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。大量のプリンタジョブが送られてもその数ページ分しか印刷しないのは、メモリの一部の領域にアクセスしている可能性があります。ここではメモリ問題によって、プリンタが新しいデータを受け取ることができず、ホストによるプリンタジョブの通信を止めてしまっていたのでした。

学んだこと

このプリンタ問題はネットワーク問題ではありませんが、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問題としています。幸いなのはTCPが、2つの機器が通信をやめてしまったという有益な情報を残してくれたことです。

今回は通信が突然停止したとき、TCPが内蔵する再送機能のおかげで、問題の箇所を正確に指摘することができました。より複雑な問題を解決する場合でも、このような機能に頼ることが頻繁に起こります。

立ち往生する支社

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サーバベースのドメインと、セカンダリのドメインコントローラを採用しています。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。ドメインコントローラはセカンダリのDNSサーバで、本社のDNSサーバから、リソースレコード情報を受け取るようになっています。

デプロイメント担当チームが支社に新インフラの導入を行う際、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

支社

ワークステーション　172.16.16.101

支社のルータ

支社のスレーブDNSサーバ　172.16.16.251

WAN

本社

本社のルータ

本社のアプリケーションサーバ　172.16.16.200

本社のマスターDNSサーバ　172.16.16.250

ケーブルにももぐりこむ

問題は本社と支社間の通信にあるので、問題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、これらコンピュータのうちの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、それを使ってデータ収集するほかの場所を決めます。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。

解析

図8-34のように、アドレス172.16.16.101のワークステーションのユーザーが、本社のアプリケーションサーバ172.16.16.200でホスト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。これは本社の172.16.16.200にあるサーバのDNS名です。

図8-35から分かるように、このパケットへの応答がサーバ障害①で、これはDNSクエリに妨げがあったことを示しています。エラー（サーバ障害）のため、このパケットはクエリに回答していません②。

これで通信問題とDNSに何らかの関係があ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　AレコードのDNSクエリで始まる通信

図8-35　クエリレスポンスがアップストリーム問題を示している

支社のDNSサーバから適切なトラフィックをキャプチャするため、スニッファを設置し、ポートミラーリングの設定を変更して、ワークステーションのトラフィックではなく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。この追加のパケットはちょっと妙な感じです。標準DNSサーバの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。

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

支社にあるDNSサーバは本社のDNSサーバのスレーブ、つまりリソースレコードを受け取るのに本社サーバに依存していることになります。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、ここでのDNS問題が、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。ゾーン転送が失敗した理由を見つけることで、さらに1段階進みましょう。犯人はオフィス間のルータか、本社のDNSサーバそのものであるというところまで絞り込めます。本社DNSサーバのトラフィックをスニッフして、SYNパケットがサーバへ送られているかどうかを見てみます。

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した。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妨害し、支社のクライアントが本社からクエリの応答を受け取れなくしていたのです。

学んだこと

犯罪ドラマを見れば、ネットワーク通信問題の捜査についてかなり学習できます。犯罪が起きると、刑事はまず被害者の取材に着手します。それで手がかりを得て、さらに調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（ワークステーション）の調査から始め、DNS通信問題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、最終的には問題の原因であるルータまでたどりつけたのです。

解析を行う場合、パケットをヒントだと考えるようにしましょう。ヒントは誰が罪を犯したかは教えてくれませんが、最終的には犯人まで導いてくれます。

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。

今回のシナリオでは、ある開発者が、複数の店舗の売り上げを追跡し、中央データベースへと報告を戻すアプリケーションを開発したとしましょう。通常の営業時間中の帯域をセーブしたいので、これはリアルタイム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挿入されます。

ところがこの新開発の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、データベースに挿入さ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。プログラマーがネットワークに問題があるというので、システム管理者は狼狽しています。プログラマーは、ファイルが店舗から中央データレポジトリへ転送される途中で消失しているのに間違いないと言うのです。彼の間違いを証明するのが、今回の目標です。

ケーブルにもぐりこむ

必要なデータを収集するため、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。問題はすべての店舗に影響しているので、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スレッドである、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、トラフィックをスニッフします。キャプチャしたトラフィックは1店舗の単独インスタンスへと分けて、サーバへそのCSVファイルをアップロードします。このキャプチャファイルがtickedoffdeveloper.pcapです。

解析

プログラマー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。[Statistics]から[Flow Graph]を選択、[OK]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通信がひとめで分かる

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で②これがクライアントで、172.16.16.121はデータをコンパイル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接続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内にはFTPリクエストのコマンドが散乱しているので、エクスプレッションビルダで数百ものプロトコルやオプションをソートするのではなく、パケット一覧のペインから直接フィルタを構築します。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。リストの上部に近いので、5番目のパケットを選びましょう。次にパケット詳細のペインで[FTP]を開き、[USER]を開きます。[Request Command: USER]フィールドを右クリックし、[Prepare a Filter]を選択、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すべてのパケットをフィルタにかけ、フィルタダイアログに記録する準備ができました。今度は図8-38のように、USERをSTORに置き換えて、フィルタを編集します①。

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を押してフィルタを有効にすると、64番目のパケットには、STORコマンドのインスタンス1つしか存在していないのがわかります②。

データ転送の始まる場所が分かったので、パケット一覧のペイン上部の[Clear]ボタンをクリックして、フィルタを解除します。

64番目のパケットで始まるキャプチャファイルを調べると、このパケットがファイルstore4829-03222010.csvの転送を指定しているのが分かります（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転送の一部と識別されています。データが転送されたのは確認しましたが、プログラミングの間違いは証明できていません。キャプチャしたパケットから転送されたファイルを抽出して、ネットワーク上を移動したのちもファイルのコンテンツ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ないフォーマットで転送されると、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、宛先に到着したけれど、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ストリームに表示されます。

データはFTP上をプレインテキストの状態で転送されているようですが、ファイルが無事なのはストリームのおかげなのかどうかは確かではありません。データを組み立てるとオリジナルのフォーマットで抽出できるので、[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。そして[Save] をクリックしてください。

図8-40　データの転送状態を示すTCPストリーム

図8-41　ストリームをオリジナルのファイル名で保存する

保存を行うと、格納システムから最初に転送したファイルとまったく同じバイトレベルのCVSファイルができるはずです。オリジナルファイルのMD5ハッシュと抽出したファイルのハッシュを比較すれば検証できます。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。格納システムからコレクションサーバへ転送したファイルは、サーバ到着時には損傷がないので、損傷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になります。

図8-42　オリジナルファイルと抽出ファイルのMD5ハッシュは同じ

学んだこと

パケットレベルの解析がすばらしいのは、ごちゃごちゃしたアプリケーションに対処しなくて済む点です。優れたものよりも、作りが雑なアプリケーションのほうが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。プログラマー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心配しますが、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かTCP、あるいはIPに過ぎないのです。これら基本プロトコルの知識を利用すれば、通信プロセスが正しく処理されているのを確認し、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過ぎないと覚えておきましょう。

まとめ

本章では、パケット解析によって通信の問題がより理解できる基本的なシナリオをいくつか例に挙げました。一般的なプロトコルの基本解析を使えば、ネットワーク問題を追跡し、タイミングよく解決できるのです。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、独自の問題の解析に役立つはずです。