ファジング実践資料（テストデータ編）

「ファジング活用の手引き」別冊
効果的なファジングツールの選定につながるテストデータの理解

独立行政法人情報処理推進機構
技術本部 セキュリティセンター

2016 年 3 月

本書は、以下の URL からダウンロードできます。

「ファジング活用の手引き」 別冊 「ファジング実践資料(テストデータ編)」
https://www.ipa.go.jp/security/vuln/fuzzing.html

目次

目次 1
本書の要旨 2
本書の想定読者 3
本書の構成 3
本書を読む上での注意事項 3
1. 「ファジング」の要点 4
2. 背景 5
2.1. 製品開発における「ファジング」導入の推進 5
2.2. 効果的なファジングツール選定につながる「テストデータ」の理解 5
3. ファジングにおけるテストデータ 6
3.1. ファジングにおける「テストデータ」とは 6
3.2. データ構造の「どの部分」を「どのように」細工するか 8
3.3. テストデータのまとめ 21
4. テストデータの実例 21
4.1. IP パケットのテストデータ 22
4.2. TCP パケットのテストデータ 26
4.3. HTTP リクエストのテストデータ 31
4.4. UPnP リクエストのテストデータ 32
4.5. JPEG 画像のテストデータ 34
4.6. 無線 LAN フレームのテストデータ 38
5. テストデータに関する知識の活用 40
5.1. 一歩進んだ知識の活用：ファジングツールの独自開発 40
6. おわりに 42
7. 本書に関連する仕様／規格 42
付録：検出できるテストデータが分かる脆弱性情報 43

ファジング実践資料（テストデータ編）

～ 効果的なファジングツールの選定につながるテストデータの理解 ～

2013 年 11 月 7 日
IPA（独立行政法人 情報処理推進機構）
セキュリティセンター

本書の要旨

本書は、セキュリティテスト「ファジング」（1 章参照）で問題を検出するための「テストデータ」に焦点を当てた実践資料である。

効果的にファジングを実践するためには、ファジングツールの特徴をふまえて、複数のファジングツールを活用することが有効である。これを実践するためには、ファジングにおける「テストデータ」の理解が欠かせない。

本書では、IPA が 24 種類のファジングツールを使用した経験をもとに、ファジングにおけるテストデータの考え方「データ構造の『どの部分』を『どのように』細工するか」を解説している。また、可能な範囲でテストデータの実例も掲載して、テストデータの理解を深めることができるように工夫した。

本書でファジングにおけるテストデータを知っていただき、特徴の違うファジングツールを選定して、効果的なファジングを実践していただきたい。さらに、本書で紹介してテストデータの考え方が、ファジングツールの独自開発の一助となれば幸いである。

本書の想定読者

- 製品開発企業においてファジングを活用している技術者
- 製品開発企業におけるテスト関係者（テスト計画の立案者や実際のテスト担当者など）

本書の構成

本書は、7 つの章と 1 つの付録で構成されている。

まず 1 章と 2 章で本書を読む上での前提知識（「ファジング」の要点や本書をまとめる経緯）を説明する。

続く 3 章では、「ファジング」におけるテストデータの考え方を説明する。4 章では、3 章で説明したテストデータの考え方を具体的にイメージできるように、テストデータの実例を紹介する。そして 5 章では、本書で解説したテストデータに関する知識をどう活用すればよいか、IPA の考えを述べる。最後に、6 章にて本書を締める。

7 章には、本書に関連する仕様と規格を掲載している。そして付録には、「その脆弱性を検出できるテストデータが分かる」脆弱性情報を掲載している。

本書を読む上での注意事項

IPA が使用したファジングツールの特徴を抽象化して、テストデータの実例を交えながら本書をまとめた。したがって、次の 2 つの制約がある。

- **本書ではテストデータに関する事項を網羅していない**。
- **本書のテストデータを使うことで必ず脆弱性を検出できるわけではない**。

1. 「ファジング」の要点

まず、本書を読むために「ファジング」の要点を説明しよう。

「ファジング（英名：fuzzing）」とは、多数の問題を起こしそうなデータ（例：極端に長い文字列）を対象製品に送り込み、対象製品の動作状態（例：製品が異常終了する）から脆弱性などを発見するテスト手法である。本書では、この「問題を起こしそうなデータ」を「テストデータ[1]」と呼ぶ。

図 1 にファジングの 2 つの特徴を示した。これらのうち、特に「ファジングツールによってテストデータが異なること」を覚えておいてほしい。

ファジングそのものに興味を持たれた方は、「ファジング活用の手引き」などのファジング関連資料[2]を参照してほしい。

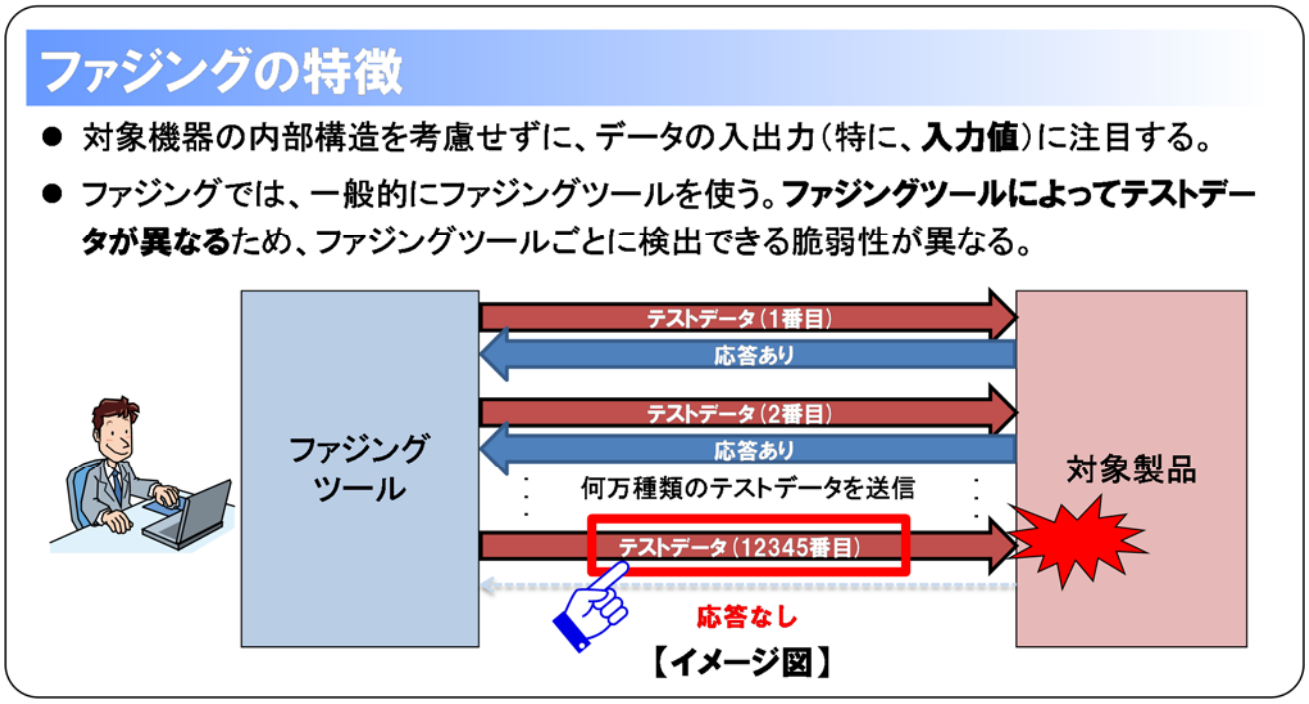


図 1 ファジングの特徴

[1] ファズ（英名：fuzz）と呼ばれることもある。
[2] IPA：ファジング活用の手引き
https://www.ipa.go.jp/security/vuln/fuzzing.html

2. 背景

2.1. 製品開発における「ファジング」導入の推進

IPA では、2011 年 8 月から「ファジング」の有効性実証と普及推進を行う「脆弱性検出の普及活動」[3]を実施している。2013 年 6 月までに、組込み製品 21 製品（ブロードバンドルータやスマートテレビなど）に対するファジングで合計 24 件の脆弱性を検出し、ファジングの有効性を実証してきた。そして、このファジング実績で得られた知見をもとに、製品開発企業にファジングの普及啓発をすすめている。

IPA の活動を通じて、新たにファジングの導入を検討する製品開発企業も出てきており、少しずつではあるが、ファジングを意識する製品開発企業が増えている。

2.2. 効果的なファジングツール選定につながる「テストデータ」の理解

ファジング実績から、IPA では次のようなファジング活用方法を導き出した。

- 可能な限りさまざまな機能に対してファジングを実施する。
- 検出できる脆弱性を補うために、複数のファジングツールを使用する。

この活用方法を実践するためには、ファジングツールごとの「テストデータ」の違いを理解する必要がある。ファジングツールが送るテストデータを理解していないと、製品の機能に合わせてファジングツールを選べず、ファジングツールごとの検出できる脆弱性の違いを判断できない。

しかし、ファジングツールの「テストデータ」は一般的にブラックボックスな状態であるため、ファジングツールを試用して「テストデータ」を調べなければならない。これに加えて、IPA が把握している限りでは、ファジングにおける「テストデータ」を解説している資料は少ない。「ファジング」の専門書籍のなかには「テストデータ」に解説している書籍もあるが、入門書に位置づけられるものではない。

そこで、「テストデータ」を解説することで、効果的なファジングツールの選定につながると考え、これまで IPA が 24 種類のファジングツール[4]を使って検証を重ねてきた経験を基に本書をまとめた。

---

[3] IPA：ソフトウェア製品における脆弱性の減少を目指す「脆弱性検出の普及活動」を開始
https://www.ipa.go.jp/about/press/20110728.html

[4] 商用製品 2 種類、オープンソースソフトウェア 22 種類のファジングツールである。

3. ファジングにおけるテストデータ

本章では、ファジングにおける「テストデータ」を明確にして、そのテストデータにおける考え方「データの『どの部分』を『どのように』細工するか」を解説する。

3.1. ファジングにおける「テストデータ」とは

ファジングにおける「テストデータ」とは、製品に何か問題を起こしそうなデータである。製品が受け取るデータの仕様に則っていない[5]データをテストデータとしてイメージしていただきたい。

3.1.1. 製品が受け取るデータ

製品は、さまざまなデータを受け取る。製品がネットワークで通信するのであれば、通信データを受け取る。そして、この通信データの中身も製品によって異なる。また、画像や動画などを再生する機能を持つ製品であれば、画像ファイルや動画ファイルなどを受け取る。

製品が受け取るデータには仕様などで「データ構造」が決められている。製品はこのデータ構造をもとにデータを解釈する。表 3-1 に 3 種類の製品が受け取るデータとそのデータ構造の関係をまとめた。表 3-1 のウェブサーバを例に、ウェブサーバが受け取るデータを考えてみよう。

ウェブサーバは、ウェブブラウザなどから「HTTP リクエスト」を受け取る。この「HTTP リクエスト」のデータ構造は、アプリケーションプロトコル「HTTP（Hyper Text Transfer Protocol）[6]」で決められている。ウェブサーバは受け取ったデータを HTTP に則って解釈し、そのリクエストにあわせて応答する。

表 3-1 製品が受け取るデータとそのデータ構造

製品	受け取るデータ	データ構造
ネットワークで通信する製品	パケットデータ	IP や TCP などのネットワークプロトコル
画像を取り扱う製品	画像ファイル	JPEG などの画像形式
ウェブサーバ	HTTP リクエスト	HTTP

もし受け取ったデータがデータ構造において異常なものであった場合、製品に何らかの問題が生じる可能性がある。データの一部が壊れた HTTP リクエストを受け取り、ウェブサーバが異常終了してしまう様子をイメージしていただきたい。

ファジングツールはこのように問題を起こしそうなデータを意図的に作り、それで製品に問題が起きないかテストする。

---

[5] RFC(Request For Comment)で規定されているネットワークプロトコルなどの場合には、「RFC に準拠していないデータ」が該当する。

[6] RFC2616：Hypertext Transfer Protocol -- HTTP/1.1
http://tools.ietf.org/html/rfc2616

3.1.2. ファジングツールによるテストデータの作り方

ファジングツールのテストデータの作り方は、製品が受け取るデータのデータ構造を解釈するかどうかで、大きく 2 種類に分かれる。表 3-2 は、その 2 種類の作り方を示している。

表 3-2 ファジングツールのテストデータの作り方

No	データ構造の解釈	テストデータの作り方
1	解釈する	● データ構造の要素をそれぞれ細工してテストデータを作る[7]。
2	解釈しない	● 元となるデータを読み込み、そのデータをランダムに変更する[8]。 ● まったくランダムな値からデータを作る[9]。

表 3-2 の No.1, No.2 どちらの作り方でも、理論的には同じテストデータができる。しかし、No.2 の作り方は無尽蔵に時間が掛かってしまうため、多くのファジングツールは No.1 の作り方を採用している。IPA が使用したファジングツールも No.1 の作り方を採用しているものが多かった。

No.1 の作り方にもデータ構造の要素を細工する方法が色々あるため、No.1 の作り方を採用していてもファジングツールごとにテストデータが異なる。したがって、No.1 の作り方におけるデータ構造の要素を細工する方法を理解すると、ファジングツールの特徴が分かるようになる。

そこで、3.2 節から、IPA が使用したファジングツールのデータを細工する方法を抽象化して、No.1 のデータを細工する考え方をまとめた。

なお、本節以降、特に断りなく「テストデータ」と書いた場合には No.1 の方法で作るテストデータを指し、「ファジングツール」と書いた場合には No.1 の方法でテストデータを作るファジングツールを指す。

7 「Generation based fuzzing」、「Smart fuzzing」、「Intelligent fuzzing」などと呼ばれる。
8 「Mutation based fuzzing」などと呼ばれる。
9 「Dumb fuzzing」などと呼ばれる。

3.2. データ構造の「どの部分」を「どのように」細工するか

ファジングツールは、対象製品が受け取るデータをもとにテストデータを作る。このテストデータを作るときには、データ構造の「どの部分」を「どのように」細工するかが重要となる。

図 2 は、ファジング対象の製品が受け取るデータと、そのデータをもとに作ったテストデータの例を示している。この製品は、「フィールド 1」、「フィールド 2」、「フィールド 3」、「フィールド 4」の 4 つの要素を持つデータを受け取る（図 2 上部）。このデータの「フィールド 1」の値を極端に長い文字列（AAA…）に置き換えると、そのデータはテストデータとなる（図 2 下部）。このように、ファジングツールは対象製品が受け取るデータの要素を細工してテストデータを作る。

ファジングツールによって、データ構造の細工する箇所（「どの部分」）と細工方法（「どのように」細工するか）が異なる。図 2 の例では「フィールド 1」という一つの要素を対象としたが、「フィールド 1」と「フィールド 2」のように複数個の要素を対象とする場合もある。また、図 2 の例では特定の値に置き換えたが、「フィールド 1」の値そのものを削除してしまう場合もある。

このデータ構造の細工する箇所、細工方法を 3.2.1 節、3.2.2 節で詳しく解説していきたい。

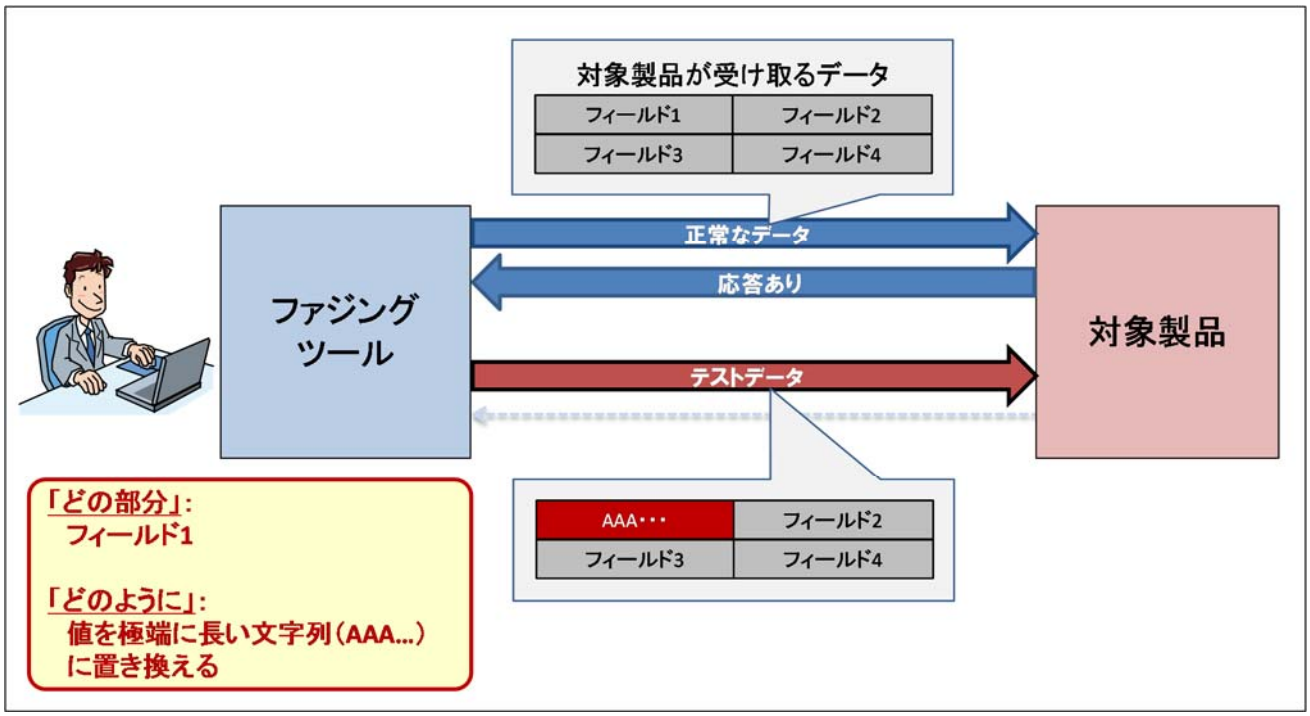


図 2 テストデータの例

3.2.1. 細工する箇所：データ構造の「どの部分」を細工するのか

対象製品が受け取る**データのすべての部分**が、テストデータを作るための細工する箇所になりえる。HTTP リクエストと TCP パケットの 2 つを例に挙げて、データのすべての部分が細工する箇所となりえることを説明する。

まず HTTP リクエストを例に挙げて説明しよう。ウェブサーバに対するファジングを考えると、ファジングツールはウェブサーバが受け取る HTTP リクエストを細工してテストデータを作る。

図 3 は、HTTP リクエストの例を示している。HTTP リクエストのデータ構造は、RFC2616[10]で規定されている。HTTP は HTTP ヘッダと HTTP ボディの 2 つの要素で構成される。そして、HTTP ヘッダには、リクエストラインと複数の HTTP ヘッダ（Host ヘッダや User-Agent ヘッダなど）の要素がある。これらの HTTP のどの要素も細工する対象となりえる。

HTTP リクエストにおいて細工できる箇所を具体的にみてみよう。図 3 の HTTP リクエストを細工してテストデータを作る場合、「HTTP ヘッダの名前」（図 3 青色部分）や「HTTP ヘッダの値」（図 3 赤色部分）、「HTTP ヘッダの名前と値」（図 3 緑色部分）、「複数の HTTP ヘッダ」（図 3 灰色部分）などを細工することが考えられる。

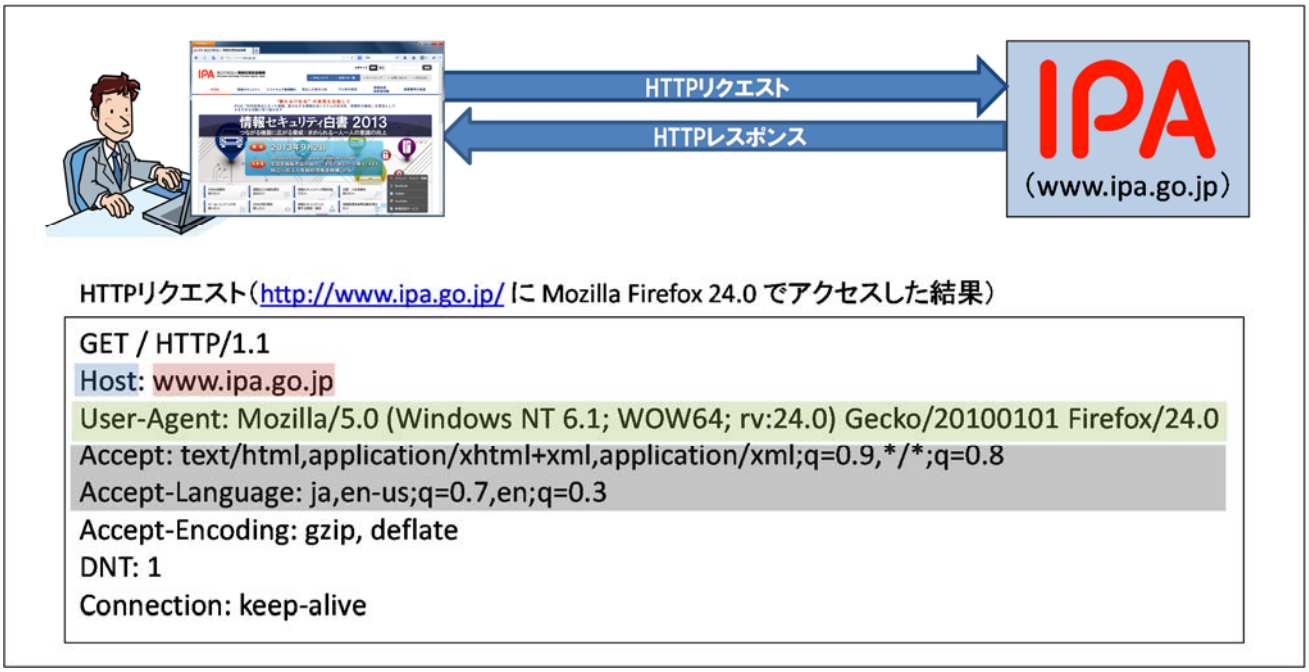


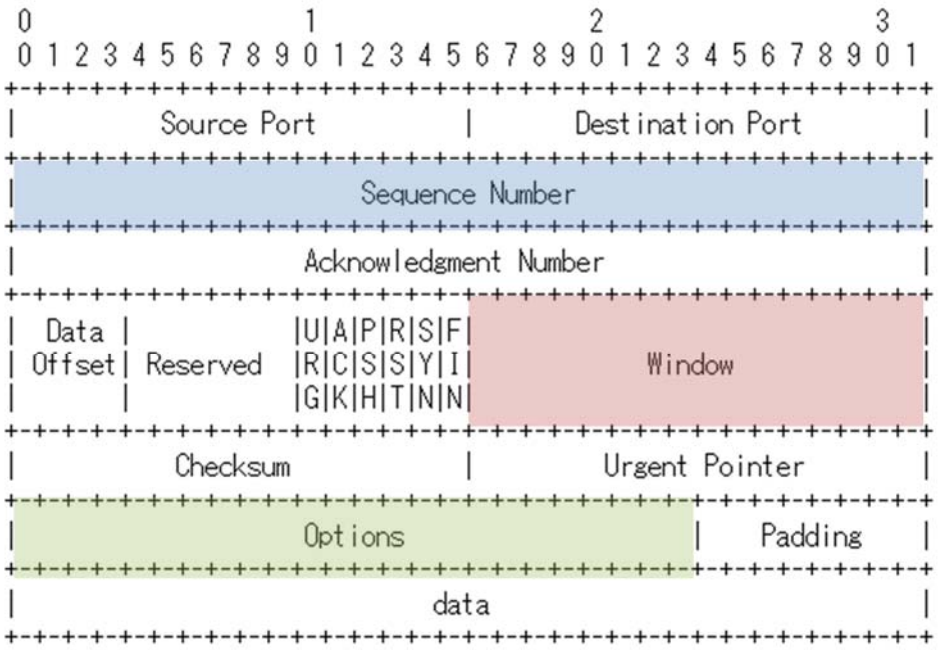
図 3 HTTP リクエストの細工する箇所

[10] RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1
http://tools.ietf.org/html/rfc2616

続いて TCP パケットを例に挙げて説明しよう。TCP[11]で通信するソフトウェアに対するファジングを考えると、ファジングツールはそのソフトウェアが受け取る TCP パケットを細工してテストデータを作る。

図 4 は、RFC793 で規定されている TCP のデータ構造を示している。TCP パケットはこのデータ構造を基に構成される。TCP では 1 ビット単位で要素が規定されているが、HTTP と同様に、TCP のどの要素も細工する対象となりえる。

TCP パケットにおいて細工できる箇所を具体的にみてみよう。TCP パケットを細工してテストデータを作る場合、「『Sequence Number』ヘッダ」(図 4 青色部分)や「『Window』ヘッダ」(図 4 赤色部分)、「『Options』ヘッダ」(図 4 緑色部分)などを細工することが考えられる。また、ヘッダの一部分だけではなく複数の部分をまとめて細工すること(図 4 青色部分と赤色部分など)も考えられる。



http://tools.ietf.org/html/rfc793#section-3.1 から引用。

図 4 TCP パケットの細工する箇所

HTTP リクエストや TCP パケットの例のように、対象製品が受け取るデータのデータ構造に則っていれば、データの「どの部分」も細工する対象となりえる。ただし、検査効率化のため、一部のフィールドを対象外とすることもある。

11 RFC 793 - Transmission Control Protocol
http://tools.ietf.org/html/rfc793

3.2.2. 細工方法：「どのように」細工するのか

ファジングツールは、選んだ箇所を**データ構造と照らし合わせたときに異常なものとなるように**細工する。

IPA が使用してきたファジングツールを分析すると、(a), (b), (c)の 3 種類の細工方法が用いられていた。この 3 種類の細工方法を(a), (b), (c)の順番に説明していきたい。

(a) 「特定の値」に細工する。

(b) 「データ構造」そのものを細工する。

(c) 「データ間のつながり」を細工する。

本節では、細工方法を説明するためにネットワークプロトコルやファイル形式などの仕様／規格を多数引用しているが、**それらを熟知している必要はない**。本節を読む上で必要な事項については文中で解説している。仕様／規格そのものを確認したい場合、7 章を参照していただきたい。

(a) 「特定の値」に細工する

ファジングツールの細工方法には、データ構造における値を「特定の値」に細工する方法がある。図 5 は、その細工方法のイメージ図である。ファジングツールは、データ構造の細工する部分に合わせて「特定の値」を選び、その値で該当部分を置き換えたり、その値を挿入したりする。

「特定の値」には、「特定の脆弱性検出に特化した値」や「特別な意味を持つ値」などがある。表 3-3 に「特定の脆弱性検出に特化した値」の例、表 3-4 に「特別な意味を持つ値」の例を示す。

「特定の脆弱性検出に特化した値」には、バッファオーバーフローの脆弱性を検出する「極端に長い文字列」や、書式文字列の問題を検出する「書式文字列」、または数値処理に関する問題を検出する「意味のある数値」などがある。「特別な意味を持つ値」には、C 言語などの言語処理系で文字列の終端を意味する「ヌルバイト」やデータ構造における要素の区切りを意味する「区切り文字」などがある。

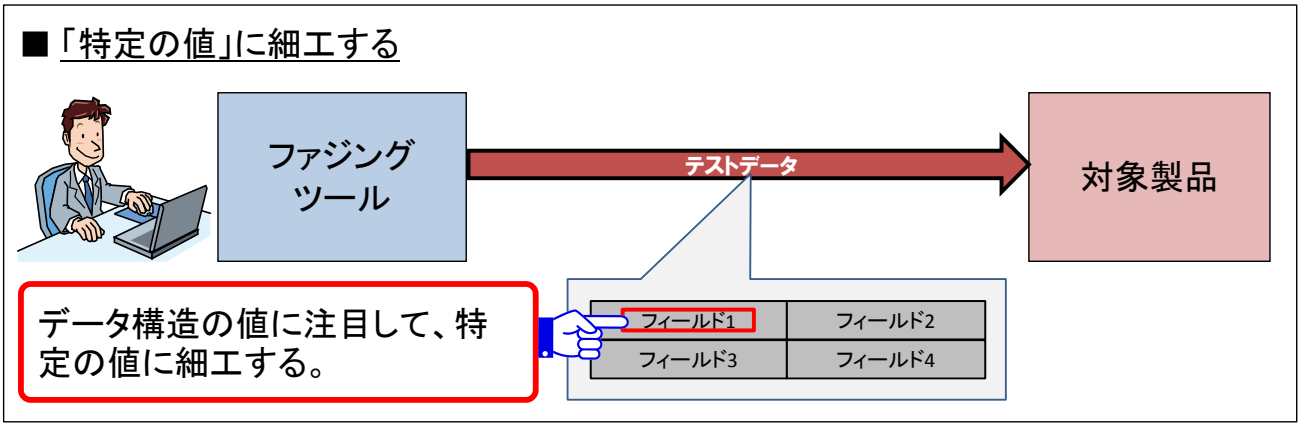


図 5 「特定の値」に細工する（イメージ図）

表 3-3 「特定の脆弱性検出に特化した値」の例

脆弱性名	値
バッファオーバーフローの脆弱性	極端に長い文字列（例：「A」1000 個以上）
書式文字列の問題	C 言語の printf()関数などで使う書式文字列（例：「%s%s%s%s」）
数値処理に関する問題（整数オーバーフローの脆弱性など）	バッファの上限値や下限値として使われそうな数値やプログラミング言語のデータ型のサイズに関連する数値（例：0, 65535 や 65536）

表 3-4 「特別な意味を持つ値」の例

特別な意味を持つ値	値
ヌルバイト（NULL）	「0x00」（16 進数表記） （C 言語などの言語処理系では、文字列の終端を意味する）
区切り文字	データ構造におけるデータの区切りを意味する値 例 1：改行コード（「0x0d」（16 進数表記）や「0x0a」（16 進数表記）） 例 2：「”」や「#」

具体的なイメージを掴むために、HTTP リクエストのテストデータを例示しよう。

図 6 は、HTTP リクエストを「特定の値」に細工する様子を示している。図 6 では、HTTP リクエストにおける URI「/」を細工してテストデータを作る。URI「/」を「極端に長い文字列」に置き換えると、URI を処理する部分にバッファオーバーフローの脆弱性がないかテストするデータになる。

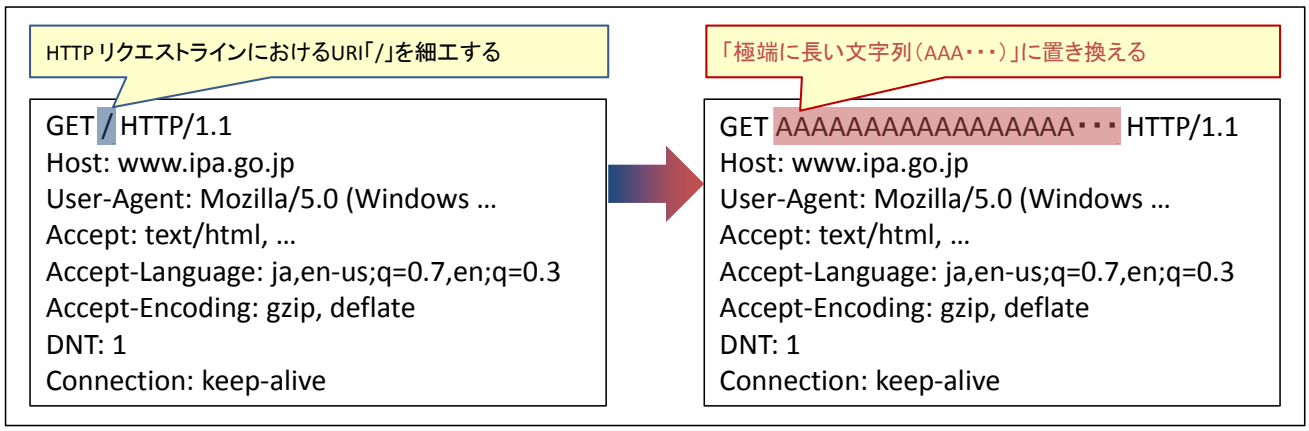


図 6 HTTP リクエストを「特定の値」で細工する様子

テストデータ実例：4.3 節、4.4.1 節、4.5.1 節へ

(b)　「データ構造」そのものを細工する

ファジングツールの細工方法には、「データ構造」そのものを細工する方法がある。図 7 は、その細工方法のイメージ図である。「データ構造」そのものを細工するというとイメージしづらいが、「仕様などで決まっているデータ構造から逸脱するように、意図的にデータ構造を壊す」と理解していただくとよいだろう。

「データ構造」そのものを細工する方法として、次の①から④の 4 種類の方法を取り上げる。この 4 種類の方法を、①、②、③、④の順番に説明していきたい。

① 「仕様で明確に決まっていない値」などに細工する。

② 同じ要素を一定回数繰り返す。

③ データ長と実際のデータの長さを矛盾させる。

④ データ構造全体を仕様と異なるように細工する。

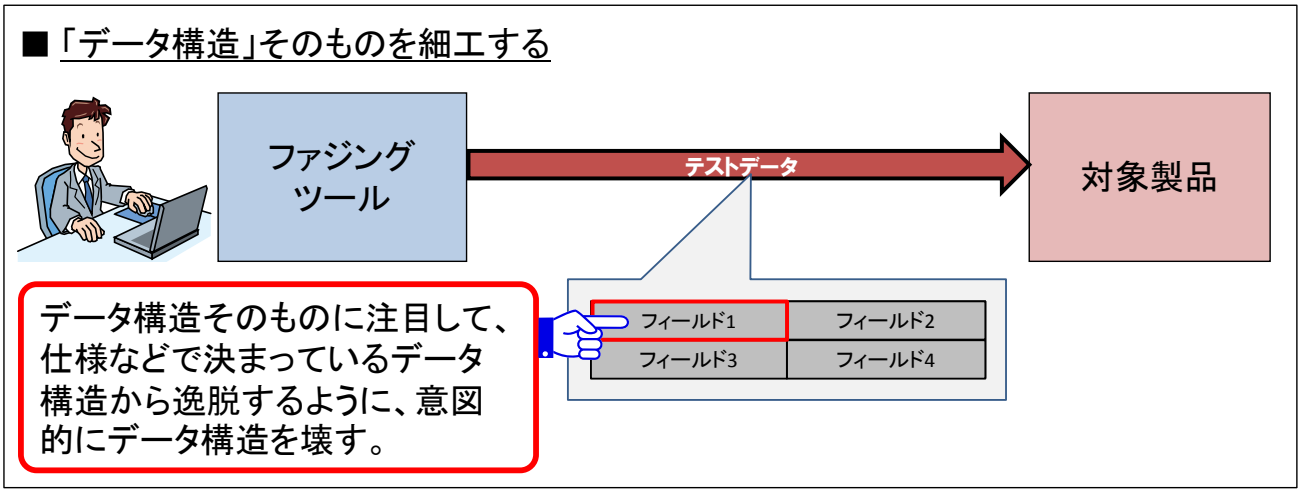


図 7 「データ構造」そのものを細工する（イメージ図）

① 「仕様で明確に決まっていない値」などに細工する。

この細工方法では、細工する箇所を、仕様で明確に決まっていない値などで置き換えたり、挿入したりする。この「仕様で明確に決まっていない値」には、「仕様で規定されていない値」や「今後の拡張などを意識して予約されている値」などがある。

具体的なイメージを掴むために、無線 LAN のテストデータを例示しよう。

図 8 は、無線 LAN でやり取りされる無線 LAN フレームに仕様で明確に決まっていない値を追加する様子を示している。無線 LAN フレームのデータ構造は、「IEEE Std 802.11」[12]で決められている。この規格では「Information Element（以降、IE）」という部分が定義されている（青枠部分）。そして、この「IE」には予約状態となっている要素が多くあり、Element ID「129」[13]はそれらの一つである。

無線 LAN フレームに ElementID「129」の「IE」を追加すると、仕様で明確に決まっていない ElementID「129」があったときに問題が起きないかテストするデータとなる。

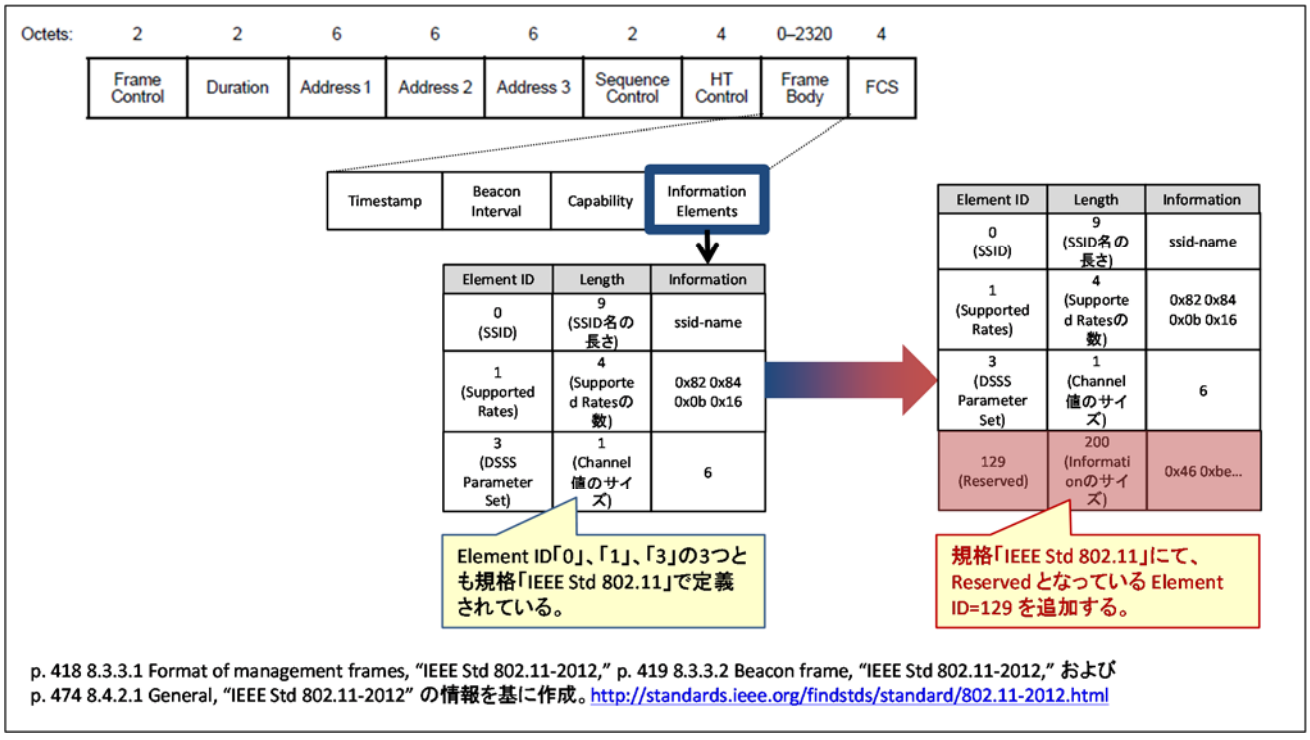


図 8 無線 LAN フレームに仕様で明確に決まっていない値を追加する様子

テストデータ実例：4.6 節へ

[12] IEEE Std 802.11-2012
http://standards.ieee.org/findstds/standard/802.11-2012.html
[13] p.474 8.4.2.1 General, "IEEE Std 802.11-2012"

② 同じ要素を一定回数繰り返す。

この細工方法では、データ構造の細工する部分を一定回数繰り返す。繰り返す回数はファジングツールに依存する。

具体的なイメージを掴むために、HTTP リクエストのテストデータを例示しよう。

図 9 は、HTTP リクエストの要素を一定回数繰り返す様子を示している。図 9 では HTTP リクエストの「Host ヘッダのヘッダ名および値」を細工してテストデータを作る。「Host ヘッダのヘッダ名および値」を 5 回繰り返すと、同じ Host ヘッダが複数あるときに問題が起きないかテストするデータになる。

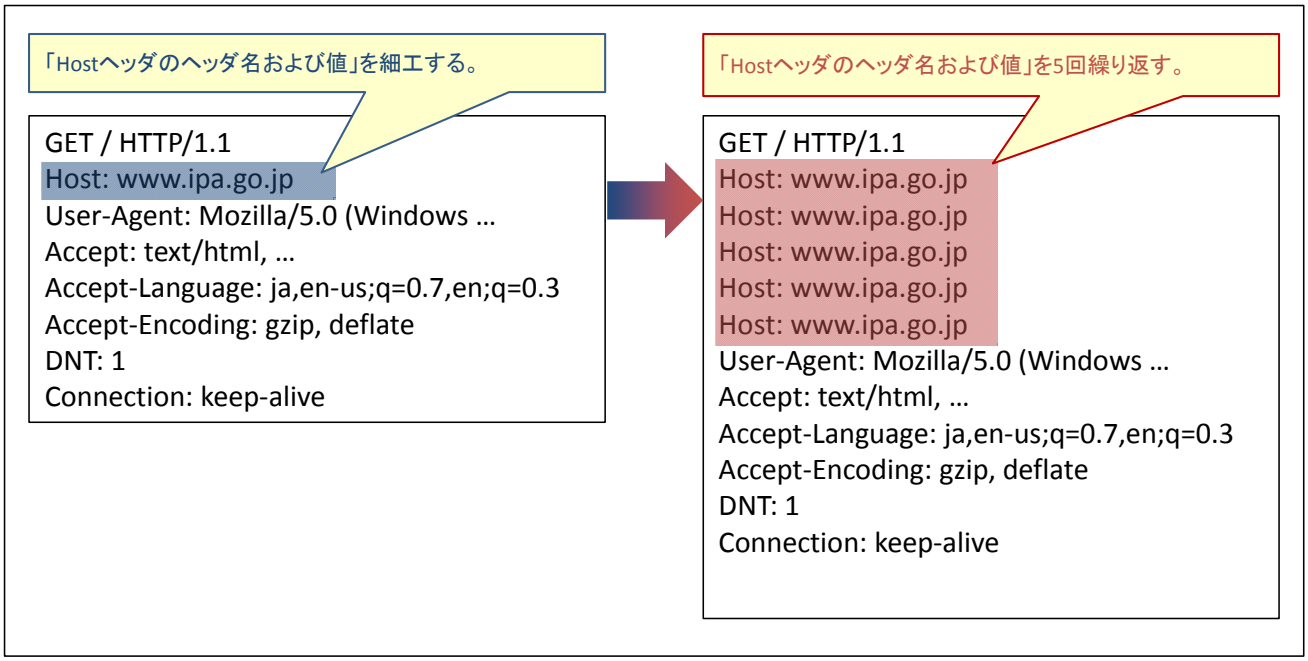


図 9 HTTP リクエストの要素を一定回数繰り返す様子

テストデータ実例：4.4.2 節へ

③ データ長と実際のデータの長さを矛盾させる。

この細工方法では、データ構造における「データ長」を、実際のデータの長さと矛盾する値で置き換える。

データ構造のなかには、「データのタイプ」、「データ長」、そして「データの値」の 3 つの要素で構成されるデータ構造（以降、TLV 構造[14]）がある。TLV 構造における「データ長」は、一般的にこれら 3 つの要素全体の長さ、または「データの値」の長さを示す。この「データ長」をそれが示す長さよりも大きな値や小さな値に置き換えることで、TLV 構造に矛盾を生じさせてしまう。

具体的なイメージを掴むために、TCP パケットのテストデータを例示しよう。

図 10 は、TCP Options ヘッダ「Window Scale Option」のデータ長を細工する様子を示している。「Window Scale Option」は TLV 構造であり、そのデータ長「Length」は RFC1323[15]で「3」と定義されている（図 10 の青色部分）。この「Length」を「3」以外の値にすると、「Window Scale Option」を取り扱うときに問題が起きないかテストするデータとなる（図 10 では「Length」を「0」とした）。

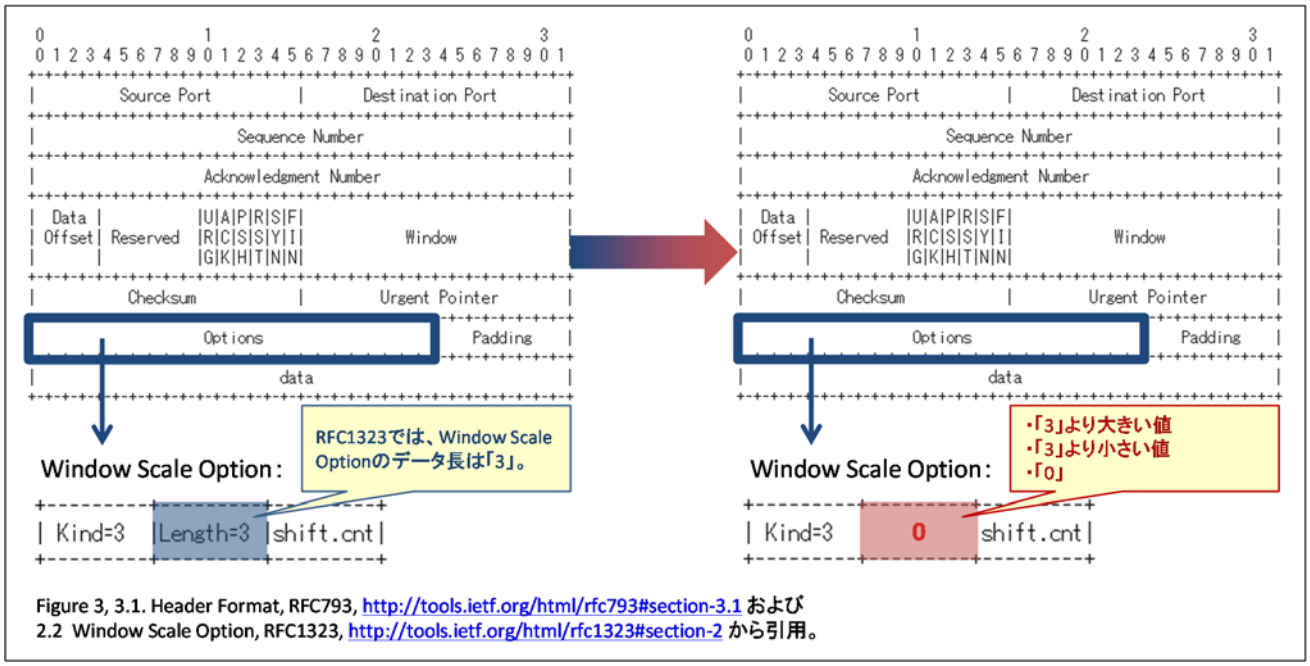


図 10 TCP Options ヘッダ「Window Scale Option」のデータ長を細工する様子

テストデータ実例：4.6 節へ

[14] Tag-Length-Value（TLV）、または Type-Length-Value（TLV）構造と呼ばれることがある。
[15] 2.2　Window Scale Option, “RFC 1323 - TCP Extensions for High Performance, “ http://tools.ietf.org/html/rfc1323#section-2

④ データ構造全体を仕様と異なるように細工する。

この細工方法では、データ構造全体を仕様と異なるように細工する。仕様と異なるように細工する方法には、「データ構造の要素の順序を仕様と異なるものに変えること」や「仕様で定義されていない要素を追加すること」などがある。

具体的なイメージを掴むために、JPEG 画像ファイルのテストデータを例示しよう。

図 11 は、Exif[16]形式の JPEG 画像ファイルを仕様と異なるように細工する様子を示している。Exif 形式のデータ構造は「デジタルスチルカメラ用画像ファイルフォーマット規格 Exif 2.3」[17]という規格で決められている。

Exif 形式の JPEG 画像ファイルは、複数のセグメント（SOI セグメントや APP1 セグメントなど）と呼ばれる要素で構成される。Exif 形式の規格によると、SOI セグメントの後には APP1 セグメントが続くことになっている。この SOI セグメントの後に余分な SOF セグメントを追加すると、セグメントの順序が Exif 形式の規格と異なっているときに問題が起きないかテストするデータとなる。

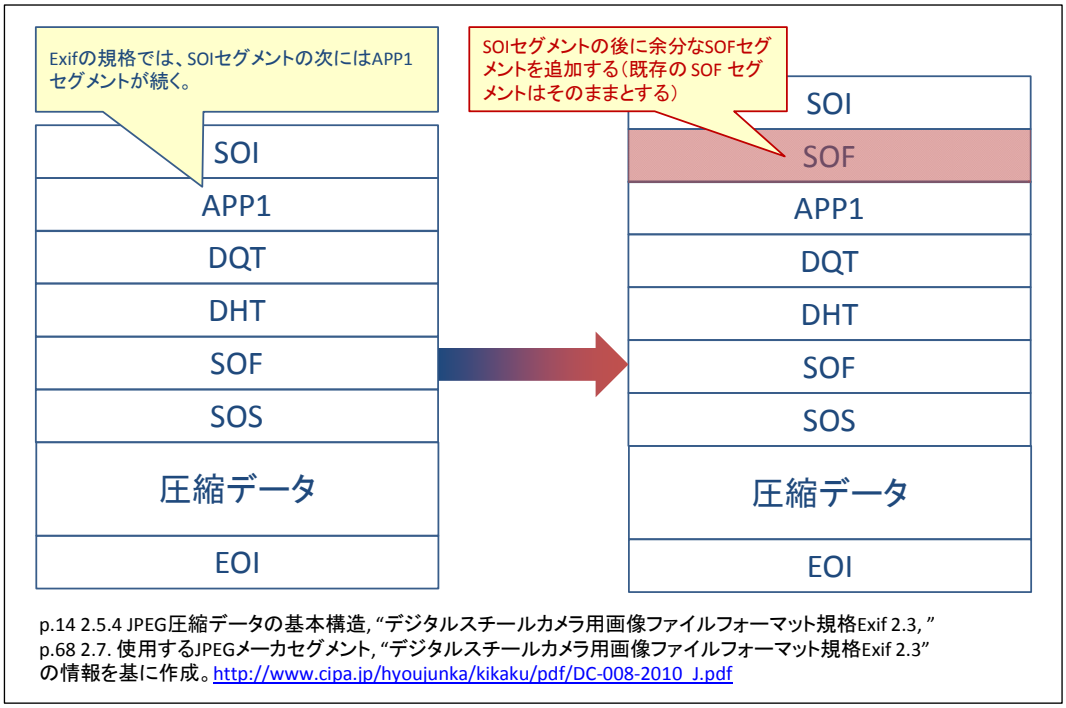


図 11 Exif 形式の JPEG 画像ファイルにおけるテストデータの例

テストデータ実例：4.5.2 節へ

16 Exchangeable image file format for digital still cameras。JPEG 画像のデータ形式の一つである。
17 デジタルスチルカメラ用 画像ファイルフォーマット規格 Exif 2.3
http://www.cipa.jp/std/documents/j/DC-008-2012_J.pdf

(c)　「データ間のつながり」を細工する

ファジングツールの細工方法には、複数のデータの「データ間のつながり」を細工する方法がある。図 12 は、その細工方法のイメージ図である。

データ構造によっては、それぞれのデータのつながりを意味する要素がある。例えば、ネットワークプロトコル「IP（Internet Protocol）」であれば IP ヘッダ「Fragment Offset」[18]、「TCP」であれば TCP ヘッダ「Sequence Number」[19]が挙げられる。これらの要素を細工することで、データ間のつながりに問題を生じさせる。

具体的なイメージを掴むために、IP ヘッダ「Fragment Offset」を細工するときの考え方、TCP ヘッダ「Sequence Number」を細工するときの考え方を例示しよう。

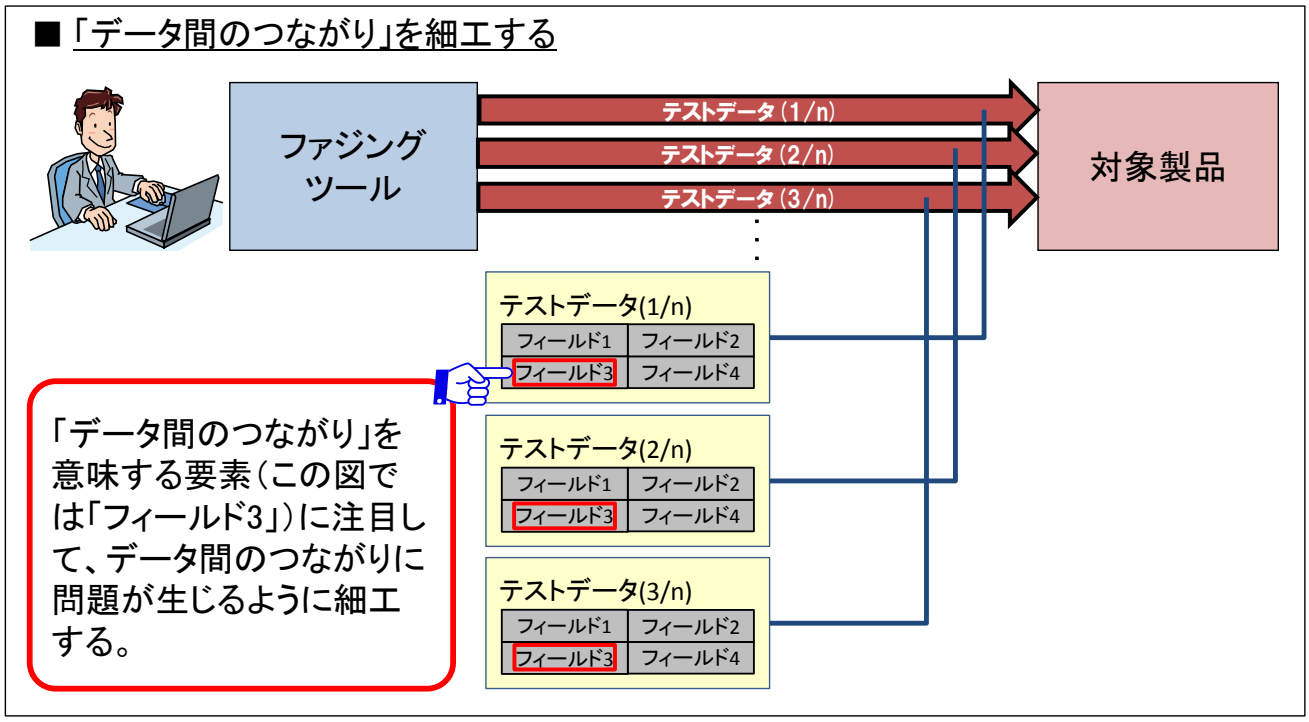


図 12 「データ間のつながり」を細工する（イメージ図）

[18] Fragment Offset, 3.1. Internet Header Format, RFC791, http://tools.ietf.org/html/rfc791#section-3.1

[19] 3.3. Sequence Numbers, RFC793, http://tools.ietf.org/html/rfc793#section-3.3

■ IP ヘッダ「Fragment Offset」を細工するときの考え方

「Fragment Offset」は、複数の IP パケットにデータを分割したときに、それぞれの IP パケットがデータのどの部分を送っているかを示す値である。

ネットワークの最大伝送単位（MTU:Maximum Transmission Unit）を超えるデータを IP パケットで送信するとき、複数の IP パケットにデータを分割して送る。このとき、それぞれの IP パケットの「Fragment Offset」には「その IP パケットのデータが分割する前のどの部分であるか」を示す値が設定される。データを分割した IP パケット群を受け取った製品は、「Fragment Offset」をもとに元のデータを組み立てる。

具体的な例を考えてみよう。図 13 は、最大伝送単位が 1500 バイトのネットワークにおいて、3680 バイトのデータを 3 つの IP パケットに分割して送る様子を示している。3680 バイトのデータは、1480 バイトのデータを持つ IP パケット①、②、残る 720 バイトのデータを持つ IP パケット③に分割して送信される。このとき、IP パケット①、②、③の「Fragment Offset」はそれぞれ「0」、「1480」、「2960」となる。

図 13 の IP パケットにおける「Fragment Offset」を細工するときには、次のような細工方法がある。実際に、過去に IP ヘッダ「Fragment Offset」に起因した脆弱性が発見されて修正されている[20]。

- ①の「Fragment Offset」を「0」以外の値とする。
- ②の「Fragment Offset」を「1480」以外の値（「1480」より大きい値、小さい値）とする。
- ③の「Fragment Offset」を「2960」以外の値（「2960」より大きい値、小さい値）とする。
- 前述 3 つの細工方法を組み合わせる

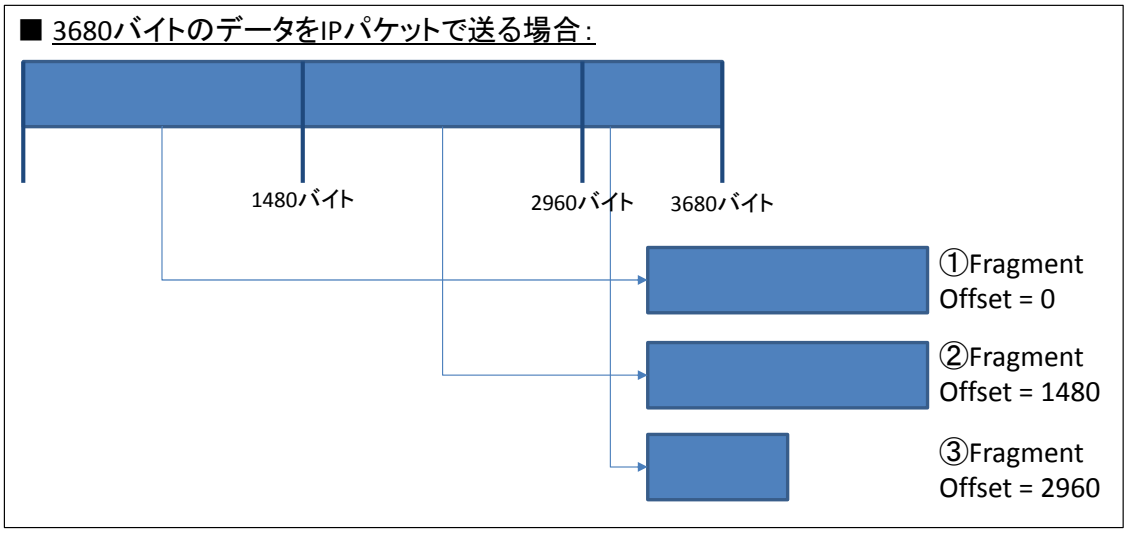


図 13 3680 バイトのデータを 3 つの IP パケットに分割して送る様子

テストデータ実例：4.1 節へ

[20] IPA：「TCP/IP に係る既知の脆弱性に関する調査報告書 改定第 5 版」
pp.166-171 22). フラグメントパケットの再構築時にシステムがクラッシュする問題(Teardrop Attack)
http://www.ipa.go.jp/files/000024459.pdf

■ TCP ヘッダ「Sequence Number」を細工するときの考え方

「Sequence Number」は、TCP パケットがデータのどの部分を送っているかを示す値である。TCP では、この「Sequence Number」と、データを受け取る側がデータのどの部分まで受け取ったかを示す「Acknowledge Number」を組み合わせて、伝送経路においてデータの損失を防ぐ信頼性のある通信を実現する。

具体的な例を考えてみよう。図 14 は、最大伝送単位が 1500 バイトのネットワークにおいて、3680 バイトのデータを 3 つの TCP パケットで順番に送る様子を示している。3680 バイトのデータは、1460 バイトのデータ①、②、残る 760 バイトのデータ③に分けて順番に送られる。

まず①のデータを TCP パケットで送るときには、「Sequence Number」に「1」を設定する。すると、①のデータを受け取った対象製品から、次に期待するデータの位置「1461」を「Acknowledge Number」に設定した TCP パケットが応答される。続く②のデータでは、「Sequence Number」に「1461」を設定した TCP パケットを送り、「Acknowledge Number」に「2921」が設定された TCP パケットが応答される。最後の③のデータでは「Sequence Number」が「2921」、「Acknowledge Number」が「3681」となる。

図 14 の TCP パケットにおける「Sequence Number」を細工するときには、次のような細工方法がある。

- ①を送る TCP パケットの「Sequence Number」を「1」以外の値とする。
- ②を送る TCP パケットの「Sequence Number」を「1461」以外の値とする。
- ③を送る TCP パケットの「Sequence Number」を「2921」以外の値とする。
- 前述 3 つの細工方法を組み合わせる。

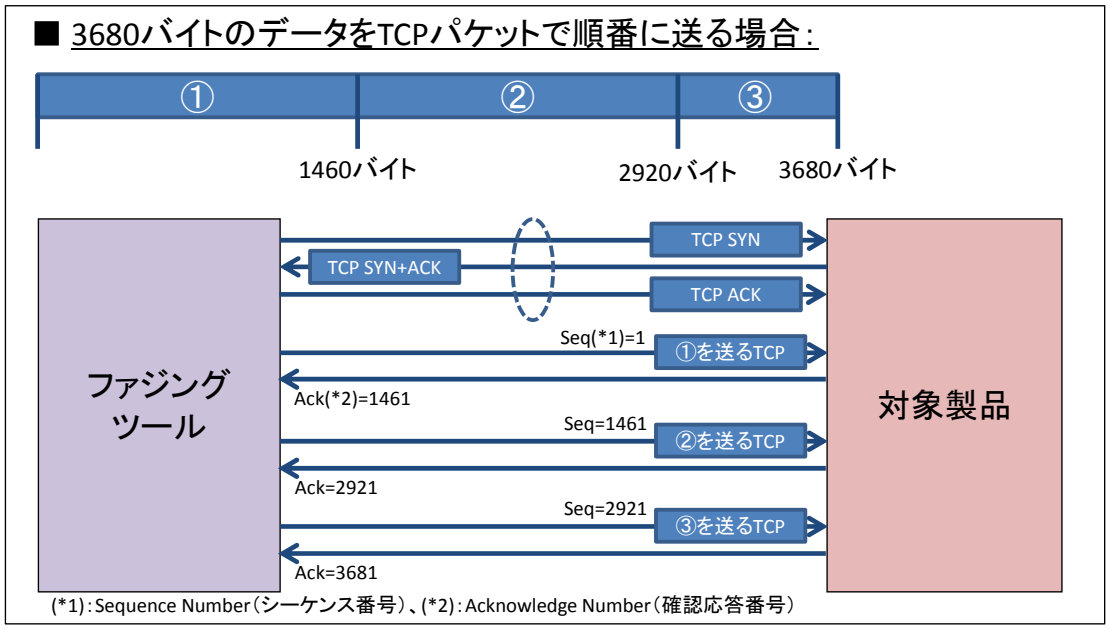


図 14 3680 バイトのデータを 3 つの TCP パケットで順番に送る様子

テストデータ実例：4.2 節へ

3.3. テストデータのまとめ

本章では、テストデータにおける考え方「データの『どの部分』を『どのように』細工するか」を「どの部分」と「どのように」に分けて説明してきた。ファジングにおけるテストデータを考える上では、次の 2 点を忘れないでほしい。

- ファジングでは、**データの構造におけるすべての部分**を細工の対象として考える。もし検査時間がかかりすぎるのであれば、一部分を細工の対象から外す。
- ファジングでは、**データ構造と照らし合わせたときに異常と判定されるように**データを細工する。

4. テストデータの実例

本章では、IPA の「脆弱性検出の普及活動」において、実際に脆弱性を検出した 8 種類のテストデータを紹介する。

3 章ではテストデータの考え方に終始しているため、具体的なテストデータをイメージしづらい方もいらっしゃっただろう。そこで、本章では 3 章の考え方に即したテストデータを取り上げる。3 章の内容を念頭におきながら読んでほしい。

なお、本書をまとめるうえで、インターネットで公開されている脆弱性情報の中から「その脆弱性を検出できるテストデータが分かるもの」を調査した。テストデータの実例をより多く知りたい方にとって参考になると考え、その脆弱性情報を付録に掲載した。

4.1. IP パケットのテストデータ

この節では、IP ヘッダ「Fragment Offset」に本来の値とは異なる値が設定された IP パケットを紹介する。表 4-1 は、このテストデータの細工箇所と細工方法を示している。

表 4-1 IP パケットのテストデータ

データの「どの部分」	IP ヘッダ「Fragment Offset」
「どのように」細工したか	「データ間のつながり」を細工した。 関連： 3.2.2(c)

このテストデータは、IP アドレス 192.168.11.20 から 192.168.11.1 に送信された IP パケットである。図 15 は、そのテストデータの一部をパケット解析ツール「Wireshark」[21]で表示した様子を示している。

図 15 の赤枠部分に注目してほしい。この赤枠部分は、2 つ目の IP パケットの「Fragment Offset」を示している。1 つ目の IP パケットのデータは 8 バイトであるため（図 15 青枠部分）、赤枠部分の「Fragment Offset」は、正しくは「8」となる。しかし、実際には「16」となっている。

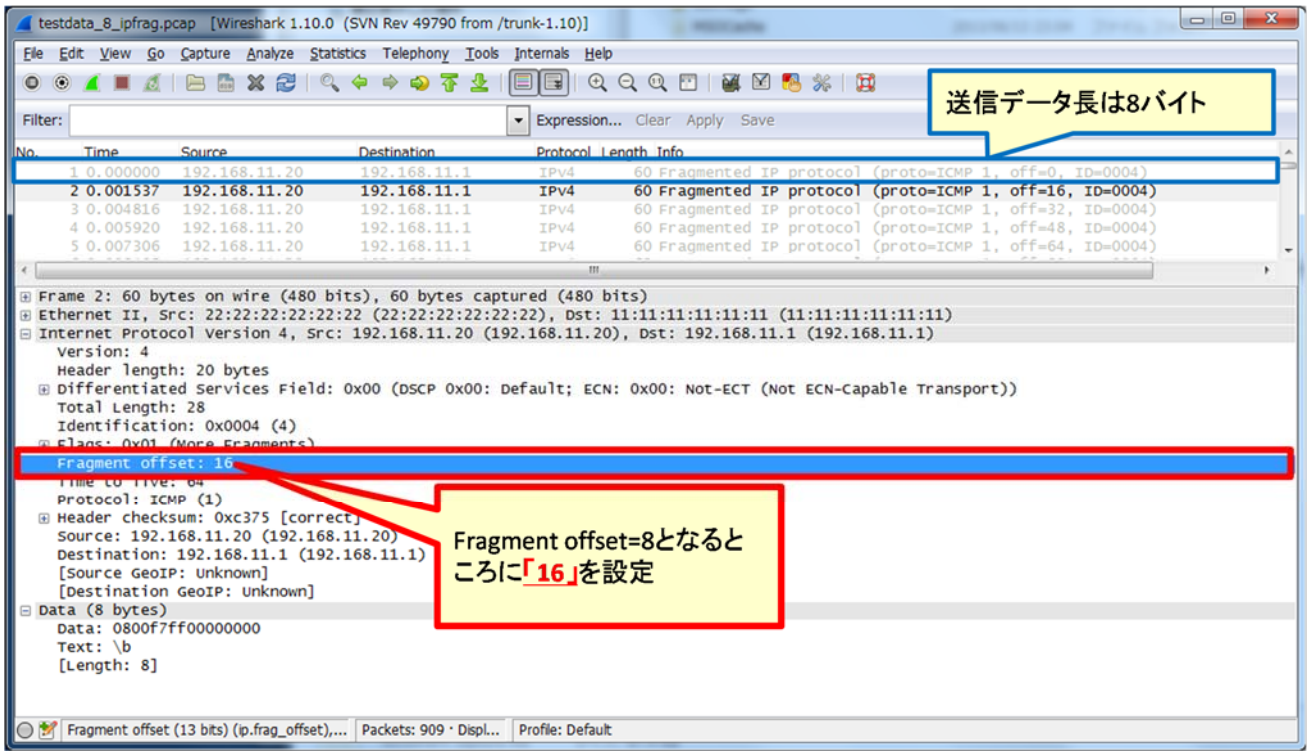


図 15 「Fragment Offset」を細工した IP パケット

[21] https://www.wireshark.org/

さらに、このテストデータを詳しくみてみよう。図 16 は図 15 のパケットを時系列で並べた様子[22]を示している。図 16 の[Comment]列に表示されている「off」部分が「Fragment offset」に該当する（図 16 の赤枠部分）。

IP パケットの「Fragment offset」には、一つ前の IP パケットで送信したデータのデータ長を加算した値が設定される。このテストデータの場合、各 IP パケットのデータがすべて 8 バイトであるため、「Fragment offset」が 0, 8, 16, …と 8 ずつ加算されるはずである（図 16 青枠部分）。しかし、実際には、0, 16, 32, … と一つ前のパケットで送信したデータを無視して、16 ずつ加算されていた。

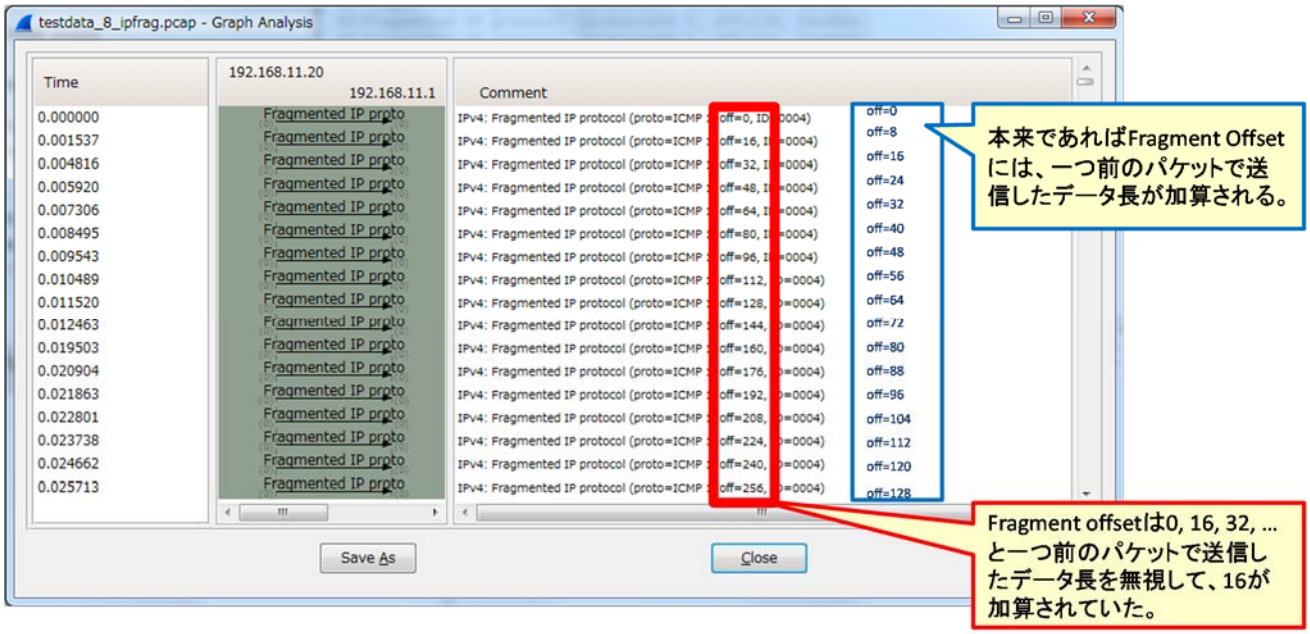


図 16 IP パケットのテストデータを時系列に並べた様子

参考までに、このテストデータの最初の 3 つの IP パケットをテキスト形式で掲載する。なお、製品の特定につながる恐れがあるため、IP パケットに含まれる送信先 MAC アドレスと送信元 MAC アドレスをそれぞれ「11:11:11:11:11:11」、「22:22:22:22:22:22」に置換していることに注意していただきたい。

```
Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
```

22 テストデータを記録したパケットキャプチャファイルを「Wireshark」で開き、[Statistics]メニューの[Flow Graph…]を実行した結果である。この結果のスクリーンショットを取得して、それに筆者が青字で加筆した。

```
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
Transport) (0x00)
    Total Length: 28
    Identification: 0x0004 (4)
    Flags: 0x01 (More Fragments)
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..1. .... = More fragments: Set
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xc377 [correct]
    Source: 192.168.11.20 (192.168.11.20)
    Destination: 192.168.11.1 (192.168.11.1)
Data (8 bytes)

0000  08 00 f7 ff 00 00 00 00                          ........
    Data: 0800f7ff00000000
    [Length: 8]

Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
Transport) (0x00)
    Total Length: 28
    Identification: 0x0004 (4)
    Flags: 0x01 (More Fragments)
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..1. .... = More fragments: Set
    Fragment offset: 16
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xc375 [correct]
```

```
    Source: 192.168.11.20 (192.168.11.20)
    Destination: 192.168.11.1 (192.168.11.1)
Data (8 bytes)

0000  08 00 f7 ff 00 00 00 00                          ........
    Data: 0800f7ff00000000
    [Length: 8]

Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
ECN-Capable Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
Transport) (0x00)
    Total Length: 28
    Identification: 0x0004 (4)
    Flags: 0x01 (More Fragments)
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..1. .... = More fragments: Set
    Fragment offset: 32
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xc373 [correct]
    Source: 192.168.11.20 (192.168.11.20)
    Destination: 192.168.11.1 (192.168.11.1)
Data (8 bytes)

0000  08 00 f7 ff 00 00 00 00                          ........
    Data: 0800f7ff00000000
    [Length: 8]
```

4.2. TCP パケットのテストデータ

この節では、TCP ヘッダの「Sequence Number」に本来の値と異なる値が設定された TCP パケットを紹介する。表 4-2 は、このテストデータの細工箇所と細工方法を示している。

表 4-2 TCP パケットのテストデータ

データの「どの部分」	TCP ヘッダ「Sequence Number」
「どのように」細工したか	「データ間のつながり」を細工した。 関連： 3.2.2(c)

このテストデータは、IP アドレス 192.168.11.20 と 192.168.11.1 の間で送受信された TCP パケットである。図 17 は、このテストデータの一部を「Wireshark」で表示した様子を示している。

図 17 の赤枠部分に注目してほしい。この赤枠部分は、TCP 通信を確立したあと（図 17 の青枠部分）の最初の TCP パケットの「Sequence Number」を示している。一つ前の TCP パケットは、「Sequence Number」に「1」が設定され、0 バイトのデータをもつ TCP パケットである。このため、赤枠部分の「Sequence Number」は、正しくは「1」となる。しかし、テストデータの「Sequence Number」は「9445」となっている。

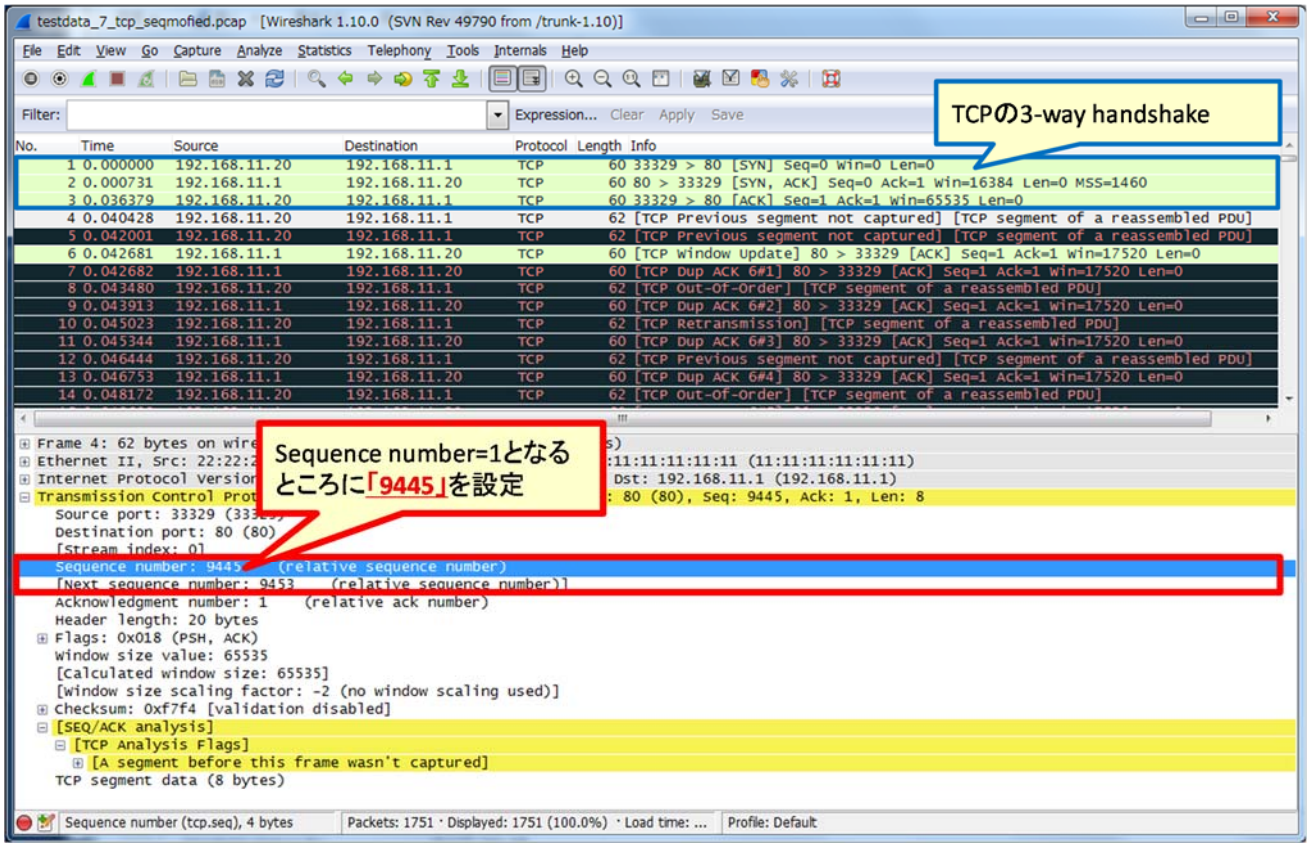


図 17 「Sequence Number」を細工した TCP パケット

さらにこのテストデータを詳しくみてみよう。図 18 は、図 17 のパケットを時系列で並べた様子[23]を示している。図 18 の [Comment]列に表示されている「Seq」部分が「Sequence Number」に該当する（図 18 の赤枠部分）。

TCP パケットの「Sequence Number」には、一つ前の TCP パケットで送信したデータのデータ長を加算した値が設定される。このテストデータの場合、各 TCP パケットのデータがすべて 8 バイトであるため、「Sequence Number」が 1, 9, 17, 25, …と 8 ずつ加算されるはずである（図 18 青枠部分）。しかし、実際には、9445, 12209, 1693, … と一つ前のパケットで送信したデータのデータ長を無視していた。

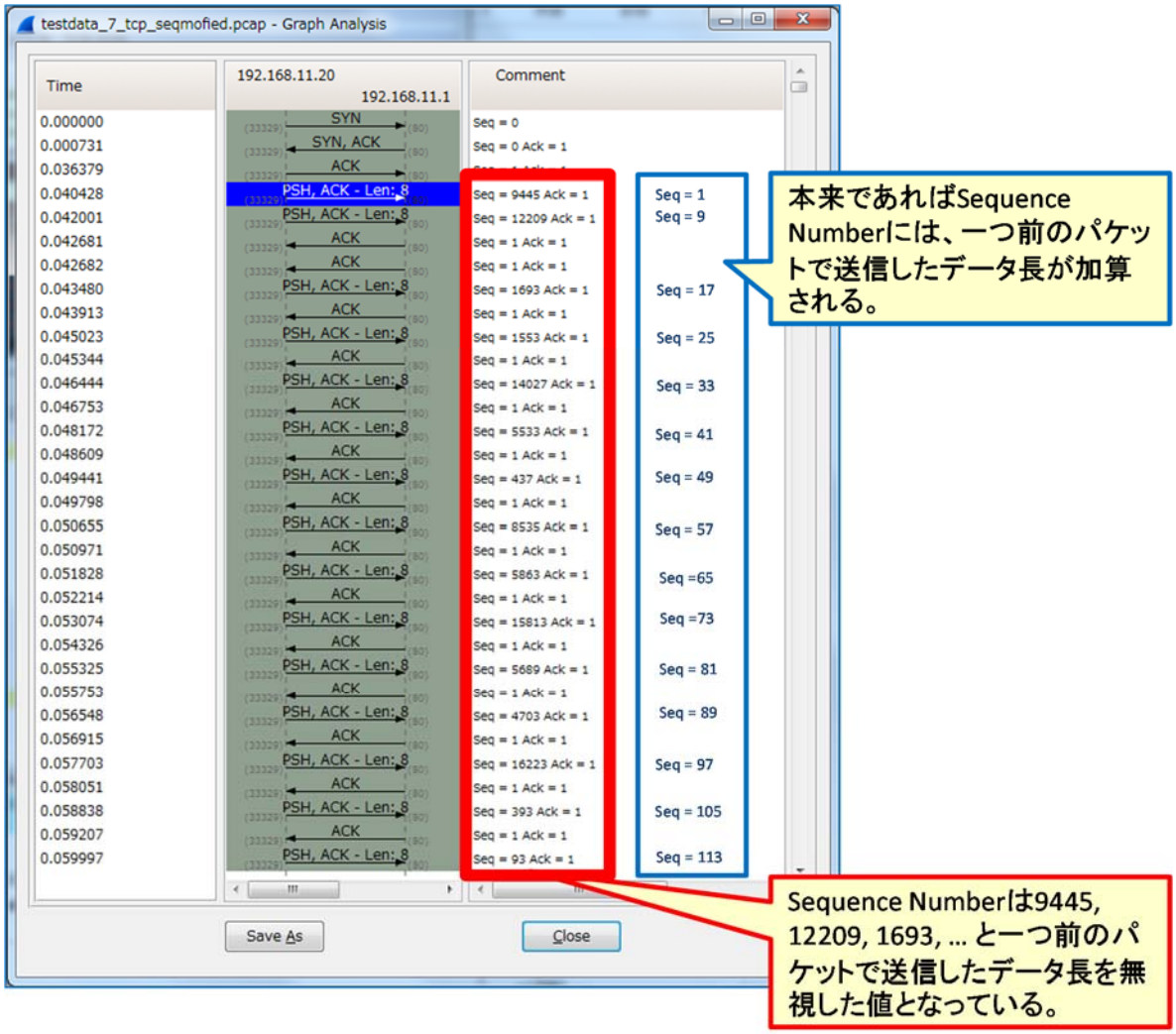


図 18 TCP パケットのテストデータを時系列に並べた図

[23] テストデータを記録したパケットキャプチャファイルを「Wireshark」で開き、[Statistics]メニューの[Flow Graph…]を実行した結果である。この結果のスクリーンショットを取得して、筆者が青字で加筆した。

参考までに、このテストデータのうち、TCP 通信を確立したあとに 192.168.11.20 から 192.168.11.1 に送信した 3 つの TCP パケットをテキスト形式で掲載する。なお、製品の特定につながる恐れがあるため、IP パケットに含まれる送信先 MAC アドレスと送信元 MAC アドレスをそれぞれ「11:11:11:11:11:11」、「22:22:22:22:22:22」に置換していることに注意していただきたい。

```
Frame 4: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
Transmission Control Protocol, Src Port: 33329 (33329), Dst Port: 80 (80), Seq: 9445,
Ack: 1, Len: 8
    Source port: 33329 (33329)
    Destination port: 80 (80)
    [Stream index: 0]
    Sequence number: 9445    (relative sequence number)
    [Next sequence number: 9453    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 20 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
    Window size value: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xf7f4 [validation disabled]
    [SEQ/ACK analysis]
        [TCP Analysis Flags]
            [A segment before this frame wasn't captured]
                [Expert Info (Warn/Sequence): Previous segment not captured (common
at capture start)]
                    [Message: Previous segment not captured (common at capture
start)]
                    [Severity level: Warn]
                    [Group: Sequence]
```

```
    TCP segment data (8 bytes)

Frame 5: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
Transmission Control Protocol, Src Port: 33329 (33329), Dst Port: 80 (80), Seq:
12209, Ack: 1, Len: 8
    Source port: 33329 (33329)
    Destination port: 80 (80)
    [Stream index: 0]
    Sequence number: 12209    (relative sequence number)
    [Next sequence number: 12217    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 20 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
    Window size value: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xed28 [validation disabled]
    [SEQ/ACK analysis]
        [TCP Analysis Flags]
            [A segment before this frame wasn't captured]
                [Expert Info (Warn/Sequence): Previous segment not captured (common
at capture start)]
                    [Message: Previous segment not captured (common at capture
start)]
                    [Severity level: Warn]
                    [Group: Sequence]
    TCP segment data (8 bytes)

Frame 8: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
```

```
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 11:11:11:11:11:11
(11:11:11:11:11:11)
Internet Protocol Version 4, Src: 192.168.11.20 (192.168.11.20), Dst: 192.168.11.1
(192.168.11.1)
Transmission Control Protocol, Src Port: 33329 (33329), Dst Port: 80 (80), Seq: 1693,
Ack: 1, Len: 8
    Source port: 33329 (33329)
    Destination port: 80 (80)
    [Stream index: 0]
    Sequence number: 1693    (relative sequence number)
    [Next sequence number: 1701    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 20 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
    Window size value: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x163d [validation disabled]
    [SEQ/ACK analysis]
        [Bytes in flight: 10524]
        [TCP Analysis Flags]
            [This frame is a (suspected) out-of-order segment]
                [Expert Info (Warn/Sequence): Out-Of-Order segment]
                    [Message: Out-Of-Order segment]
                    [Severity level: Warn]
                    [Group: Sequence]
    TCP segment data (8 bytes)
```

4.3. HTTP リクエストのテストデータ

この節では、Host ヘッダを特定の値で置き換えた HTTP リクエストを紹介する。表 4-3 は、このテストデータの細工箇所と細工方法を示している。

表 4-3 HTTP リクエストのテストデータ

データの「どの部分」	HTTP における「Host ヘッダ」の値
「どのように」細工したか	「特定の値」(特定の脆弱性検出に特化した値)に細工した。 関連: 3.2.2(a)

このテストデータは、ウェブサーバ 192.168.11.1 に対して送信された HTTP リクエストである。このテストデータのもととなった HTTP リクエストでは、Host ヘッダの値に 192.168.11.1 が設定されていた。この Host ヘッダの値を「1024 個の『A』」(赤字部分)に置き換えた。

```
GET / HTTP/1.1
Host: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

4.4. UPnP リクエストのテストデータ

この節では、「Body」部分を細工した UPnP リクエスト（1）、（2）を紹介する。表 4-4 はテストデータ（1）の細工箇所と細工方法、表 4-5 はテストデータ（2）の細工箇所と細工方法を示している。なお、UPnP リクエストからファジングを実施した製品を特定できる恐れがあるため、UPnP リクエストを整形したうえで掲載した。

表 4-4 UPnP リクエストのテストデータ （1）

データの「どの部分」	UPnP リクエストにおける「Body」部分
「どのように」細工したか	「特定の値」（特定の脆弱性検出に特化した値）に細工した。 関連： 3.2.2(a)

表 4-5 UPnP リクエストのテストデータ （2）

データの「どの部分」	UPnP リクエストにおける「Body」部分
「どのように」細工したか	「データ構造」そのものを細工した。 関連： 3.2.2(b) ②

4.4.1. UPnP リクエストのテストデータ （1）

このテストデータは、UPnP に対応した製品 192.168.11.1 に対して送信された UPnP リクエストである。このテストデータのもととなった UpnP リクエストでは、「Body」部分の XML 要素「Element2」の値に文字「a」が設定されていた。この「a」を文字列「00 00 00 00」（赤色部分）に置き換えた。

この「00 00 00 00」が文字列、10 進数の数値、または 16 進数の数値の形式で解釈されるかは確認していないが、少なくとも、もとの UpnP リクエストに比べてデータサイズが大きくなっているはずである。だが、このテストデータを作ったファジングツールは、「Body」部分のデータサイズを示す Content-Length ヘッダの値を修正していなかった（赤字部分）。そのため、Content-Length ヘッダの値を解釈する過程で問題が生じた可能性もある（実際には分からない）。

```
POST /UPnPControl HTTP/1.1
SOAPACTION: "urn:schemas:service:X_UPnPControl:1#X_UPnPControlAction"
CONTENT-TYPE: text/xml ; charset="utf-8"
HOST: 192.168.11.1:12345
Content-Length: 383

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:s="h
ttp://schemas.xmlsoap.org/soap/envelope/">
   <s:Body>
      <u:X_UPnPControlAction xmlns:u="urn:schemas:service:X_UPnPControl:1">
         <Element1>a</Element1>
         <Element2>      </Element2>
      </u:X_UPnPControlAction>
   </s:Body>
</s:Envelope>
```

「00 00 00 00」

4.4.2. UPnP リクエストのテストデータ （２）

テストデータ(1)と同様に、このテストデータも UPnP に対応した製品 192.168.11.1 に送信された UPnP リクエストである。このテストデータの場合、特定の値に置き換えるのではなく、「Body」部分(赤字部分)を 15 回繰り返した。

```
POST /UPnPControl HTTP/1.1
SOAPACTION: "urn:schemas:service:X_UPnPControl:1#X_UPnPControlAction"
CONTENT-TYPE: text/xml ; charset="utf-8"
HOST: 192.168.11.1:12345
Content-Length: 4935

<?xml version="1.0" ?><s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/" xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<u:X_UPnPControlAction xmlns:u="urn:schemas:service:X_UPnPControl:1">
<Element1>a</Element1>
<Element2>a</Element2>
</u:X_UPnPControlAction>
</s:Body>
</s:Envelope>
<?xml version="1.0" ?><s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/" xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<u:X_UPnPControlAction xmlns:u="urn:schemas:service:X_UPnPControl:1">
<Element1>a</Element1>
<Element2>a</Element2>
</u:X_UPnPControlAction>
</s:Body>
</s:Envelope>
...
（赤字部分を 12 回繰り返し）
...
<?xml version="1.0" ?><s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/" xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<u:X_UPnPControlAction xmlns:u="urn:schemas:service:X_UPnPControl:1">
<Element1>a</Element1>
<Element2>a</Element2>
</u:X_UPnPControlAction>
</s:Body>
</s:Envelope>
```

4.5. JPEG 画像のテストデータ

この節では、Exif 形式の要素を細工した JPEG 画像（1）、（2）を紹介する。表 4-6 はテストデータ（1）の細工箇所と細工方法、表 4-7 はテストデータ（2）の細工箇所と細工方法を示している。

表 4-6 JPEG 画像（Exif 形式）のテストデータ（1）

データの「どの部分」	「SOF」セグメントの「垂直ライン数」
「どのように」細工したか	「特定の値」（特定の脆弱性検出に特化した値）に細工した。 関連： 3.2.2(a)

表 4-7 JPEG 画像（Exif 形式）のテストデータ（2）

データの「どの部分」	Exif 形式のセグメント構造
「どのように」細工したか	「データ構造」そのものを細工した。 関連： 3.2.2(b) ④

4.5.1. JPEG 画像（Exif 形式）のテストデータ（1）

このテストデータは、「SOF0」セグメントの垂直ライン数を細工した JPEG 画像である。

図 19 に、このテストデータ（1）の細工箇所と細工方法を図示した。Exif 形式には、画像データに関する情報（画像の幅や高さなど）を記録する Start of Frame（SOF）セグメントという領域がある。そして、この SOF セグメントには、画像の高さを意味する「垂直ライン数」が含まれている。元の JPEG 画像の垂直ライン数は 480（単位：pixel）であったが（図 19 の青色部分）、これを「0」に置き換えた（図 19 の赤色部分）。

図 20 は、JPEG 画像解析ツール「Jpeg Analyzer plus」[24]でこのテストデータを表示した様子を示している。図 20 の赤枠部分に注目してほしい。「640[0] x 0[0] pixel ...」という文字列を確認できる。この「0[0] pixel」が画像の垂直ライン数を示している。

[24] デジカメ画像の Exif 情報を詳細に表示する画像ファイル解析ソフト JpegAnalyzer Plus
http://homepage3.nifty.com/kamisaka/JpegAnalyzer/
ダウンロード URL
http://www.vector.co.jp/soft/dl/win95/art/se257653.html

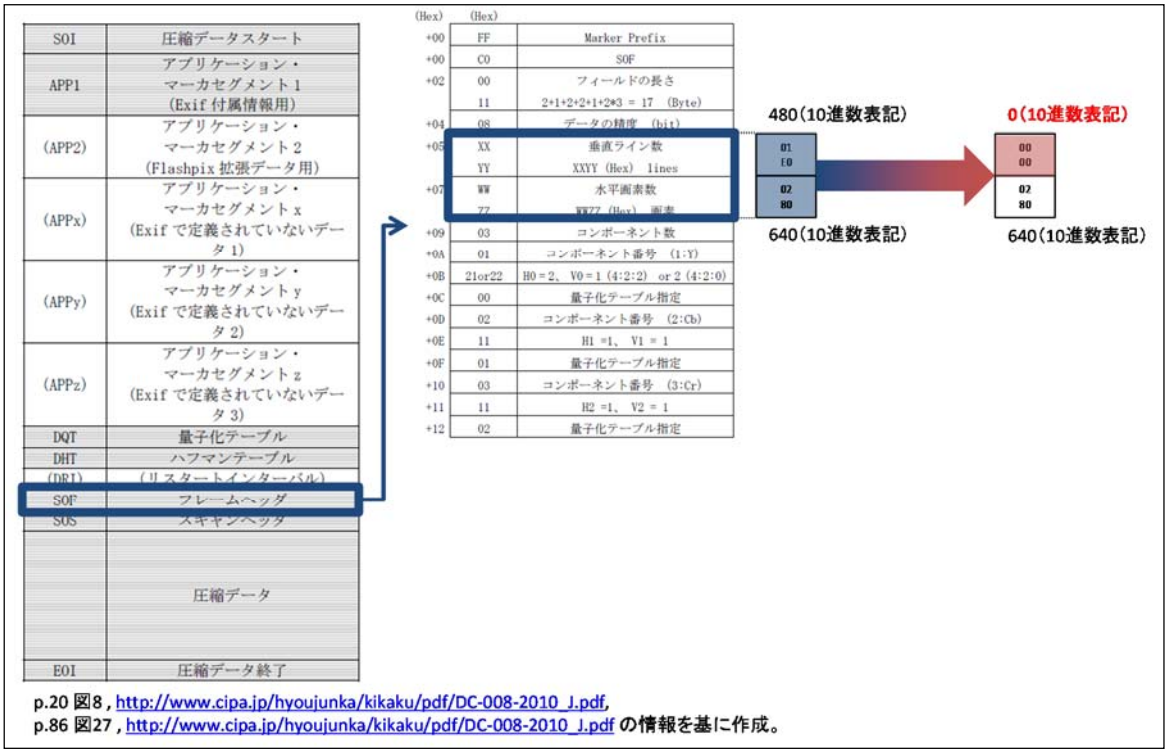


図 19 テストデータ（1）の細工箇所と細工方法

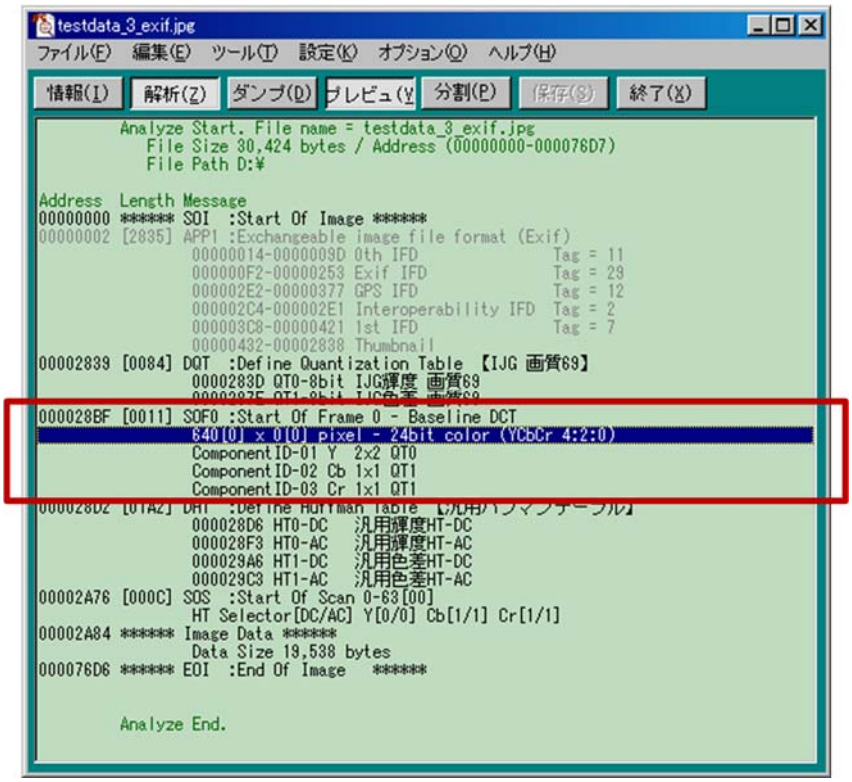


図 20 「JpegAnalyzer Plus」でテストデータ（1）を表示した様子

4.5.2. JPEG 画像（Exif 形式）のテストデータ（２）

このテストデータは、Exif 形式の規格[25]と異なるようにセグメント構造を細工した JPEG 画像である。

図 21 は、テストデータ(2)のセグメント構造を示している。このセグメント構造を Exif 形式の規格と照らし合わせてみると、次の違いがある。

- 画像の先頭を示す Start of Image(SOI)セグメントの次に、SOF0 セグメント[26]が存在する。
  Exif 形式の規格によると、SOI セグメントの後に APP1 セグメントが続く。しかし、このテストデータでは SOI セグメントの後に SOF0 セグメントが続いている。
- Application Segment 1(APP1)セグメントだけではなく、Application Segment 0(APP0)が存在する。
  Exif 形式の規格では、APP1 セグメントに言及しているが、APP0 セグメント[27]には言及していない。

図 22 は、「Jpeg Analyzer plus」でこのテストデータを表示した様子を示している。図 22 の赤枠部分に注目してほしい。「SOF0」と「APP0」という文字列が確認できるだろう。また、図 22 の中央部分の反転部分をみていただくと、この部分でも「SOF0」という文字列を確認できる。このテストデータには、SOF0 セグメントが 2 つ存在している。また、「Jpeg Analyzer plus」も警告メッセージを出力しているように、SOI セグメントの後に続く SOF0 セグメントが異常な値となっていた。

結果的に、このテストデータのどの値で問題が生じたか、実際には分からない。

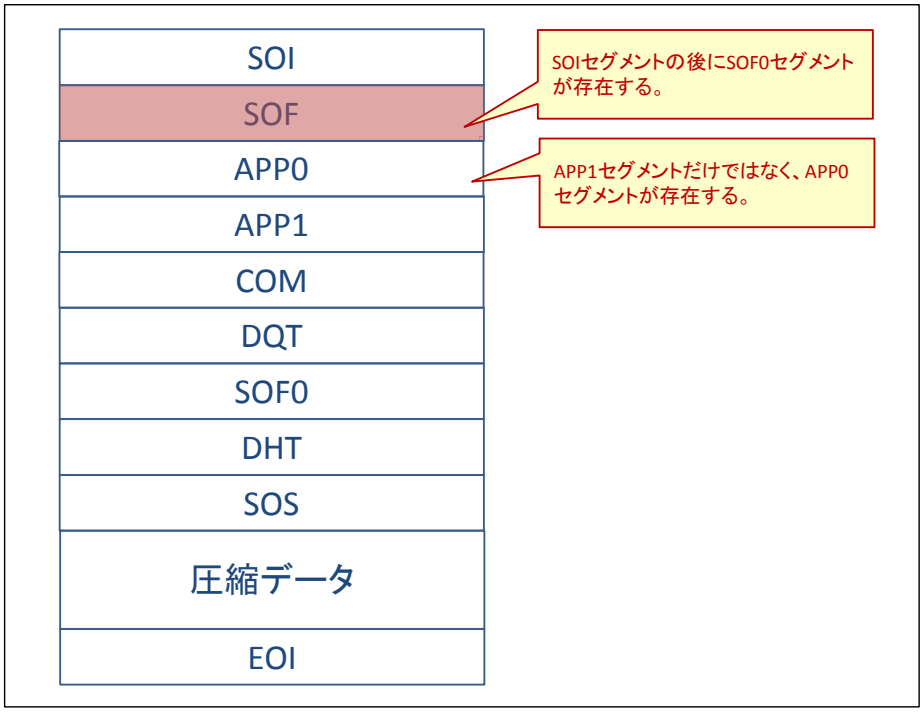


図 21 テストデータ（２）のセグメント構造

---

25 デジタルスチルカメラ用 画像ファイルフォーマット規格 Exif 2.3
http://www.cipa.jp/std/documents/j/DC-008-2012_J.pdf

26 SOF セグメントには、画像データの符号化方式などの違いによって複数の種類がある。SOF0 セグメントはそのうちの一つである。

27 JPEG 画像のデータ形式の一つである「JFIF 形式」が定義しているセグメントである。

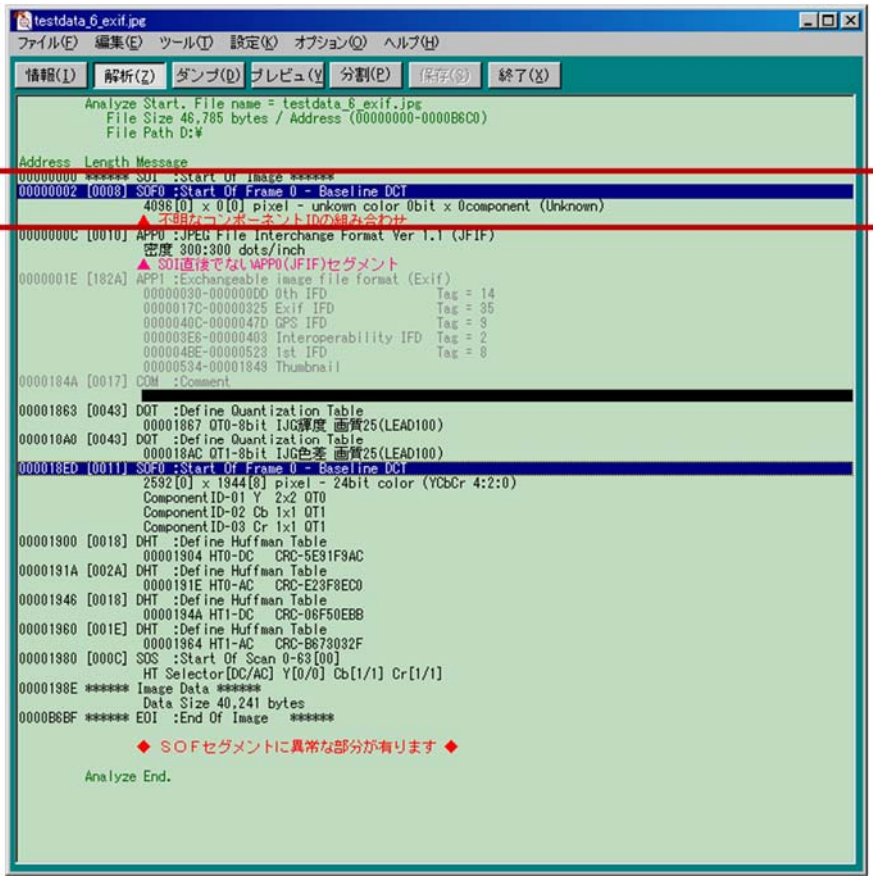


図 22 「JpegAnalyzer Plus」でテストデータ（2）を表示した様子[28]

[28] コメントが含まれる COM セグメントの値には、ファジングツールを示す文字列が含まれていたため、伏字とさせていただいた。

4.6. 無線 LAN フレームのテストデータ

この節では、「Information Element」（以降、「IE」）をランダムに細工した無線 LAN フレームを紹介する。表 4-8 は、このテストデータの細工箇所と細工方法を示している。

表 4-8 無線 LAN フレームのテストデータ

データの「どの部分」	無線 LAN フレームの「Information Element」
「どのように」細工したか	「データ構造」そのものを細工した。 関連： 3.2.2(b) ①、③

このテストデータは、MAC アドレス ff:ff:ff:ff:ff:ff に送信された無線 LAN フレームである。図 23 は、このテストデータの一部を「Wireshark」で表示した様子を示している。このテストデータには、無線 LAN の規格「IEEE Std 802.11」[29]で取り扱いが明確に決まっていない「IE」や、実際のデータと矛盾するデータ長が設定された「IE」が複数含まれていた。

規格で取り扱いが明確に決まっていない「IE」、実際のデータと矛盾するデータ長が設定された「IE」をそれぞれ Tag Number「129」の「IE」（図 23 青枠部分）、Tag Number「15」の「Schedule element」（図 23 赤枠部分）で例示しよう。

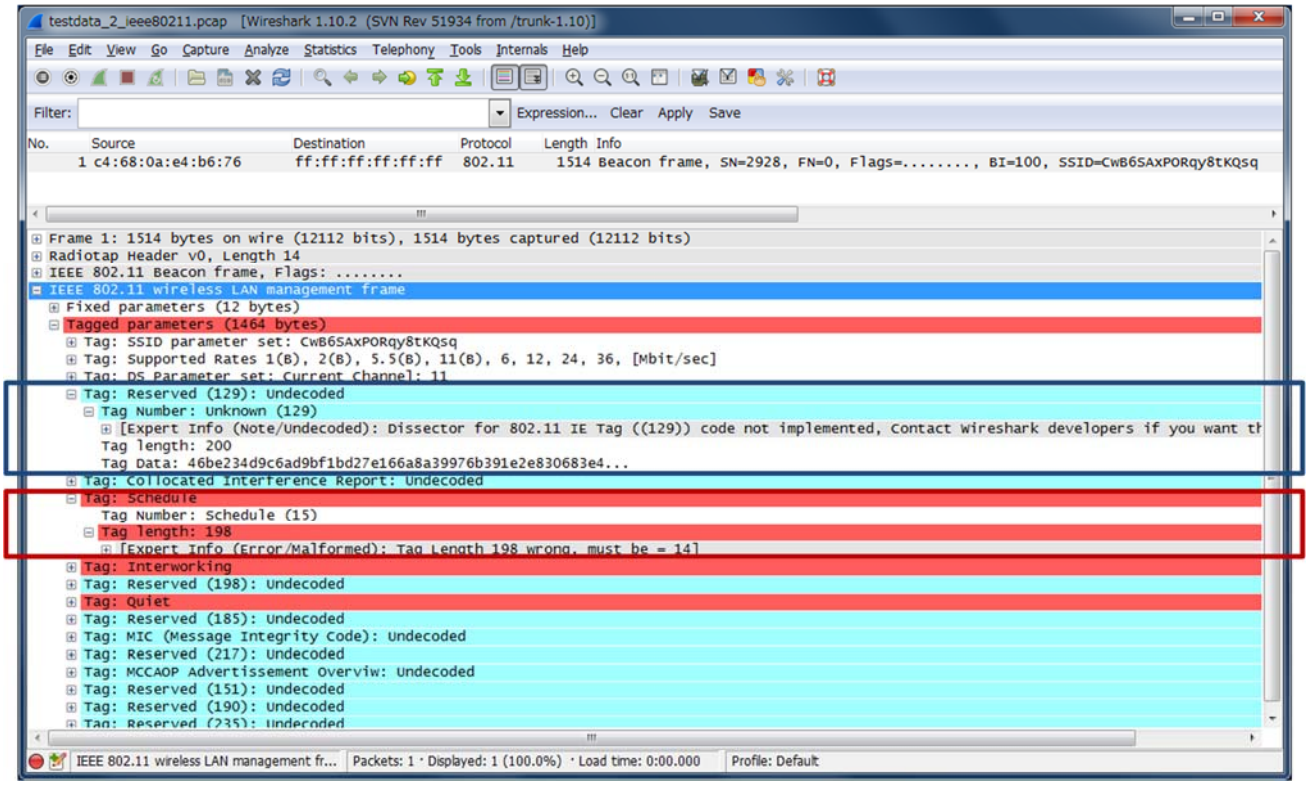


図 23 ランダムに細工した無線 LAN フレーム

[29] IEEE Std 802.11-2012
http://standards.ieee.org/findstds/standard/802.11-2012.html

まず、Tag Number「129」の「IE」をみてみよう。図 24 は、図 23 の Tag Number「129」の「IE」を拡大した様子を示している。図 24 から、この「IE」のデータ長が 200 で、値が 46be2…であることを読み取れる。

無線 LAN の規格「IEEE Std 802.11」では、この「IE」を「Reserved」(予約)と規定している[30]。したがって、Tag Number「129」の「IE」の取り扱いが規格で明確に決まっていないため、製品によってはこの「IE」を解釈するときに問題が起きる可能性がある。

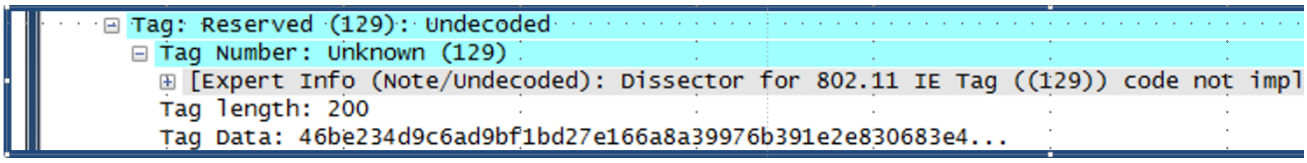


図 24 Tag Number「129」の「IE」のデータ長と値

続いて、Tag Number「15」の「Schedule element」をみてみよう。図 25 は、図 23 の Tag Number「15」の「Schedule element」を拡大した様子を示している。図 25 から、「Tag Length 198 wrong, must be = 14」という文字列を読み取れる。

無線 LAN の規格「IEEE Std 802.11」では、この「Schedule element」のデータ長を 14 バイト[31]と規定している[32]。しかし、このテストデータにはデータ長に 14 バイトよりも大きい 198 バイトが設定されている。したがって、テストデータの「Schedule element」には実際のデータの長さと異なるデータ長が設定されているため、製品によってはこれを解釈するときに問題が起きる可能性がある。

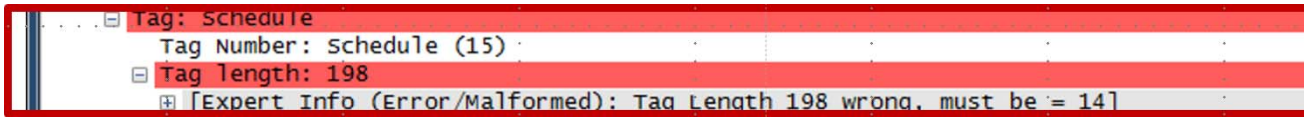


図 25 「Schedule element」のデータ長と値

[30] p.474 8.4.2.1 General, "IEEE Std 802.11-2012"
[31] 「IEEE Std 802.11-2012」では、厳密には Length として「Tag Number」の 1 バイト、Length の 1 バイトの計 2 バイトを除いた「12」バイトを定義している。
[32] p.579 8.4.2.36 Schedule element, "IEEE Std 802.11-2012"

5. テストデータに関する知識の活用

3 章、4 章でファジングのテストデータにおける考え方、およびテストデータの実例を紹介してきた。これによって、ファジングツールがどのようなテストデータを作るのか、理解いただけたと思う。

ファジングツールは、対象製品が受け取るデータを様々な方法で細工してテストデータを作る。しかし、あらゆるテストデータでファジングを実施すると膨大な時間がかかってしまう。そこで、様々な考え方でファジングツールが開発されている[33]。例えば、「時間が掛かったとしても、可能な限り多くのテストデータでファジングを実施する」ツールがあれば、「時間短縮を意識して、データ構造の一部分のみ細工したテストデータでファジングを実施する」ツールもある。

ファジングツールのテストデータを把握すると、テストデータの網羅性を高めて、効果的なファジングを実践できる。さらに、本書で学んだ知識をファジングツールの独自開発に役立てることもできる。

5.1. 一歩進んだ知識の活用：ファジングツールの独自開発

製品のなかには、既存のファジングツールでその製品に合わせたテストデータを送れずに、既存のツールでファジングを実施できないものがある。このような製品に対してファジングを実施する場合、製品開発企業および研究者が製品に合わせたファジングツールを独自に開発する方法がある。

これまでは、ファジングにおける「テストデータ」を体系的にまとめている資料は少なかったため、製品に合わせたテストデータを検討することが難しかったと考える。実際に、ファジングを実践している製品企業から、「自社製品向けのファジングツールを開発したいが、自社の技術者ではファジングツールの開発が難しい」という課題を伺った。

そこで、本書で紹介したテストデータの考え方（データ構造の「どの部分」を「どのように」細工するか）を、製品に合わせた「テストデータ」の検討に活用いただけるだろう。IPA でも活動で培ったテストデータの知識をふまえて、JPEG テスト支援ツール「iFuzzMaker」を開発した。本書がファジングツールの独自開発の一助となることを期待している。

[33] 商用製品「FFR Raven」、「Codenomicon Defensics」、オープンソースソフトウェアの「Taof」の 3 つのファジングツールの特徴を以下のレポートで考察している。
IPA：「製品の品質を確保する『セキュリティテスト』に関するレポート」
4.5 [2nd ステップ]商用製品を活用したファジング
https://www.ipa.go.jp/about/technicalwatch/20120920.html

5.1.1. 活用事例：JPEG テスト支援ツール「iFuzzMaker」

「iFuzzMaker」とは、「JPEG 画像を読み込む機能」を持つ製品に対するファジングを支援するツールである。この「iFuzzMaker」では、それらの製品に対するファジングで使うテストデータを作ることができる。図 26 に、「iFuzzMaker」で作ったテストデータを使ったファジングのイメージを図示した。

幅広く製品開発者に活用されるよう、IPA では利用マニュアルとともに「iFuzzMaker」をオープンソースソフトウェアとして公開している。

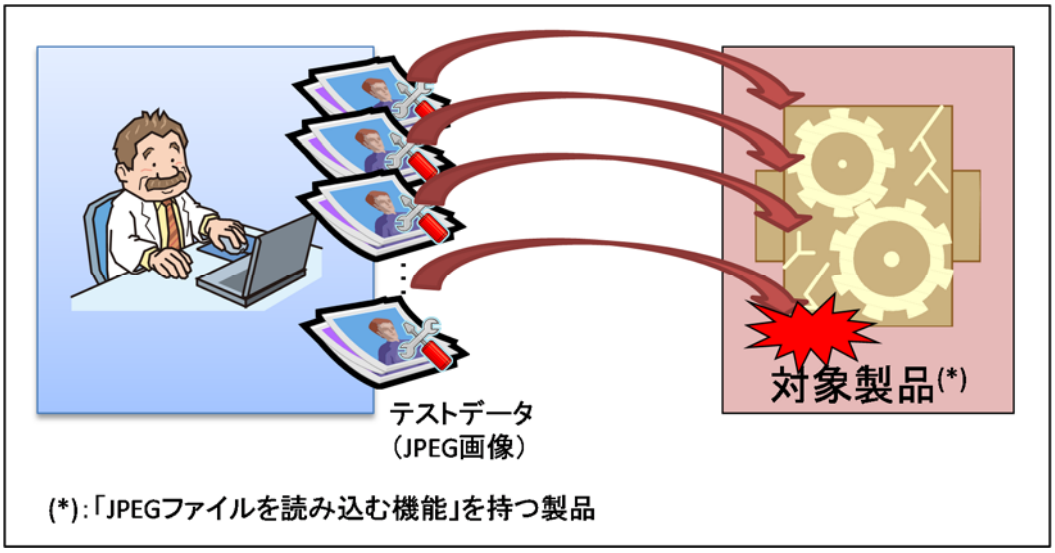


図 26 「iFuzzMaker」で作ったテストデータを使ったファジング（イメージ図）

■ 「iFuzzMaker」の概要

https://www.ipa.go.jp/security/vuln/iFuzzMaker/index.html

対象利用者	JPEG 画像を扱う情報家電やソフトウェア製品の関係者 （開発者や品質保証担当者などを想定）
動作環境	OS　：Windows 7 SP1 Windows 8.1 Windows 10 上記 OS は、32bit 版と 64bit 版で動作確認済み。 CPU　：1GHz 以上の x86 互換プロセッサ メモリ：1GB 以上の空きメモリ HDD　：1GB 以上の空き領域
機能	● JPEG 画像を読み込む機能に対するファジングで使う、テスト JPEG 画像を作ること ● 利用者が指定した Exif 形式の JPEG 画像を作ること

6. おわりに

2011 年 8 月からファジングなどの脆弱性検出技術の普及推進を目的とした「脆弱性検出の普及活動」を IPA が始めてから、2 年以上が経過した。活動当初よりも「ファジング」を知っている技術者もふえ、少しずつ日本でも製品開発にファジングの導入を検討する企業が出てきた。しかし、本書で取り上げたファジングにおけるテストデータの理解など、企業が製品開発の現場でファジングを活用するときの課題はいくつもある。今後も IPA ではファジングの利用が広がるための活動を継続していきたい。

本書が、製品開発におけるファジングの活用、ひいては製品開発における脆弱性低減の一助となれば幸いである。

7. 本書に関連する仕様／規格

仕様名／規格名	URL
Exif (Exchangeable image file format)	デジタルスチルカメラ用画像ファイルフォーマット規格 Exif 2.3： http://www.cipa.jp/std/documents/j/DC-008-2012_J.pdf（2016 年 2 月確認）
HTTP (Hypertext Transfer Protocol)	RFC2616： http://tools.ietf.org/html/rfc2616（2016 年 2 月確認）
IEEE802.11	IEEE802.11-2012： http://standards.ieee.org/findstds/standard/802.11-2012.html （2016 年 2 月確認）
IP (Internet Protocol)	RFC791： http://tools.ietf.org/html/rfc791（2016 年 2 月確認）
TCP (Transmission Control Protocol)	RFC793： http://tools.ietf.org/html/rfc793（2016 年 2 月確認）
UPnP (Universal Plug and Play)	UPnP Device Architecture 1.1： http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf （2016 年 2 月確認）

付録：検出できるテストデータが分かる脆弱性情報

IPA では、US-CERT[34]と Exploit Database[35]の 2 つのウェブサイトを中心に、2011 年 1 月から 2013 年 6 月の期間に公開された脆弱性情報の中から、脆弱性の説明や実証コードからその脆弱性を検出できるテストデータが分かるものを調査した。下表はその調査結果を示している。

本付録をご活用いただく場合、「対象とした期間に公表された脆弱性情報をすべて調査したわけではないこと」に注意していただきたい。

No	製品名	製品種別	脆弱性を検出できるテストデータ（URLの情報をもとに作成）	URL	脆弱性が存在した機能	脆弱性が存在した箇所
1	WL2600CAM POE2600HDなど	IPカメラ	/aaa・・・(3000個a)	http://www.exploit-db.com/exploits/26174/	ウェブサービス機能	HTTPリクエストラインのURI
2	airCamシリーズ	IPカメラ	rtsp://<IPアドレス>/AAA・・・(271個のA)0x78 0x56 0x34 0x12/live/ch00_0	http://www.coresecurity.com/advisories/buffer-overflow-ubiquiti-aircam-rtsp-service	ストリーミング機能 （RTSP機能）	DESCRIBEリクエストにおけるURI
3	Xpient POS system	POSシステム	1 1	http://www.coresecurity.com/advisories/xpient-cash-drawer-operation-vulnerability	独自サービス （7510/tcp）	データ全体 （データを受け付けることを想定しないようである）
4	AR1220	ルータ	文字列「4096個の「A」」	http://blog.emaze.net/2013/05/multiple-buffer-overflows-on-huawei.html	SNMP機能	SNMPv3 USMの AuthoritativeEngineID および UserName
5	MayGion IP cameras	IPカメラ	/AAA・・・(3000個A).html	http://www.coresecurity.com/advisories/maygion-ip-cameras-multiple-vulnerabilities	ウェブサービス機能	HTTPリクエストラインのURI
6	PT7135	IPカメラ	文字列「1000個の「a」」 文字列「10000個の「a」」	http://www.coresecurity.com/advisories/vivotek-ip-cameras-multiple-vulnerabilities	ストリーミング機能 （RTSP機能）	DESCRIBEリクエストにおける Authorizationヘッダ
7	D6000	スマートテレビ	・「¥n¥n¥n¥t¥t¥t¥tHACKED!¥n¥n¥n¥n¥n」 ・文字列「1040個の「a」」	http://www.exploit-db.com/exploits/18751/	スマフォアプリとの連携機能 （55000/tcp）	連携するためのプロトコルにおける name値およびMACアドレス値
8	MegaDome AV1355DN	IPカメラ	どんなデータでも構わない （NmapやNessusのテストデータでもOK）	http://www.kb.cert.org/vuls/id/375180	DHCPサーバ機能 （69/udp）	データ全体 （データを受け付けることを想定しないようである）
9	TL-WR740N	ルータ	「/...」（ドット「.」を3つ付与したもの）	http://www.zeroscience.mk/en/vulnerabilities/ZSL-2013-5135.php	ウェブサービス機能	HTTP リクエストラインのURI
10	Broadcom UPnP Stack	UPnPライブラリ	書式文字列	http://www.defensecode.com/public/DefenseCode_Broadcom_Security_Advisory.pdf	UPnP機能	UPnP ControlにおけるSetConnectionType」 アクションのNewConnectionType要素
11	Intel/Portable SDK for UPnP Device	UPnPライブラリ	uuid:schemas:device:AAA....AAA:anything （値の一部分に極端に長い文字列を含む）	https://community.rapid7.com/docs/DOC-2150	UPnP機能	SSDPにおけるM-SEARCHリクエストに おける STヘッダ
12	MiniUPnP	UPnPライブラリ	改行コード（CR+LF）を除外した値	https://community.rapid7.com/docs/DOC-2150	UPnP機能	SSDPにおけるM-SEARCHリクエストに おける STヘッダ
13			・過度に長いクォートされたメソッド ・# （シャープ記号）を欠いた SOAPAction ヘッダ ・" （二重引用符）を欠いた SOAPAction ヘッダ		UPnP機能	UPnP ControlにおけるSOAPActionヘッダ
14	複数のプリンタ	プリンタ	PCLで規定されていない文字（制御文字など）	https://viaforensics.com/security/exploiting-printers-via-jetdirect-vulns.html	JetDirect機能	PCL (Printer Control Language)で書かれたデータ
15	E585u-82	モバイルルータ	「Authorization: Basic¥r¥n¥r¥n」	http://www.kb.cert.org/vuls/id/871148	ウェブサービス機能	HTTP Authorizationヘッダ
16	BCM4325 BCM4329	無線LANチップセット	「65535」(0xff 0xff)	http://www.coresecurity.com/content/broadcom-input-validation-BCM4325-BCM4329	無線LAN機能	IEEE802.11のBeaconフレームにおける RSN Information Element(IE) のAuthentication suites count
17	KDL-32CX525	スマートテレビ	一定数のTCP-SYNセグメント	http://www.exploit-db.com/exploits/18705/	TCP/IP処理機能	TCPセグメント
18	Cisco TelePresence Series	ビデオ会議システム	4x8 a"s	http://www.exploit-db.com/exploits/17871/	SIP処理機能	SIP INVITEリクエストのMACアドレス
19	SoundPoint IP 335	IP電話	「Authorization: Basic¥r¥n¥r¥n」	http://www.exploit-db.com/exploits/16068/	ウェブサービス機能	HTTP Authorizationヘッダ
20	CMNC-200 Full HD	IPカメラ	文字列「512個の「A」」	https://www.trustwave.com/spiderlabs/advisories/TWSL2010-006.txt	ウェブサービス機能	HTTP Cookieヘッダ

以上

34 http://www.kb.cert.org/vuls/
35 https://www.exploit-db.com/

更新履歴

更新日	更新内容
2013 年11月 7 日	第1版 発行。
2016 年 3 月 31 日	第1版 第2刷発行。 p.41「iFuzzMaker の概要」更新。 p.42「本書に関する仕様／規格」更新。 脚注内 URL 修正

編集責任　　金野 千里

執筆者　　勝海 直人
協力者　　鵜飼 裕二　　園田 道夫
澤田 迅（株式会社ＩＴ働楽研究所）

栗栖 正典　　板橋 博之　　岡崎 圭輔　　山下 勇太
相馬 基邦

ファジング実践資料（テストデータ編）

～ 効果的なファジングツールの選定につながるテストデータの理解 ～

［発　　行］2013 年 11 月 7 日 第１版
2016 年 3 月 31 日 第１版 第２刷

［著作・制作］独立行政法人情報処理推進機構 技術本部 セキュリティセンター
情報セキュリティ技術ラボラトリー