

ファジング実践資料

「ファジング活用の手引き」別冊
ファジングツールの使い方とファジング結果の再現手順



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

<https://www.ipa.go.jp/security/vuln/fuzzing.html>

目次

目次.....	1
はじめに	2
本書の使い方	2
本書の動作環境.....	2
本書を読む上での注意事項	2
1. ファジングツール「Taof」の使い方	3
1.1. 概要	3
1.2. Taofによるファジングの流れ.....	3
1.3. 使い方	6
1.4. ファジングの実践.....	15
1.5. Taofが生成するファズデータ	18
2. ファジングツール「Peach」の使い方	22
2.1. 概要	22
2.2. Peachによるファジングの流れ.....	23
2.3. 使い方	23
2.4. 設定ファイルについて.....	25
2.5. ファジングの実践.....	31
2.6. Peachが生成するファズデータ	34
3. ファジング結果の再現手順.....	37
3.1. 概要	37
3.2. ファジング結果の再現環境.....	37
3.3. ファジング結果の再現手順.....	42

はじめに

本書は、IPA の「ファジング活用の手引き」の別冊資料です。本書には、ファジングを実践するための「ファジングツールの使い方」と「ファジングツールを使わずにファジング結果（パケットキャプチャファイル）を再現する手順（ファジング結果の再現手順）」を収録しています。

「ファジングツールの使い方」では、IPA の「脆弱性検出の普及活動」において脆弱性の検出実績があるファジングツール「Taof」、「Peach Community edition」（以下、Peach と略す）の基本的な使い方を紹介します。

「ファジング結果の再現手順」では、パケットキャプチャツール「Wireshark」で取得したファジング実行時のパケットから、ファジングの結果を再現する手順を紹介します。

本書の使い方

本書は次の 3 つの使い方を想定しています。

(1). ファジングを試してみたい。

「1 ファジングツール「Taof」の使い方」と「2 ファジングツール「Peach」の使い方」の手順にしたがって、ファジングを実践できます。

(2). ファジングで使われているファズデータがどんなものか知りたい。

「1.5 Taof が生成するファズデータ」と「2.6 Peach が生成するファズデータ」を読むと実際にファジングツールで使われているファズデータを確認できます。

(3). ファジングツールを使わずにファジング結果を再現したい。

「3 ファジング結果の再現手順」の手順にしたがい、パケットキャプチャファイルからファジング結果を再現できます。この手順で再現できるファジング結果は、ネットワークを介したファジングのみである点に注意してください。

本書の動作環境

本書で収録している「ファジングツールの使い方」および「ファジング結果の再現手順」は、Windows 7 Service Pack1(SP1)上で動作確認しています。

本書を読む上での注意事項

- 本書で紹介するファジングツールの機能には、IPA が使用していない機能があります。それら機能に関しては、本文にて「**【IPA 未使用】**」と明記しています。それらの機能に関してはソフトウェアのドキュメントをご確認ください。
- 本書で収録している URL および手順は、2017 年 2 月末日時点で確認しています。読者の動作環境などによっては本書の手順では正しく動作しない可能性があります。

1. ファジングツール「Taof」の使い方

本章ではファジングツール「Taof」の概要と使い方、「Taof」によるファジング手順、「Taof」が生成するファズデータを説明します。

1.1. 概要

「Taof」は GNU General Public License (GPL) に基づいて提供されているオープンソースソフトウェアです。Taof は Windows 上で動作し、HTTP サーバや FTP サーバなどに対してネットワークを介したファジングを実践できます。

表 1.1-1 Taof の概要

項目	内容
IPA 使用バージョン	0.3.2
ライセンス	GNU General Public License (GPL)
ファジング手法	ネットワークを介したファジング
URL	https://sourceforge.net/projects/taof/ (2017年2月確認)
ダウンロード先	https://sourceforge.net/projects/taof/files/Taof%20Windows%20Binary/taof-0.3Win32/taof-0.3.2_Win32.zip/download
特記事項	2007年2月以降、アップデートされていない。

1.2. Taof によるファジングの流れ

Taof によるファジングではクライアント⇄サーバ間のリクエストデータ（クライアントからサーバへの送信データ）をキャプチャして使用します。キャプチャしたリクエストでファズデータに置き換えるポイント（ファジングポイント）を設定して、そのポイントを自動的にファジングデータに置き換えて送信します。

Taof によるファジングは具体的には「リクエストデータの収集」、「ファジングポイントの設定」、「ファジングの実行」の3つの工程に分けて実施します。

(1) リクエストデータの収集

クライアント⇄サーバの間を Taof で中継し、リクエストデータを収集します。パケットの収集イメージを図 1-1 に示します。

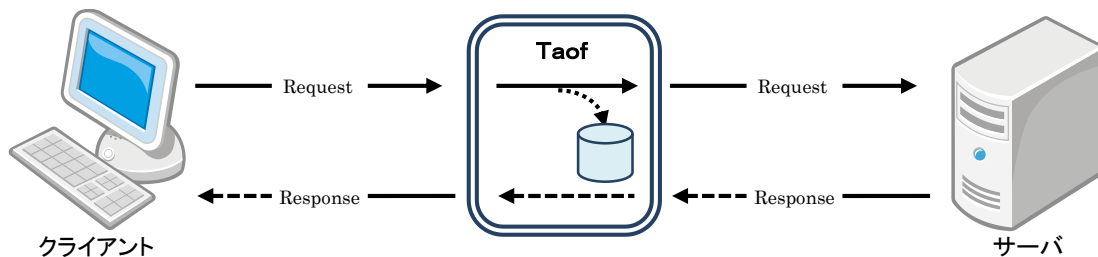


図 1-1 パケット収集イメージ

Taof は指定した IP アドレスとポート番号でクライアントからのリクエストを待ち受けます。クライアントが Taof にリクエストを送信すると、Taof がそのリクエストをサーバに送信します。Taof が中継することによって、クライアント⇄サーバ間の通信がクライアント⇄Taof、Taof⇄サーバの 2 つに分かれることになります。

(2) ファジングポイントの設定

収集したリクエストから、ファジングポイントを設定します。Taof によってファジング実行時にこのファジングポイントが自動的に変更して送信します。ファジングポイントのデータをファズデータに置き換えるイメージを図 1-2 に示します。

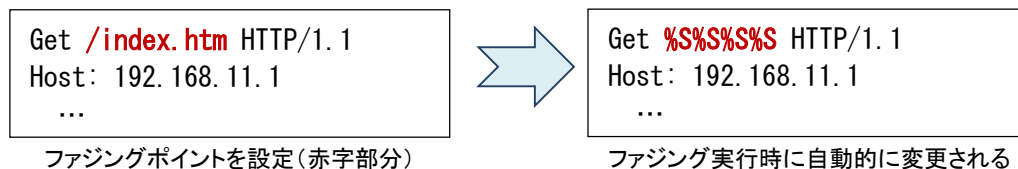


図 1-2 ファジングポイント変更イメージ

(3) ファジングの実行

ファジングポイントを指定したリクエストを元に、サーバへファジングを実行します。Taofはファジングデータを自動的に生成し送信します。ファジングが完了するとTaofはその結果を表示します。ファジング実行結果の例を図1-3、図1-4に示します。

```
Starting fuzzing session against 192.168.11.1:80 - 17:27:53
Check 'debugging' file for further information

Fuzzing request: 0
Number of fuzzing points: 1

+ Buffer overflows
*****
Fuzzing request: 1
Number of fuzzing points: 0
Fuzzing request: 2
Number of fuzzing points: 0

[*] Fuzzing session finished.
```

図 1-3 サーバが応答を返す場合

```
Starting fuzzing session against 192.168.11.1:80 - 17:31:18
Check 'debugging' file for further information

Fuzzing request: 0
Number of fuzzing points: 1

+ Buffer overflows
*****
+ Format Strings
*****
+ Integer Overflows
*****
[*] It was not possible to connect to 192.168.11.1:80. It might be down. Retrying now...

[*] It was not possible to connect to 192.168.11.1:80. It might be down. Retrying now...

[*] I could not connect to the server. I might have killed the service (which is good!).

[*] Fuzzing session because remote server is not responding.
```

図 1-4 サーバから応答がなくなった場合

1.3. 使い方

1.3.1. インストールとアンインストール

(1) インストール

Taof のインストールは ZIP ファイルを解凍するだけです。1.1 節に記載したダウンロード先から「taof-0.3.2_Win32.zip」をダウンロードし、展開します。展開先のフォルダには日本語のパスが含まれないようにします。(※Taofは日本語のパスに対応していません。)

(2) アンインストール

Taofはレジストリを使用しないので、フォルダごと削除することでアンインストールできます。

1.3.2. Taof の使用

Taofを起動するには、展開したフォルダ内の「taof.exe」を実行します。Taofは主に表 1.3-1の6つの画面で構成されています。

表 1.3-1 Taofの画面一覧

画面	内容
メイン画面	リクエストデータの一覧を表示します。
設定画面	ファジング中のタイムアウトや辞書ファイルを設定します
データ収集画面	クライアント・サーバ間のリクエストデータを収集します
ネットワーク設定画面	クライアント・サーバの IP アドレス、ポートを設定します
ファジングポイント設定画面	リクエストからファジングポイント設定します
ファジング実行画面	サーバへファジングを実行します

Taofは、図 1-5 のように画面が遷移します。Taofを終了するには、メイン画面で右上の×ボタンをクリックします。

これより各画面を説明していきます。なお、特に説明がいらないと判断した画面については説明していません。それらの画面については（本文に説明なし）と補足しています。

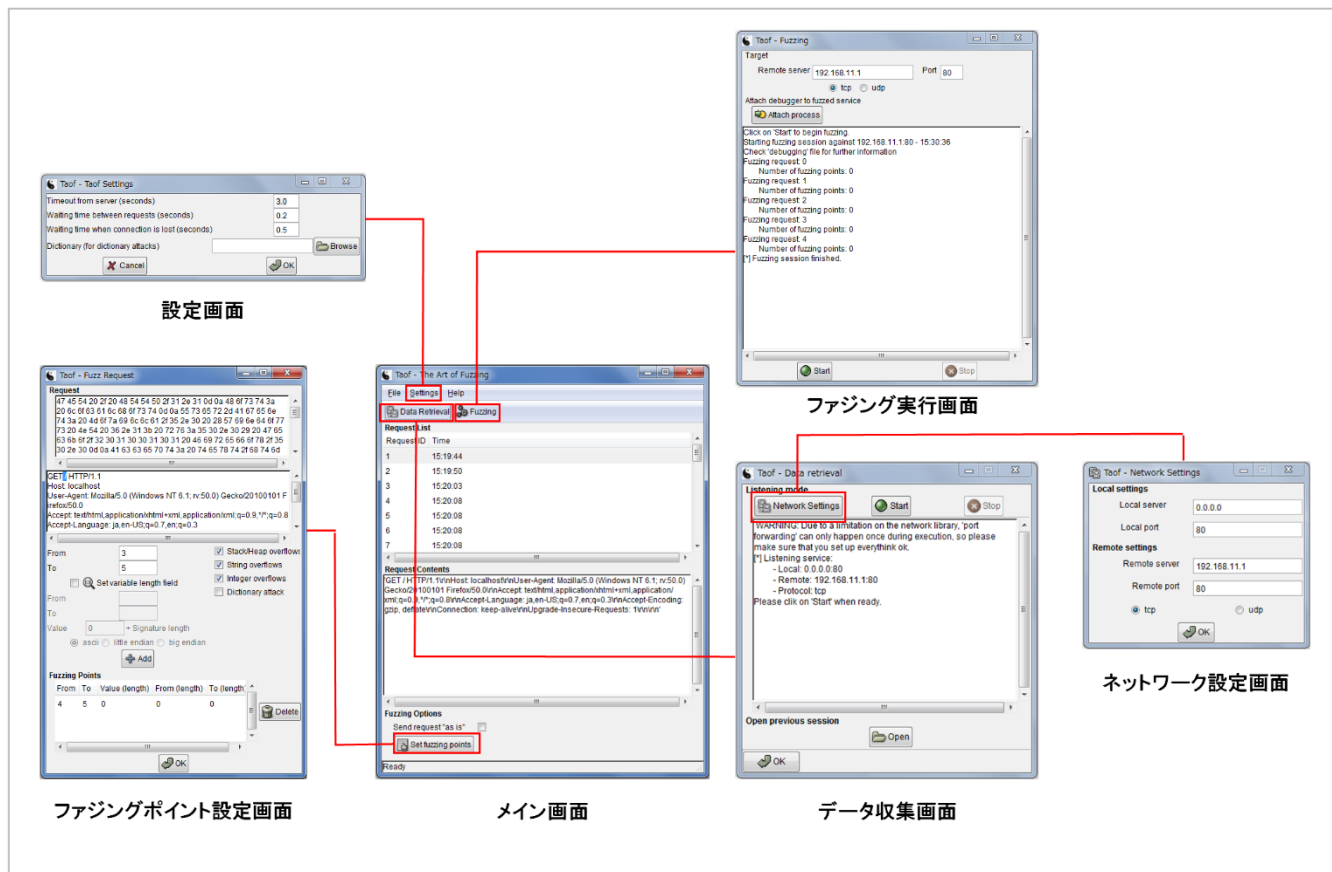


図 1-5 Taof の画面遷移

■ メイン画面

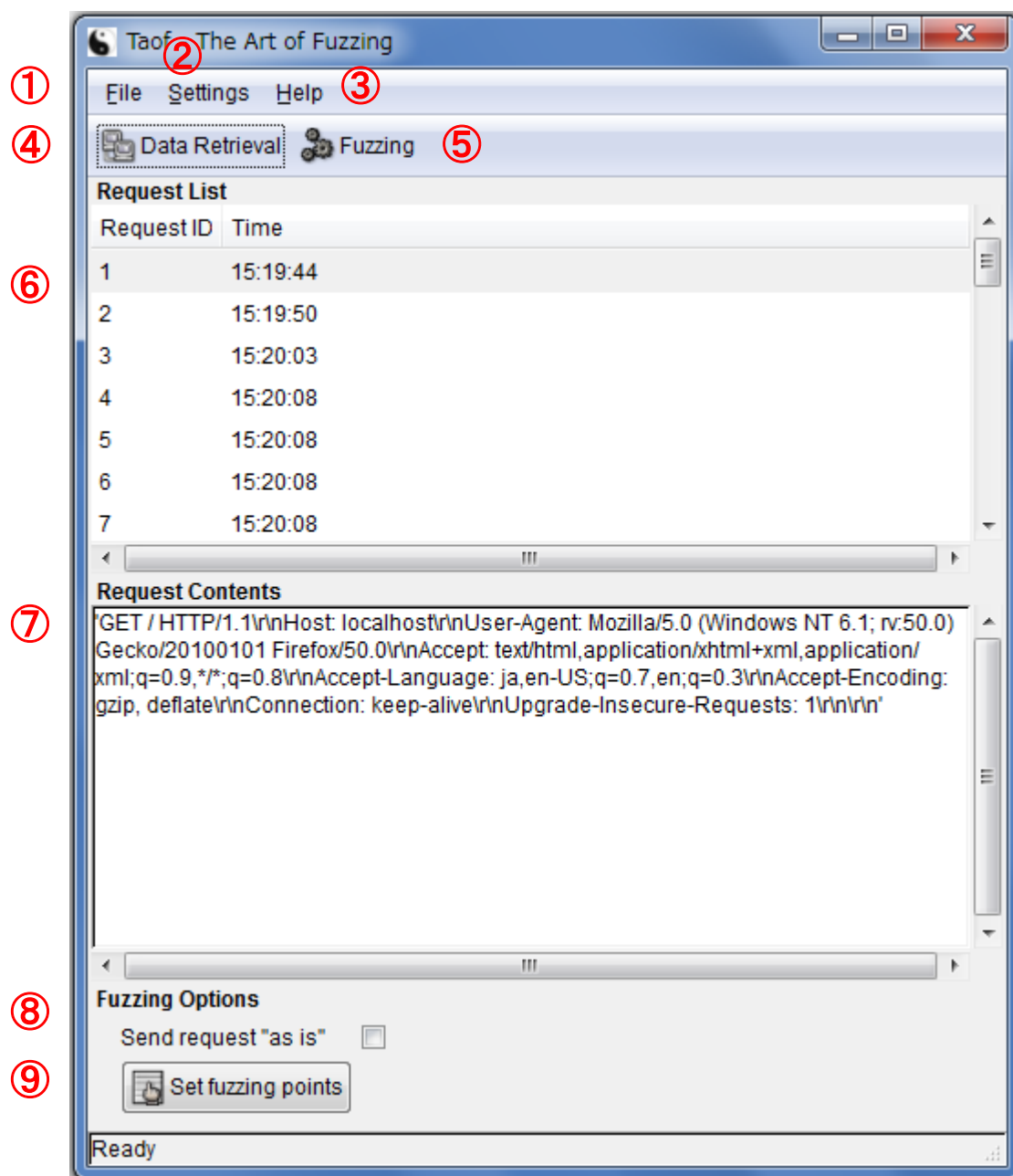


図 1-6 メイン画面

- ① ファイルメニュー ([File])
[Open]で過去のファuzzingデータ選択画面（本文に説明なし）を開きます。[Quit]で Taofを終了します。
- ② 設定メニュー ([Settings])
設定画面を開きます。
- ③ ヘルプメニュー ([Help])
[Help]で説明画面（Html形式ファイル、本文に説明なし）を開きます。[About]でバージョン表示画面（本文に説明なし）が開きます。

- ④ データ取得ボタン ([Data Retrieval])
データ収集画面を開きます。
- ⑤ ファジングボタン ([Fuzzing])
ファジング実行画面を開きます。
- ⑥ リクエスト一覧 ([Request List])
取得したリクエストを一覧表示します。
- ⑦ リクエスト内容 ([Request Contents])
選択されているリクエストの内容を表示します。
- ⑧ 未変更データ送信 ([Send request “as is”])
ファジング実行前に、ファジングポイントに変更を加えていないリクエストを一度だけ送信します。
- ⑨ ファジングポイント設定ボタン ([Set fuzzing points])
ファジングポイント設定画面を開きます。

■ 設定画面

ファジング中のタイムアウトや辞書ファイルを設定します。

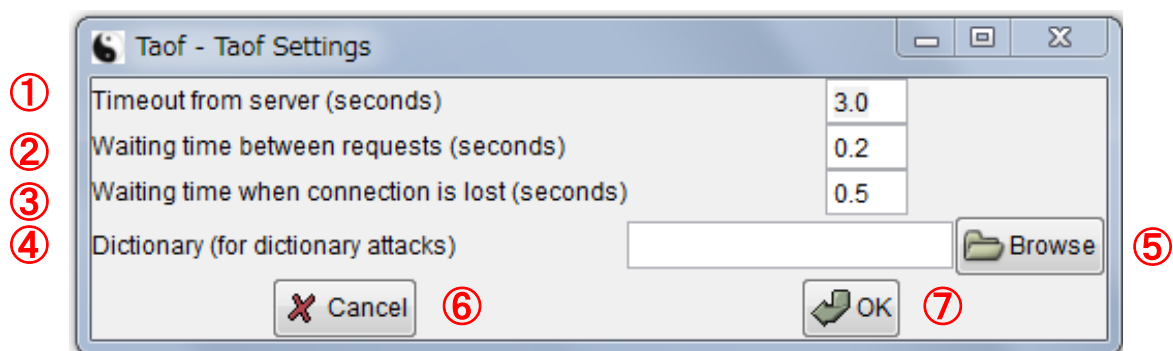


図 1-7 設定画面

- ① サーバタイムアウト設定 ([Timeout from server (seconds)])
ファジング実行中のサーバからの応答待機時間を設定します。(初期値は 3.0)
- ② リクエスト送信間隔 ([Waiting time between request (seconds)])
リクエストデータの送信間隔を設定します。(初期値は 0.2) [【IPA 未使用】](#)
- ③ コネクション喪失時タイムアウト設定 ([Waiting time when connection is lost (seconds)])
コネクション喪失時のタイムアウト時間を設定します。(初期値は 0.5) [【IPA 未使用】](#)
- ④ 辞書ファイル設定 ([Dictionary (for dictionary attacks)])
辞書ファイルを設定します。
- ⑤ 辞書ファイル選択ボタン ([Browse])
辞書ファイルの選択画面 (本文に説明なし) を開きます。
- ⑥ キャンセルボタン ([Cancel])
設定を反映せずに設定画面を閉じます。
- ⑦ OK ボタン ([OK])
設定を反映し、設定画面を閉じます。

■ データ収集画面

クライアント⇄サーバ間のリクエストデータを収集します。

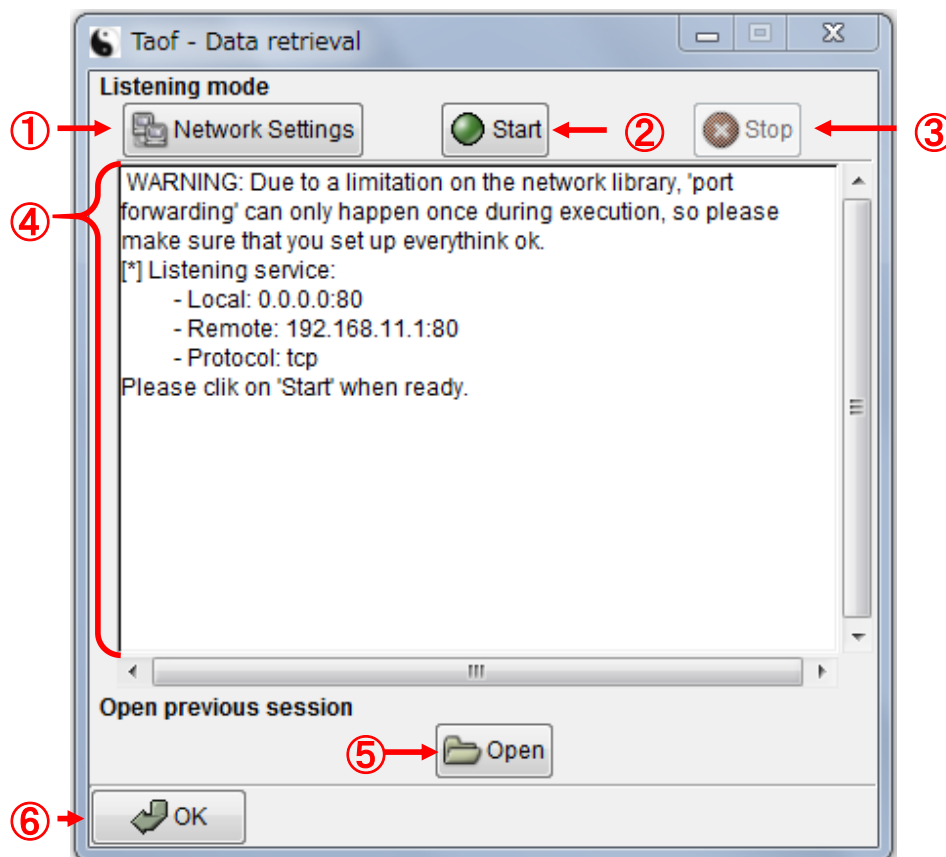


図 1-8 データ収集画面

- ① ネットワーク設定ボタン ([Network Settings])
ネットワーク設定画面を表示します。
- ② 収集開始ボタン ([Start])
リクエストデータのキャプチャを開始します。
- ③ 収集停止ボタン ([Stop])
リクエストデータのキャプチャを停止し、画面を閉じます。
- ④ ログ表示
データ収集のステータスを表示します。
- ⑤ 開くボタン ([Open])
過去のファジングデータ (ネットワーク設定) 選択画面 (本文に説明なし) を開きます。
- ⑥ OK ボタン ([OK])
設定を反映し、画面を閉じます。

■ ネットワーク設定画面

クライアントとサーバの IP アドレス、ポート番号を設定します。

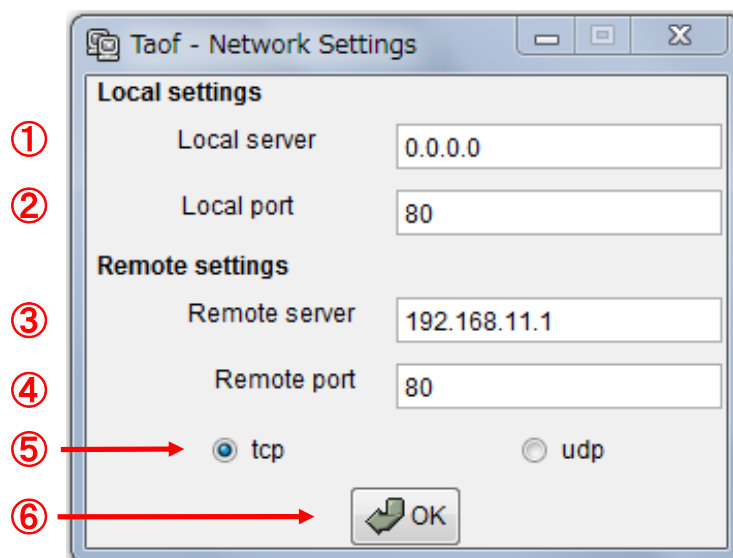


図 1-9 ネットワーク設定画面

- ① クライアントからのリクエストを待ち受ける IP アドレスの設定 ([Local server])
クライアントからのリクエストを待ち受ける IP アドレスを設定します (初期値は 0.0.0.0)。
- ② クライアントからのリクエストを待ち受けるポート番号の設定 ([Local port])
クライアントからのリクエストを待ち受けるポート番号を設定します。
- ③ サーバの IP アドレスの設定 ([Remote server])
サーバの IP アドレスを設定します。
- ④ サーバのポート番号の設定 ([Remote port])
サーバのポート番号を設定します。
- ⑤ プロトコル設定
④で指定したポート番号のプロトコルとして TCP、または UDP のどちらかを選択します。
- ⑥ OK ボタン ([OK])
設定を反映し、画面を閉じます。

■ ファジングポイント設定画面

リクエストからファジングポイントを設定します。⑧から⑩の設定次第で Taof が生成するファズデータが変わります。Taof が生成するファズデータについては「1.5 Taof が生成するファズデータ」を参照してください。

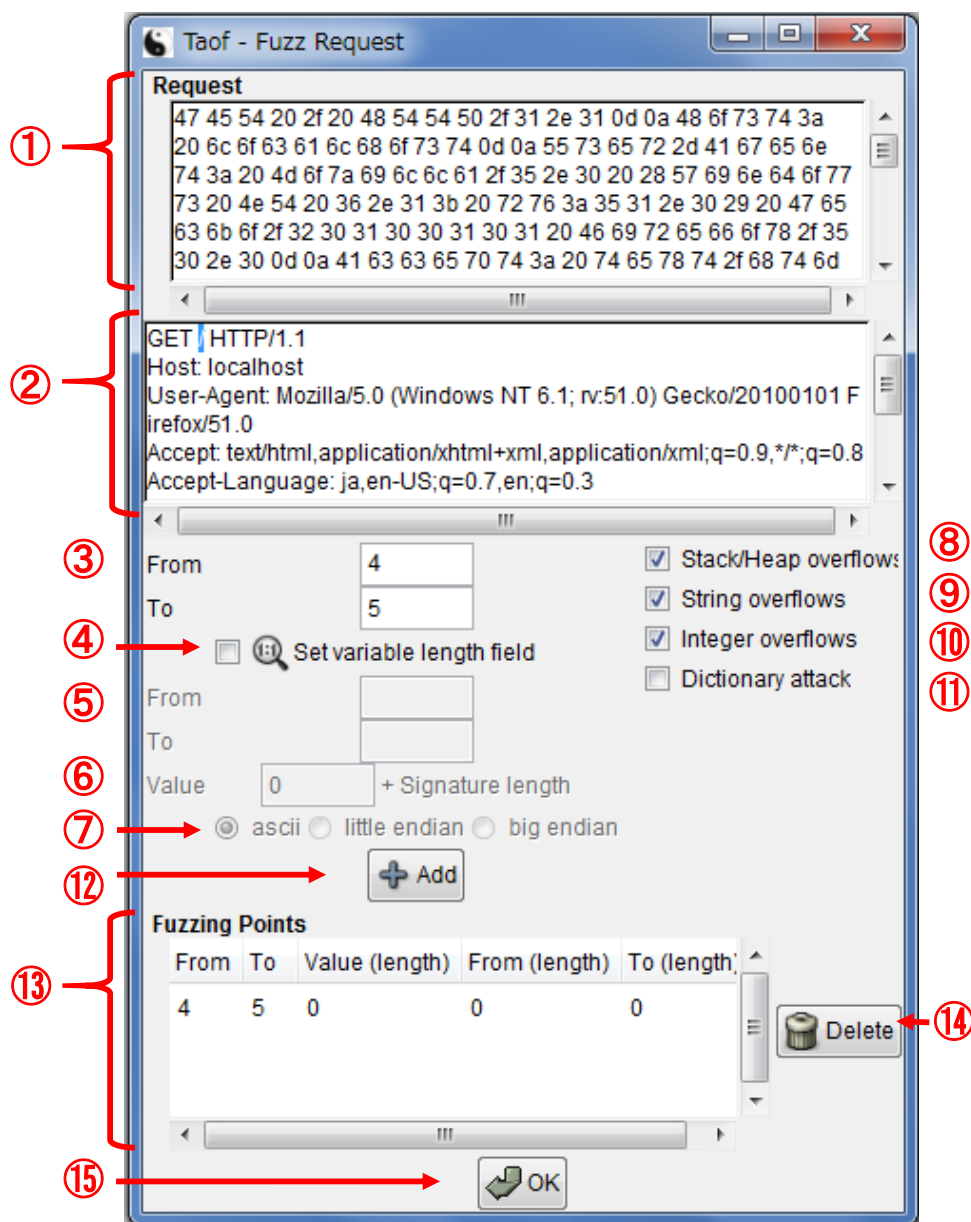


図 1-10 ファジングポイント設定画面

- ① リクエストバイナリ表示 ([Request])
リクエストデータを 16 進数で表示します。
- ② リクエストテキスト表示
リクエストデータを ASCII 文字列 (テキスト) で表示します。表示されている文字をマウスで反転選択するとファジングポイントの開始位置・終了位置が更新されます。
- ③ ファジングポイント開始位置・終了位置設定 ([From]、[To])
ファジングポイントの開始位置・終了位置を設定します。

- ④ 可変長文字列設定 ([Set variable length field]) **【IPA 未使用】**
- ⑤ 可変長文字列設定開始位置・終了位置設定 ([From (length)], [To (length)]) **【IPA 未使用】**
- ⑥ 可変長文字列長設定 ([Value (length)]) **【IPA 未使用】**
- ⑦ 文字列エンコード設定 ([ascii], [little endian], [big endian]) **【IPA 未使用】**
- ⑧ スタック・ヒープオーバーフロー設定 ([Stack/Heap overflows])
スタック・ヒープオーバーフローにつながる文字列に基づくファジングを有効にします。
- ⑨ 文字列オーバーフロー設定 ([String overflows])
書式文字列の問題につながる文字列に基づくファジングを有効にします。
- ⑩ 整数オーバーフロー設定 ([Integer overflows])
整数オーバーフローにつながる数値に基づくファジングを有効にします。
- ⑪ 辞書攻撃設定 ([Dictionary attack])
辞書ファイルに登録した文字列に基づくファジングを有効にします。
- ⑫ 追加ボタン ([Add])
ファジングポイントを追加します。
- ⑬ ファジングポイント一覧 ([Fuzzing Points])
設定したファジングポイントを確認できます。
- ⑭ 削除ボタン ([Delete])
ファジングポイントを削除します。
- ⑮ OK ボタン ([OK])
設定を反映し、画面を閉じます。

■ ファジング実行画面

サーバへファジングを実行します。

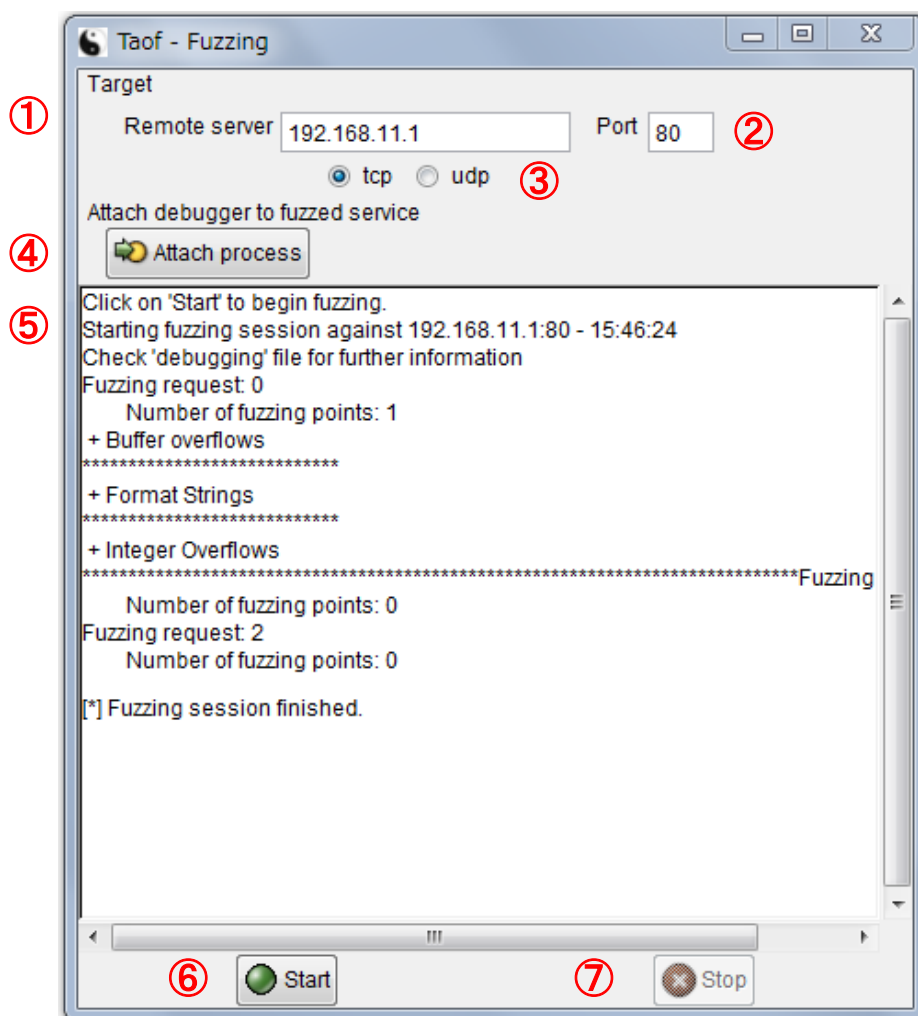


図 1-11 ファジング実行画面

- ① リモートサーバ設定 ([Remote server])
ファジング対象サーバを設定します。
- ② リモートサーバポート設定 ([Port])
ファジング対象サーバのポート番号を設定します。
- ③ プロトコル設定
TCP、または UDP を設定します。
- ④ プロセス監視設定 ([Attach process]) 【IPA 未使用】
- ⑤ ログ表示
ファジングの実施状況や結果を表示します。
- ⑥ 開始ボタン ([Start])
ファジングを開始します。
- ⑦ 停止ボタン ([Stop])
ファジングを停止します。

1.4. ファジングの実践

IPA の「脆弱性検出の普及活動」におけるブロードバンドルーターを対象にしたファジングを例に、Taof の使用方法とファジングの実行方法を説明します。ファジングの対象ページは、ブロードバンドルーターのウェブ管理インターフェイスのトップページです。ファジングの実行環境を図 1-12 に示します。

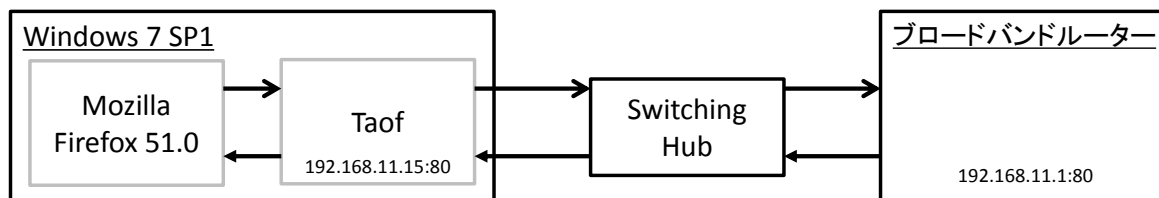


図 1-12 実行環境

(1) リクエストデータの収集

- ① Taof のメイン画面で[Data retrieval]を選択します。
- ② データ収集画面で[Network Settings]を選択します。
- ③ ネットワーク設定画面で[local server]を「0.0.0.0」（初期値¹）に、[local port]を「80」²に、[Remote server]を「192.168.11.1」に、[Remote port]を「80」に設定し、[OK]を選択します。
- ④ データ収集画面で[Start]を選択します。Windows 7 上で Windows Firewall を有効にしていると、このとき Windows Firewall により警告画面が表示される場合があります。この警告画面は「Taof に『0.0.0.0:80』での TCP 通信の待ち受けを許可してもよいか」ということを利用者に伝えています。Taof を使用するうえで必要となりますので、許可してください。
- ⑤ ブラウザを立ち上げ、「http://localhost/」にアクセスします。
- ⑥ ブラウザにトップページが表示されたら、Taof のデータ収集画面で[Stop]を選択します。
- ⑦ メイン画面の[Request List]に収集したリクエストが一覧表示されます。

¹ 「0.0.0.0」にした場合、Windows 7 に搭載されているネットワークインターフェイスすべてに指定した TCP ポート番号を割り当てます。

² Taof を使用する Windows 上で別のプログラムが 80/tcp を使用している場合、別のポート番号を指定してください。

- ⑧ 収集したリクエストの例を図 1-13 に示します。

```
GET / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*:q=0.8
Accept-Language: ja,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate, br
Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

図 1-13 GET リクエスト例

- ⑨ 収集したリクエストは Taof のインストール先に設定フォルダが生成され、自動的に保存されます。フォルダ名は「fuzzing_session-**DDD-HH.mm.ss**」で作成されます。(DDD : 1月1日からの経過日数/HH.mm.ss : 時分秒)

(2) ファジングポイントの設定

- ① メイン画面の[Request List]でファジングを実施するリクエスト GET リクエスト (例 : Request ID 1) を選択します。
- ② [Set fuzzing points]を選択します。
- ③ ファジングポイント設定画面の上から 2 段目、リクエストが表示されているテキストボックス内の一行目 4~5 文字目の「/ (スラッシュ)」を範囲指定します。
- ④ [From]が「4」に、[To]が「5」に設定されます。
- ⑤ [Stack/Heap overflows]、[String overflows]、[Integer overflows]にチェックをつけ、[Add] ボタンを選択します。
- ⑥ [OK]ボタンを選択します。
- ⑦ 設定例を図 1-14 に示します。

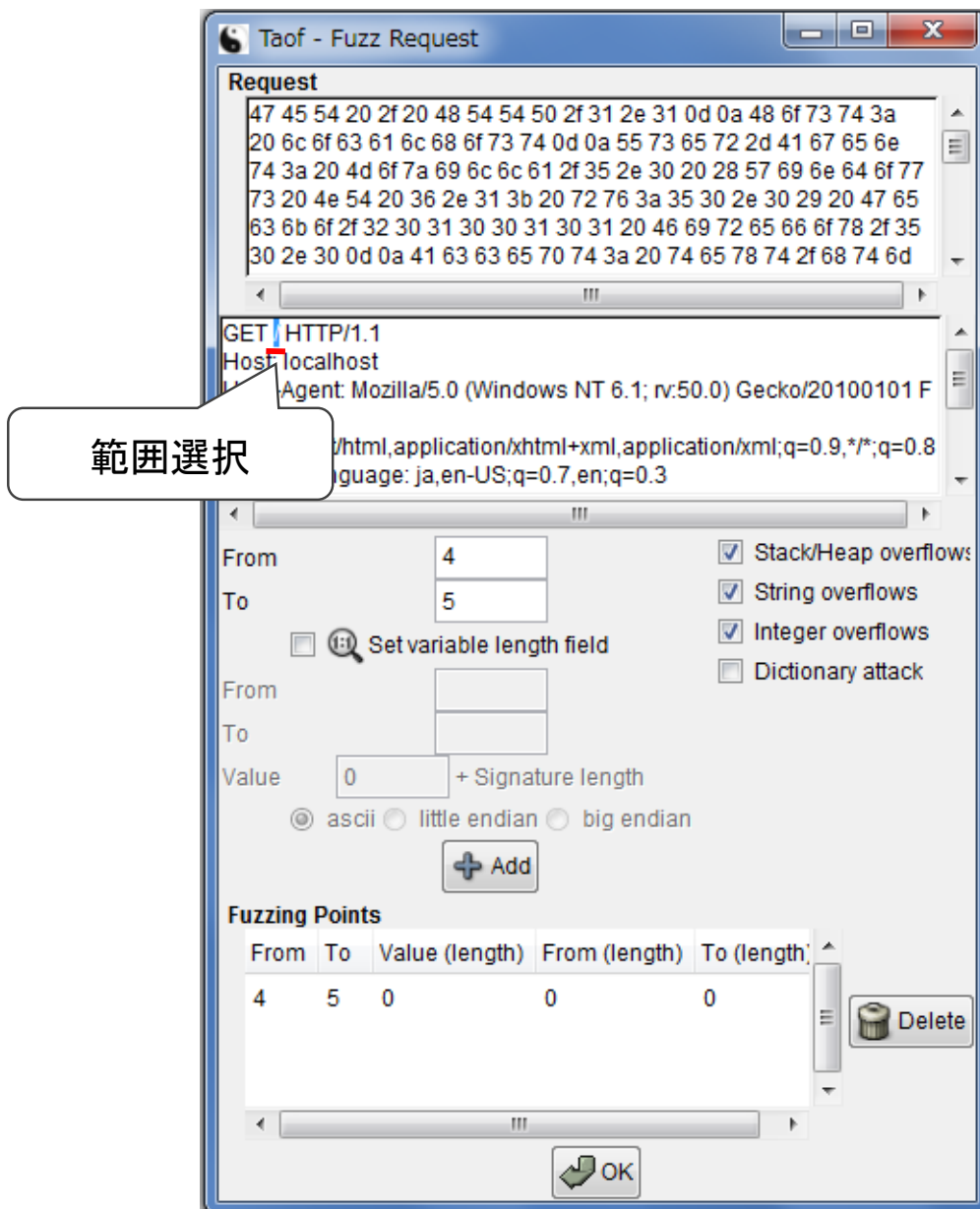


図 1-14 ファジングポイント設定例

- ⑧ 設定内容は設定フォルダ内に自動的に保存されます。過去の設定を再利用する場合はメイン画面のメニューから[Open]を選択し、ファイル選択画面で「fuzz.xml」ファイルを指定します。

(3) ファジングの実行

- ① メイン画面の[Fuzzing]ボタンを選択します。
- ② ファジング実行画面で[Remote Server]を「192.168.11.1」に、[Port]を「80」に設定します。
- ③ [Start]ボタンを選択すると、ファジングが実行されます。

④ 送信されるファジングデータの例を図 1-15に示します。

```
GET AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*:q=0.8
Accept-Language: ja,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

図 1-15 ファジングデータ例

⑤ ファジングの実行結果は設定フォルダ内の「debugging」ファイルに保存されます。

1.5. Taof が生成するファズデータ

Taof が生成するファズデータには、「スタック・ヒープオーバーフローにつながる文字列」、「書式文字列の問題につながる文字列」、「整数オーバーフローにつながる数値」、「辞書ファイルに登録した文字列」の 4 種類があります。

1.5.1. スタック・ヒープオーバーフローにつながる文字列

「スタック・ヒープオーバーフローにつながる文字列」はファジングポイントを文字列「A」に置き換えて送信するファズデータです。33 文字から最大 65536 文字までの文字列に置き換えて送信します。「スタック・ヒープオーバーフローにつながる文字列」のファズデータを表 1.5-1 に示します。

表 1.5-1 スタック・ヒープオーバーフローにつながる文字列

#	パターン
1	"A" * 33
2	"A" * 128
3	"A" * 240
4	"A" * 255
5	"A" * 256
6	"A" * 257
7	"A" * 420
8	"A" * 511
9	"A" * 512
10	"A" * 1023
11	"A" * 1024
12	"A" * 2047
13	"A" * 2048
14	"A" * 4096
15	"A" * 4097
16	"A" * 5000
17	"A" * 10000

#	パターン
18	"A" * 20000
19	"A" * 32762
20	"A" * 32763
21	"A" * 32764
22	"A" * 32765
23	"A" * 32766
24	"A" * 32767
25	"A" * 32768
26	"A" * 65534
27	"A" * 65535
28	"A" * 65536

1.5.2. 書式文字列の問題につながる文字列

「書式文字列の問題につながる文字列」はファジニングポイントを文字列のフォーマット指定子「%s」、「%x」、「%n」などに置き換えて送信するファズデータです。「書式文字列の問題につながる文字列」のファズデータを表 1.5-2 に示します。

表 1.5-2 書式文字列の問題につながる文字列

#	パターン
1	"%s" * 4
2	"%s" * 8
3	"%s" * 15
4	"%s" * 30
5	"%x" * 1024
6	"%n" * 1025
7	"%s" * 2048
8	"%s%n%x%d" * 5000
9	"%s" * 30000
10	"%s" * 40000
11	"%. 1024d"
12	"%. 2048d"
13	"%. 4096d"
14	"%. 8200d"
15	"%9999999999s"
16	"%9999999999d"
17	"%9999999999x"
18	"%9999999999n"
19	"%9999999999s" * 1000
20	"%9999999999d" * 1000
21	"%9999999999x" * 1000
22	"%9999999999n" * 1000

#	パターン
23	"%08x" * 100
24	"%20s" * 1000
25	"%20x" * 1000
26	"%20n" * 1000
27	"%20d" * 1000
28	"%#0123456x%08x%x%s%p%n%d%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x%#0123456x%x%s%p%n%d%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x"

1.5.3. 整数オーバーフローにつながる数値

「整数オーバーフローにつながる数値」はファジングポイントを数値に置き換えて送信するファズデータです。「-1」から最大「4294967295」まで置き換えて送信します。データとしての数値の表現は「ASCII 文字列 (例: 1 の場合 0x31)」、「リトルエンディアン形式の 32bit 整数型 (例: 1 の場合、0x01 00 00 00)」、「ビッグエンディアン形式の 32bit 整数型 (例: 1 の場合、0x00 00 00 01)」の 3 種類があります。

「整数オーバーフローにつながる数値」のファズデータを表 1.5-3 に示します。「整数オーバーフローにつながる数値」では 3 種類の数値の表現方法それぞれでこの数値を送信します。

表 1.5-3 整数オーバーフローにつながる数値

#	パターン
1	-1
2	0
3	1
4	127
5	128
6	255
7	256
8	32767
9	32768
10	65535
11	65536
12	2097151
13	2097152
14	4194304
15	8388608
16	16777216
17	33554432
18	67108864
19	134217728
20	268435456
21	536870912

#	パターン
22	1073741823
23	1073741824
24	2147483647
25	2147483648
26	4294967295

1.5.4. 辞書ファイルに登録した文字列

「辞書ファイルに登録した文字列」はファジングポイントを指定した辞書ファイルの内容に置き換えて送信するファズデータです。辞書ファイルの例を図 1-16 に示します。下記例ではファジングポイントが「pwd」、「pass」、「password」に変換されて送信されます。

```
pwd
pass
password
```

図 1-16 辞書ファイル例

2. ファジングツール「Peach」の使い方

本章ではファジングツール「Peach」の概要と使い方、「Peach」によるファジング手順、「Peach」が生成するファズデータを説明します。

2.1. 概要

「Peach」は、Peach Fuzzer 社が開発したソフトウェアで、複数の OS (Linux、Windows、Mac OS X) 上で動作し、ファイルを読み込むソフトウェアや TCP/IP などで通信するソフト、ウェブアプリケーションなど、幅広いソフトウェア製品に対してファジングを実践できます。

「Peach」は、有償版と無償版が提供されています。有償版は、「Professional edition」、「Enterprise edition」の 2 種類が提供されています。いずれも GUI 操作により、様々なプロトコルやソフトウェアに対するテストが可能です。無償版は、「Community edition」の 1 種類が提供されています。「Community edition」は、Michael Eddington により MIT ライセンスに基づいてフリーウェアとして公開されており、CUI で操作します。様々なプロトコルやファイルに対するテストを実施するには、同梱されているサンプル (ファイルファジングや HTTP ファジング等) を元に、利用者が設定ファイルを作成する必要があります。

本章では、主に、32bit 版の Windows で動作する無償版の Peach 3.1.124 の使い方を説明します。また、バージョン 2 との違い (主に設定ファイルの記載) についても説明します。

表 2.1-1 Peach Fuzzer Community edition の概要

項目	内容
IPA 使用バージョン	3.1.124
ライセンス	MIT License
ファジング手法	<ul style="list-style-type: none">● ファイルによるファジング● ネットワークを介したファジング● ウェブアプリケーションに対するファジング
URL	http://community.peachfuzzer.com/ (2017 年 2 月確認)
ダウンロード先	https://sourceforge.net/projects/peachfuzz/files/Peach/
特記事項	公式の最新公開バージョン ³ 3.1.124 (2017 年 2 月末日時点)

2.2. Peach によるファジングの流れ

Peach によるファジングでは、Pit と呼ばれる XML 形式の設定ファイルに、ファジング対象に読み込ませるデータの雛形（ファジング対象が扱う通常のデータ構造に沿ったもの）と、その変換箇所を記述します。Peach 実行時に、この設定ファイルを読み込ませることで、Peach に搭載された「Mutator」と呼ばれるファジングエンジンが、変更箇所として指定されたデータを様々な値に変換し、ファジング対象に送信することでファジングを行います。

インストールフォルダ内の samples フォルダに、各種ファジング用のサンプル設定ファイルがあります。これらのサンプル設定ファイルを基に用途にあった設定ファイルを作成します。設定ファイルについては、2.4 節で説明します。

Peach によるファジングは具体的には「設定ファイルの準備」、「ファジングの実行」の 2 つの工程に分けて実施します。

(1) 設定ファイルの準備

Peach によるファジングの基となる設定ファイルを準備します。Peach によるファジングを実行する前に、準備した設定ファイルに構文間違い等が無いかを、後述するコマンドオプションによって確認します。

(2) ファジングの実行

設定ファイルを指定して Peach を実行します。Peach の実行が終了したら、Peach の実行結果やログファイル⁴を確認し、問題を起こしたファズデータを特定します。

2.3. 使い方

2.3.1. インストールとアンインストール

(1) インストール

Peach のインストールは、Microsoft.NET v4 Runtime、および Debugging Tools for Windows をインストールした上で、Peach の ZIP ファイルを解凍します。

(2) アンインストール

Peach はレジストリを使用しないので、フォルダごと削除することでアンインストールできます。

⁴ ログファイルは、設定ファイルで指定したログフォルダに出力されます。

2.3.2. Peach の使用

インストールフォルダ内の「peach.exe」をコマンドプロンプト (cmd.exe) から起動します。「peach.exe」を実行すると、Peach のマニュアルが出力されます。

(1) Peach のオプション

Peach のマニュアルの一部を以下に抜粋します。なおこの出力結果は意味が変わらない範囲で整形しています。

```
Syntax:
peach -a
peach -c peach_xml_file [run_name]
peach -g
peach [--skipto #] peach_xml_file [run_name]
peach -p 10,2 [--skipto #] peach_xml_file [run_name]
peach --range 100,200 peach_xml_file [run_name]
peach -t peach_xml_file

-1                Perform a single iteration 【IPA 未使用】
-a, --agent       Launch Peach Agent 【IPA 未使用】
-c, --count       Count test cases 【IPA 未使用】
-t, --test xml_file  Test parse a Peach XML file
-p, --parallel M, N  Parallel fuzzing. Total of M machines, this is machine N. 【IPA 未使用】
--debug           Enable debug messages. Usefull when debugging
                  your Peach XML file. Warning: Messages are very cryptic sometimes.
                  【IPA 未使用】
--skipto N        Skip to a specific test #. This replaced -r for restarting a Peach run.
                  【IPA 未使用】
--range N, M      Provide a range of test #'s to be run.
--seed N          Seed to use for Random strategies
--analyzer=CLASS  Use analyzer via command line 【IPA 未使用】
--parser=CLASS    Use a custom parser analyzer (replaces default XML parser) 【IPA 未使用】
```

上記 Peach のマニュアルに記載されているように Peach には多くのオプションがあります。本項では、これらオプションの利用例として、問題の原因となるファズデータを特定する際に用いられる「**seed** オプション」と「**range** オプション」を解説します。

■ seed オプション

Peach のファズデータは、実行時に生成される乱数値 (seed 値) に基づいて生成されます。よって、同じ設定ファイルを用いてファジングを実行した場合でも、seed 値が異なっていれば生成されるファズデータのパターンは異なるものになるため、テストの無作為性を担保できます。

「seed オプション」は、この seed 値を固定値として設定することで、実行時毎にファジングのパターンを変化させることなく、同じパターンでファジングを行うためのオプションです。

ファジングの実施により問題が発生することを確認した場合などは、どのファズデータによって問題が発生したのかを特定するために、問題が発生した際に用いられたファズデータと同じファズデータを再現して絞り込む必要があります。そこで、「seed オプション」により seed 値を固定値としておくと、毎回同じパターンでファジングを実行できるため、問題が再現するファズデータの調査がしやすくなります。

例) 「seed オプション」に 12345 を指定して Peach を実行する場合

```
$> peach --seed 12345 sample.xml
```

■ range オプション

「range オプション」を使用すると、テストするパターンの範囲を指定してファジングを実行できます。途中でファジングを中断した場合や、ファジング対象機器が応答しないことで Peach が停止した場合に、「range オプション」を使うと任意の番号のファズデータからファジングを実行できます。そのため、上記の「seed オプション」と併用することで、問題を起こしたファズデータの絞り込み等に用いられます。

「range オプション」は、--range [開始番号],[終了番号]となっており[開始番号]から[終了番号]までがファジング範囲となります。

例) テスト番号 10 から 100 までファジングする場合

```
$> peach --seed 12345 --range 10,100 sample.xml
```

(2) Peach によるファジングの実行

Peach でファジングを実行する場合、「peach.exe」の引数に設定ファイルを指定して実行します。この設定ファイルについては、次項で説明します。

例) 設定ファイル「sample.xml」を指定して Peach を実行する場合

```
$> peach sample.xml
```

2.4. 設定ファイルについて

Peach によるファジングでは、設定ファイルを適切に記述することが重要です。設定ファイルは、複数の要素⁵から構成されますが、すべてを編集しなくてもファジングを実行できます。Peach に付属しているサンプル設定ファイル⁶を例にとると、サンプル設定ファイルは、Peach 2.x では 6 つの要素、Peach 3.x では 4 つの要素で構成されています (図 2-1)。これらの要素については表 2.4-1 を参照してください。

Peach 3.x では 2 つの要素定義が廃止されて 4 つの要素定義で構成されます。旧バージョンの Peach 2.x で作成した設定ファイルは、Peach 3.x でそのまま使用できないため、修正が必要です。

3系	2系 (参考)
XMLヘッダ	XMLヘッダ
	Include要素
DataModel要素	DataModel要素
StateModel要素	StateModel要素
Test要素	Test要素
	Run要素

図 2-1 設定ファイルの主な要素

⁵ 設定ファイルの各要素については <http://community.peachfuzzer.com/v3/PeachPit.html> を参照してください。

⁶ Peach のインストールフォルダ内の samples フォルダに保存されています。

表 2.4-1 設定ファイルの要素に関する説明

要素の説明	
3系	2系 (参考)
<p>【XML ヘッダ】 XML のヘッダ</p>	<p>【XML ヘッダ】 XML のヘッダ</p>
<p>【Include 要素】 記載不要</p>	<p>【Include 要素】 Peach の動作設定ファイルを読み込みます。サンプル設定ファイルの場合、ファズデータを生成する Mutator 要素 (2.6 節を参照) を default.xml から読み込みます。 詳細情報： http://community.peachfuzzer.com/v2/Include.html</p>
<p>【DataModel 要素】 ファズデータの元となるデータ構造 (DataModel 要素) と実際の値 (Data 要素) を定義します。 詳細情報： http://community.peachfuzzer.com/v3/DataModel.html</p>	<p>【DataModel 要素】 ファズデータの元となるデータ構造 (DataModel 要素) と実際の値 (Data 要素) を定義します。 詳細情報： http://community.peachfuzzer.com/v2/DataModel.html</p>
<p>【StateModel 要素】 ファズデータの動き (入出力など) を定義します。 詳細情報： http://community.peachfuzzer.com/v3/StateModel.html</p>	<p>【StateModel 要素】 ファズデータの動き (入出力など) を定義します。 詳細情報： http://community.peachfuzzer.com/v2/StateModel.html</p>
<p>【Test 要素】 ファジングにおけるテスト対象の指定や、ログの出力先、変換方法などを定義します。 詳細情報： http://community.peachfuzzer.com/v3/TestConfig.html</p>	<p>【Test 要素】 ファジングにおけるテスト対象の指定や、ログの出力先、変換方法などを定義します。 詳細情報： http://community.peachfuzzer.com/v2/Test.html</p>
<p>【Run 要素】 記載不要</p>	<p>【Run 要素】 実践するファジングを定義します。Test 要素で事前に定義したファジングを指定することで、指定したファジングを実践できるようになります。 詳細情報：なし</p>

IPA で Peach によるファジングを実践したケースでは、これら 4 つの要素のうち、主に「DataModel 要素」、「StateModel 要素」、「Test 要素」を編集しています。参考までに HTTP ファジングに使用できる設定ファイルにおける「DataModel 要素」、「StateModel 要素」、「Test 要素」を 2.4.1 項で紹介します。

2.4.1. HTTP ファジングに使用できる設定ファイルの例

この項では、図 2-2 のようなファジングを実践するための設定ファイルの「DataModel 要素」、「Data 要素」、「StateModel 要素」、「Test 要素」を紹介합니다。この項で取り上げる設定ファイルは、Peach 2.x に付属しているサンプル設定ファイル「HTTP.xml」を、Peach 3.x 向けに編集したものです。

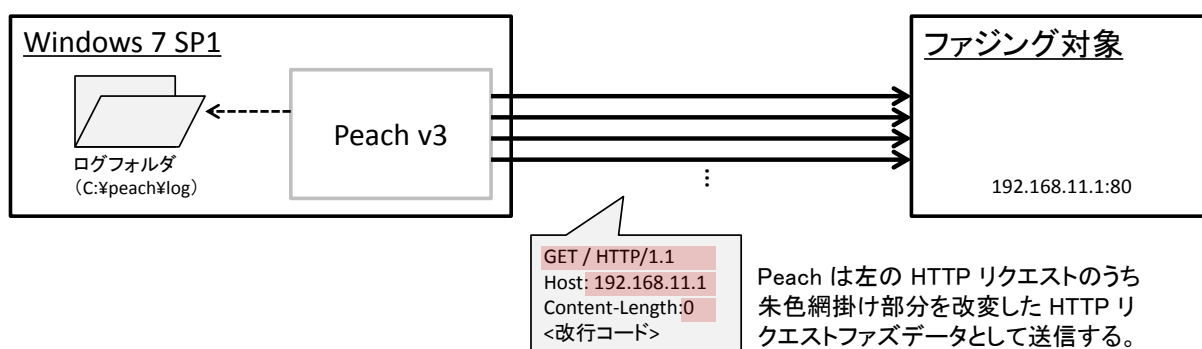


図 2-2 Peach による HTTP ファジング

■ DataModel 要素

DataModel 要素では、ファズデータの元となる HTTP リクエストのデータ構造を定義し、その各ヘッダやデータなどの各パラメータに対して初期値を指定し、そのパラメータを変換対象とするか否かを設定します。

表 2.4-2 DataModel 要素、Data 要素における該当部分の説明

該当部分	説明
①	DataModel 要素として HTTP ヘッダを定義します。この定義は「<Header>: <Value><改行コード>」というものになります。
②	DataModel 要素として HTTP リクエストを定義します。この DataModel 要素の中に、HTTP リクエストラインとして「<Method> <RequestURI> <HttpVersion><改行コード>」などを定義します。
③	HTTP リクエストの DataModel 要素に、Host ヘッダとして「Host: <value><改行コード>」を定義します。
④	Content-Length ヘッダとして「Content-Length: <Body のサイズ>」、Body 部を定義します。

```

<!-- ①ここから -->
  <DataModel name="HeaderField" >
    <String name="Header" />
    <String value=": " />
    <String name="Value" />
    <String value="¥r¥n" />
  </DataModel>
<!-- ①ここまで -->
<!-- ②ここから -->
  <DataModel name="HttpRequest" >
    <Block name="RequestLine" >
      <String name="Method" value="GET" />
      <String value=" " />
      <String name="RequestUri" value="/" />
      <String value=" "/>
      <String name="HttpVersion" value="HTTP/1.1" />
      <String value="¥r¥n" />
    </Block>
<!-- ②ここまで -->
<!-- ③ここから -->
  <Block name="HeaderHost" ref="HeaderField" >
    <String name="Header" value="Host" />
    <String value=": " />
    <String name="Value" value="192.168.11.1" />
    <String value="¥r¥n" />
  </Block>
<!-- ③ここまで -->
<!-- ④ここから -->
  <Block name="ContentLength" ref="HeaderField" >
    <String name="Header" value="Content-Length" />
    <String name="Value" >
      <Relation type="size" of="Content" />
    </String>
  </Block>
<!-- ④ここまで -->
  <String value="¥r¥n" />
  <Blob name="Content" />
</DataModel>

```

■ StateModel 要素

StateModel 要素では、ファジング対象へのデータ入出力を定義します。ここでは、前項で定義した HTTP リクエストをファジング対象に対して送信するよう設定します。

```
<StateModel name="HttpRequestStateModel" initialState="HttpRequestState">
  <State name="HttpRequestState">
    <Action type="output">
      <DataModel ref="HttpRequest" />
    </Action>
  </State>
</StateModel>
```

■ Test 要素

Test 要素では、ファジング対象の指定やログの出力先などといった、テストの設定を行います。ここでは、StateModel クラスにより、前述の StateModel 要素内で定義した StateModel name を設定します。さらに、Publisher クラスにより、ファジング対象の IP アドレスとポート番号に 192.168.11.1:80 を指定しています。

また、Logger クラスにより、ファジングのログファイルの出力先フォルダなどを定義します。ここでは、前項で定義したファジングを指定し、ログ出力先フォルダとして「C:\peach\logs」を指定します。

※Peach 2.x では、Run 要素内にログの出力先を記述していました。

```
<Test name="Default">
  <StateModel ref="HttpRequestStateModel" />
  <Publisher class="TcpClient">
    <Param name="Port" value="80" />
    <Param name="Host" value="192.168.11.1" />
  </Publisher>
  <Logger class="Filesystem">
    <Param name="Path" value="logs"/>
  </Logger>
</Test>
```


2.5. ファジングの実践

ブロードバンドルーターを対象にしたファジングを例に、Peach の使用方法とファジングの実行方法を説明します。ファジングの対象ページは、ブロードバンドルーターのウェブ管理インターフェイスのトップページです。ファジングの実行環境を図 2-3 に示します。

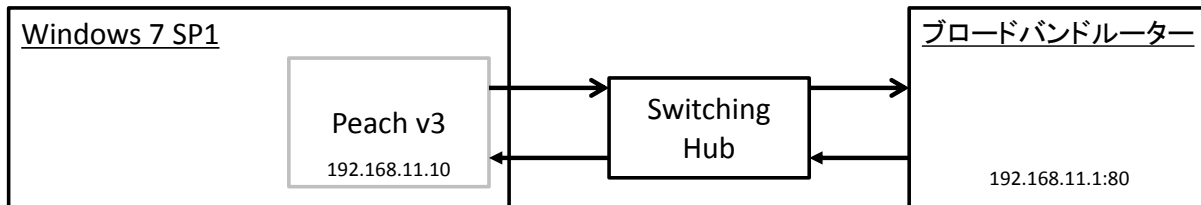


図 2-3 実行環境

(1) 設定ファイルの準備

Peach でファジングする場合、ファジングの用途に合わせた設定ファイルを準備します。ここでは 2.4.1 項で紹介した設定ファイル（ここでは、ファイル名「HTTP.xml」）を使用します。

設定ファイルを使用してファジングを開始する前に、設定ファイルに構文間違い等がないかを確認します。下記のように、`-t` オプションの後に、対象の設定ファイルを指定することで、当該ファイルの構文チェックを行うことができます。

```
$> peach -t HTTP.xml
```

設定ファイルに構文間違いなどが無い場合、図 2-4 のようなメッセージが出力されます。

```
C:\Windows\system32\cmd.exe
C:\peach>peach -t HTTP.xml
[[ Peach v3.1.124.0
[[ Copyright (c) Michael Eddington
[*] Validating file [HTTP.xml]... No Errors Found.
C:\peach>
```

図 2-4 設定ファイルに構文間違いなどが無い場合の出力結果例

設定ファイルに構文間違いなどがある場合、図 2-5 のようなメッセージが出力されます。

```
C:\Windows\system32\cmd.exe
C:\peach>peach -t HTTP.xml
[[ Peach v3.1.124.0
[[ Copyright (c) Michael Eddington
[*] Validating file [HTTP.xml]... Error, Pit file "C:\peach\HTTP.xml" failed to validate:
Line: 65, Position: 11 - 'name' 属性が宣言されていません。
Line: 65, Position: 6 - 必要な属性 'name' が見つかりません。
C:\peach>
```

図 2-5 設定ファイルに構文間違いなどがある場合の出力結果例

(2) ファジングの実行

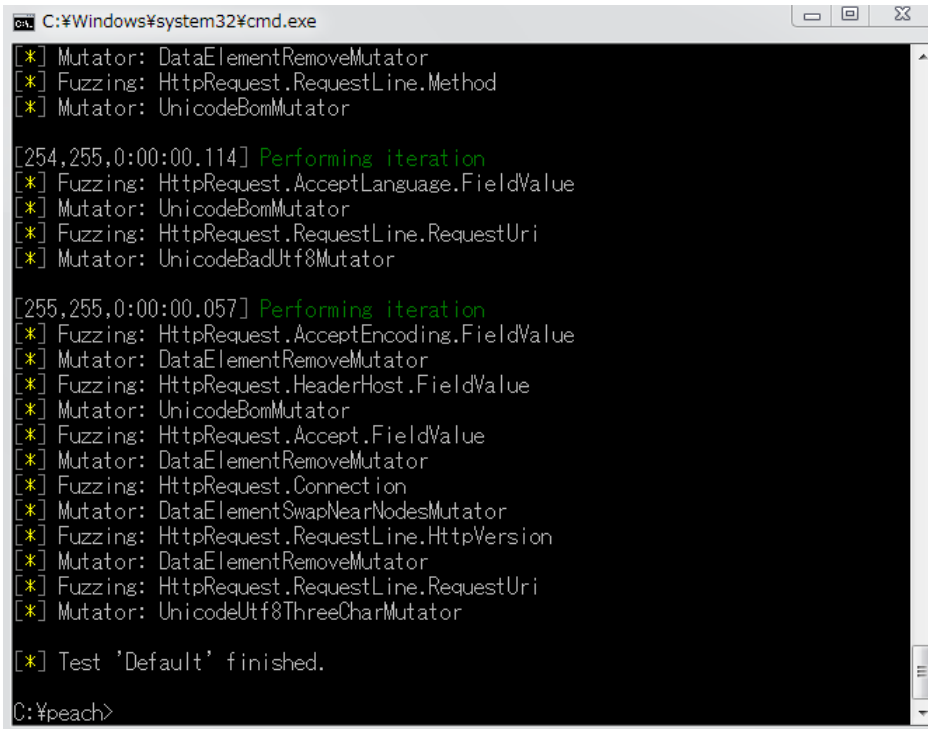
設定ファイルを指定して、ファジングを実行します。

```
$> peach HTTP.xml
```

Peach によるファジングを実行していると、コマンドプロンプトには実行しているファジング番号などが出力されます。この出力結果はログファイルに記録されないため、必要に応じてファイルに出力するとよいでしょう。以下はコマンドプロンプトの出力結果を `http_fuzz_log.txt` に出力する例です。

```
$> peach HTTP.xml > http_fuzz_log.txt
```

ブロードバンドルーターが異常終了などせずに Peach によるファジングが完了した場合、コマンドプロンプトの出力結果は図 2-6 のようになります。



```
C:\Windows\system32\cmd.exe
[*] Mutator: DataElementRemoveMutator
[*] Fuzzing: HttpRequest.RequestLine.Method
[*] Mutator: UnicodeBomMutator

[254,255,0:00:00.114] Performing iteration
[*] Fuzzing: HttpRequest.AcceptLanguage.FieldValue
[*] Mutator: UnicodeBomMutator
[*] Fuzzing: HttpRequest.RequestLine.RequestUri
[*] Mutator: UnicodeBadUtf8Mutator

[255,255,0:00:00.057] Performing iteration
[*] Fuzzing: HttpRequest.AcceptEncoding.FieldValue
[*] Mutator: DataElementRemoveMutator
[*] Fuzzing: HttpRequest.HeaderHost.FieldValue
[*] Mutator: UnicodeBomMutator
[*] Fuzzing: HttpRequest.Accept.FieldValue
[*] Mutator: DataElementRemoveMutator
[*] Fuzzing: HttpRequest.Connection
[*] Mutator: DataElementSwapNearNodesMutator
[*] Fuzzing: HttpRequest.RequestLine.HttpVersion
[*] Mutator: DataElementRemoveMutator
[*] Fuzzing: HttpRequest.RequestLine.RequestUri
[*] Mutator: UnicodeUtf8ThreeCharMutator

[*] Test 'Default' finished.
C:\peach>
```

図 2-6 Peach によるファジングが正常に終了した場合

一方、ブロードバンドルーターが異常終了などした場合、コマンドプロンプトの出力結果は図 2-7 のようになります。図 2-7 では、ブロードバンドルーターが起動していない状態でファジングを実行したため操作がタイムアウトした例を示しています。ファジングの実行中に、ファジング対象が異常終了した場合なども同様の表示が出力されます。

このように、ファジング対象が異常終了したことで Peach が停止した場合や、ファズデータの流れが極端に遅くなる場合は、ファジングによってファジング対象になんらかの異常が発生していることを検知できます。

```
cmd C:\Windows\system32\cmd.exe
C:\peach>peach HTTP.xml --range 1,255

[[ Peach v3.1.124.0
[[ Copyright (c) Michael Eddington

[*] Test 'Default' starting with random seed 35287.

[R1,-,-] Performing iteration

[*] Test 'Default' finished.
操作がタイムアウトしました。

C:\peach>
```

図 2-7 Peach によるファジングが正常に終了しなかった場合

Peach によるファジングの実行結果は、設定ファイルで指定したログ出力先フォルダに出力されます。出力ファイルは、<ログ出力先フォルダ>\[設定 File 名]_日付フォルダ\status.txt となります。また、Peach 3.x では異常終了した際、問題を起こしたファズデータが<ログ出力先フォルダ>\[設定 File 名]_日付フォルダ\Faults 以下のフォルダに格納されます。図 2-8 は、samples フォルダ内の FileFuzzerGui.xml を編集したものを実行した画面です。この例では、3 番目のファズデータに関するログが上記のフォルダ内に作成されます。

```
cmd 管理者: コマンド プロンプト
C:\peach>peach FileFuzzerGui.xml --seed 45496

[[ Peach v3.1.124.0
[[ Copyright (c) Michael Eddington

[*] Test 'Default' starting with random seed 45496.

[R1,-,-] Performing iteration

[1,-,-] Performing iteration
[*] Fuzzing: FileData.DataElement_0
[*] Mutator: UnicodeBomMutator

[2,-,-] Performing iteration
[*] Fuzzing: FileData.DataElement_0
[*] Mutator: StringMutator

[3,-,-] Performing iteration
[*] Fuzzing: FileData.DataElement_0
[*] Mutator: UnicodeBadUtf8Mutator

-- Caught fault at iteration 3, trying to reproduce --

[3,-,-] Performing iteration
[*] Fuzzing: FileData.DataElement_0
[*] Mutator: UnicodeBadUtf8Mutator

-- Reproduced fault at iteration 3 --

[4,-,-] Performing iteration
```

図 2-8 問題のあるファズデータがログに記録された際の画面

2.6. Peach が生成するファズデータ

Peach は、設定ファイルで定義したデータモデルのうち変換対象と定義したものを、「Mutator」と呼ばれるファジングエンジンで変換して、ファズデータとして使います。本節では、ファズデータにおける変換対象の定義方法と定義したデータの変換手順を説明して、ファズデータを例示します。

■ 変換対象

設定ファイルで定義したデータを変換対象とするかは、`mutable` 属性⁷の値で決まります。この `mutable` 属性の値には、「true」と「false」の 2 通りがあります。設定ファイルで定義したデータの `mutable` 属性の値が「true」であった場合、そのデータは変換対象となります。変換対象としない値がある場合、`mutable` 属性の値に「false」を設定してください（図 2-11 参照）。設定ファイルで定義したデータで `mutable` 属性を定義していなかった場合、`mutable` 属性の値が「true」と同じ意味となります。

■ 変換形式

変換手順は Python のクラス単位で書かれており、ソースコードが公開されています。表 2.6-1 に変換手順の一例を挙げます。詳細については Peach の公式サイト⁸をご確認ください。

表 2.6-1 Peach における変換手順の例

変換手順 (Mutator 名)	説明 (ソースコードのコメントに基づく)
<code>blob.BlobMutator</code>	元となるデータを 1 バイト単位で変換する。
<code>blob.BitFlipperMutator</code>	元となるデータをビット単位で反転する (0 であれば 1)。初期設定では元データの 20%を反転する。
<code>blob.DWORDSliderMutator</code>	元となるデータを 4 バイトごとにそれぞれの中身をずらす。
<code>string.StringMutator</code>	元となるデータを文字列単位で変換する。
<code>string.StringCaseMutator</code>	元となるデータの文字列の大文字、小文字を反転する。
<code>string.UnicodeBomMutator</code>	元となるデータの文字列に、BOM(Byte Order Marker)を含む文字列を挿入する。
<code>string.UnicodeBadUtf8Mutator</code>	文字エンコーディング「UTF-8」における不正な文字列を生成する。
<code>string.UnicodeUtf8ThreeCharMutator</code>	文字エンコーディング「UTF-8」により 3 バイトで構成される文字コードで構成される長い文字列を生成する。
<code>datatree.DataElementDuplicateMutator</code>	元となるデータの設定構成要素を 2 回から 50 回複数回繰り返す。
<code>size.SizedNumericalEdgeCasesMutator</code>	元となるデータのデータサイズの境界にあるものを変換する。
<code>size.SizedVarianceMutator</code>	元となるデータのデータサイズを変換する。

⁷ <http://community.peachfuzzer.com/v3/mutable.html>

⁸ <http://community.peachfuzzer.com/v3/Mutators.html>

■ Peach が生成するファズデータ例

Peach の変換手順「DataElementDuplicateMutator」で図 2-9 の HTTP リクエストを変換したファズデータを図 2-12 に例示します。この例では、図 2-9 の HTTP リクエストのデータのうち、リクエストライン「GET / HTTP/1.1」と Host ヘッダの値「192.168.11.1」のみ変換対象とします（図 2-10）。

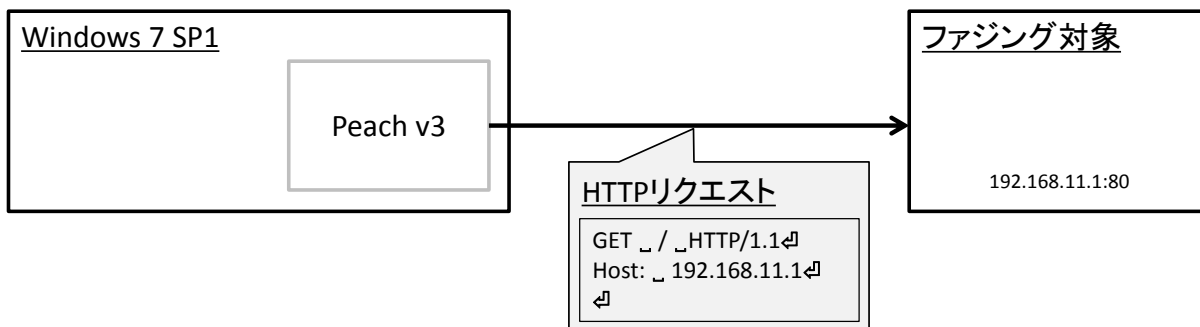


図 2-9 ファズデータの基となる HTTP リクエスト

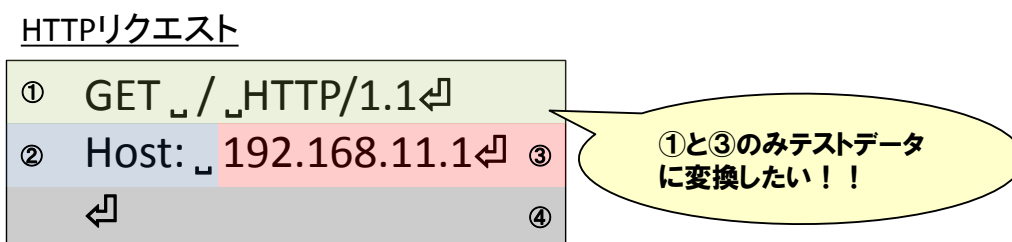


図 2-10 HTTP リクエストにおける変換対象

図 2-10 の要件を反映した設定ファイルは、図 2-11 の通りです。変換対象としない②と④に該当する String 要素には、mutable 属性を定義して値に「false」を設定します（図 2-11 の赤枠部分）。

Pitファイル

```
<DataModel name="HttpRequest">
  <String name="Method" value="GET / HTTP/1.1¥r¥n" /> ①
  <Block name="HeaderHost" ref="HeaderField" >
    <String name="Header" value="Host: "mutable="false" /> ②
    <String name="Value" value="192.168.11.1¥r¥n" /> ③
  </Block>
  (中略)
  <String value="¥r¥n"mutable="false" /> ④
</DataModel>
```

図 2-11 HTTP リクエスト（図 2-10）の変換対象を反映した設定ファイル

図 2-11 の設定ファイルを指定してファジングを実行すると、変換手順「DataElementDuplicateMutator」によるファズデータの一つは図 2-12 のようになります。元となるデータの「192.168.11.1」の部分、「192.168.11.1」が、「192.168.11.1」に変換されたことが分かります。

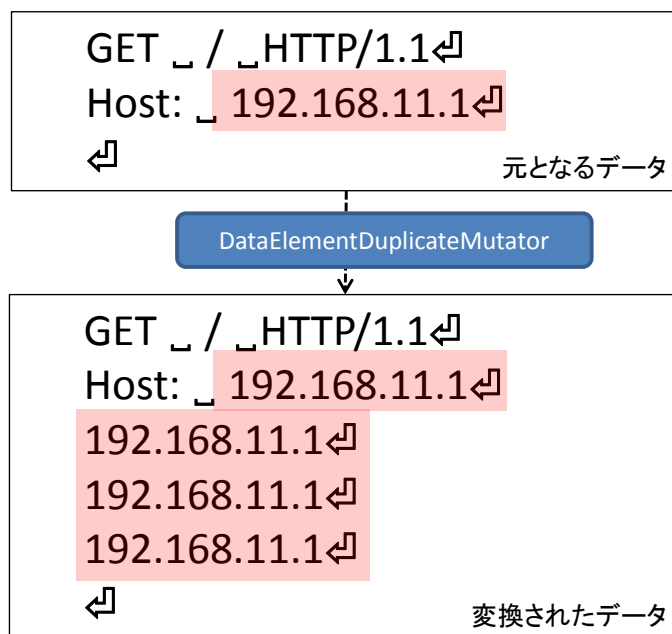


図 2-12 「DataElementDuplicateMutator」によるファズデータ例

3. ファジング結果の再現手順

本章ではファジング結果（パケットキャプチャファイル）からファジングツールを使わずにファジングを再現する手順を説明しています。この章で説明する手順では、**ネットワークを介したファジングの結果だけ再現できること**に留意してください。

3.1. 概要

本章は、ファジングの結果を記録したパケットキャプチャファイル⁹から記録されているファジング結果をツール¹⁰で再現する手順をまとめたものです。

ネットワークを介したファジングで脆弱性を検出しそれを開発部門に報告する場合、開発部門が脆弱性箇所を特定するためにその脆弱性による事象の再現手順を伝えることが重要です。再現手順には「事象を再現する実証コードを作成する」などの方法もありますが、ネットワークを介したファジングの場合「パケットキャプチャファイルをツールで再現する」方法も選択できる場合があります。本章では「パケットキャプチャファイルをツールで再現する」方法を紹介します。

3.2. ファジング結果の再現環境

「パケットキャプチャファイルをツールで再現する」場合、ツールを導入した再現環境を準備する必要があります。再現環境を準備する場合、IPA が実践した方法には 2 通りの方法があります。

- ① 「ツールをインストールした Linux 環境を作る」
- ② 「ツールがインストールされている CD 起動の Linux 環境を使う」

②の方が①よりも再現環境の準備にかかる手間がかかりませんが、(IPA が確認した限りでは)必要なツール (3.2.1 節を参照) をすべて同梱している CD 起動の Linux はありませんでした。そのため本書では①の再現環境を準備します。再現環境の OS には、「Ubuntu Desktop」を採用しました。

3.2.1. 再現環境に導入するツール

(1) **Linux**

- Ubuntu Desktop

<https://www.ubuntulinux.jp/>

本書ではミラーサイトなどから「Ubuntu 11.04 Desktop (ubuntu-11.04-desktop-i386.iso)」をダウンロードして使用します。

⁹ PCAP 形式ファイル (.pcap) を想定しています。また「ダンプファイル」などと呼ばれる場合もあります。

¹⁰ 本章で「ツール」といった場合、ファジングツールを含みません。

(2) 仮想環境実行ツール

- VMware Player

https://my.vmware.com/jp/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

(3) パケットキャプチャツール

- Wireshark

<https://www.wireshark.org/>

(4) パケットキャプチャ再現および加工ツール

- tcpreplay

PCAP 形式ファイルからネットワークパケットを再現するツール

<http://tcpreplay.synfin.net/>

- tcpreplay-edit

IP アドレス、MAC アドレスなどを置き換えて、PCAP 形式ファイルのネットワークパケットを再現するツール

<http://tcpreplay.synfin.net/>

- tcprewrite

PCAP 形式ファイルに保存しているネットワークトラフィックの IP アドレス、MAC アドレスなどを書き換えることができるツール

<http://tcpreplay.synfin.net/>

- wireplay

PCAP 形式ファイルに保存したネットワークトラフィックのうち、TCP などのように一連の流れがあるトラフィックを再現するツール

<http://code.google.com/p/wireplay/>

3.2.2. 再現環境の作成

本節ではファジング結果の再現環境の構築手順を説明します。この構築手順は、「Ubuntu をインストールした仮想マシンを作成する」と「ツールをインストールする」の 2 つの手順に分かれます。この手順を順番に説明します。

■ Ubuntu をインストールした仮想マシンを作成する。

(1) 仮想マシンの作成

- ① VMware Player を起動し、[ファイル(F)]→[新規仮想マシンの作成(C)]を選択します。
- ② 仮想マシン作成ウィザードへようこそ画面で、[後で OS をインストール(S)]→[次へ(N)]を選択します。
- ③ ゲスト OS の選択画面で[Linux(L)]→[Ubuntu]→[次へ(N)]を選択します。
- ④ 仮想マシン名、格納場所を設定し[次へ(N)]を選択します。この説明では仮想マシン名を

「FuzzUbuntu」、格納場所を「D:\FuzzUbuntu」とします。

- ⑤ ディスク容量の指定画面で[次へ(N)]を選択します。
- ⑥ 仮想マシンを作成する準備完了画面で[完了]を選択します。

(2) Ubuntu のインストール

- ① VMware Player 上で作成した仮想マシン「FuzzUbuntu」を選択し、[仮想マシン設定の編集(D)]を選択します。
- ② [CD/DVD(IDE)]→[接続]→[ISO イメージファイルを使用する]でダウンロードした「ubuntu-ja-11.04-desktop-i386.iso」を設定し、[OK]を選択します。
- ③ 仮想マシンの再生を選択し、Ubuntu をインストールします。インストール時の設定は初期設定を選択し進めてください。途中で作成したユーザとパスワードは忘れないように注意してください。この説明では途中で作成したユーザを「fuzzer」とします。
- ④ インストール完了後、仮想マシンを再起動します。

(3) Ubuntu のソフトウェアアップデート

- ① 仮想マシンを起動し、作成したユーザ「fuzzer」でログインします。
- ② [アプリケーション]→[アクセサリ]→[端末]（これより以下の手順では端末と省略）を選択します。
- ③ 端末で下記のコマンドを実行します。

```
$ sudo apt-get update
```

- ④ アップデートマネージャが起動するので、[アップデートをインストール(I)]を選択します。
- ⑤ 完了後、仮想マシンを再起動します。

(4) VMware Tools のインストール

- ① 仮想マシンを起動し、設定したユーザ（fuzzer で）ログインします。
- ② VMware Player のメニューで[仮想マシン(V)]→[VMware Tools のインストール(I)]を選択します。
- ③ CD-ROM がマウントされるので、その中の「VMwareTools-8.4.6-385536.tar.gz」¹¹をファイルマネージャから右クリックで解凍します。
- ④ 端末から解凍先の「vmware-install.pl」を実行します。インストールは Enter キーで進めてください。

```
$ sudo ./vmware-install.pl
```

- ⑤ 完了後、仮想マシンを再起動します。

動作環境がプロキシサーバを経由してインターネットに接続する場合、[システム]→[設定]→[ネットワークプロキシ]にてプロキシサーバの情報を設定してください。プロキシサーバ情報の設定手順は以下の通りです。

- ⑥ [システム]→[設定]→[ネットワークプロキシ]を選択します。

¹¹ VMware Player のバージョンによってはファイル名が異なる場合があります。

- ⑦ [マニュアルでプロキシの設定を行う(M)]で[すべてのプロトコルで同じプロキシを使う(U)]を選択します。
- ⑧ [閉じる]→[システム全体に適用]を選択します。
- ⑨ 認証パスワードを問われるので設定したパスワードを入力して[認証]を選択します(これより以下の手順で、認証を問われたら同様にパスワードを入力してください)。

(5) 共有フォルダの設定

- ① 仮想マシン「FuzzUbuntu」がパワーオフ状態で、[仮想マシン設定の編集(D)]を選択します。
- ② [オプション]→[共有フォルダ]→[常に有効]→[追加]を選択します。
- ③ [ホストパス(H)]と[名前(A)]を設定し[次へ(N)]選択します。この説明ではホストパスに「D:\vmshare」、名前に「vmshare」を設定します。
- ④ [共有する(E)]→[完了]を選択します。
- ⑤ 仮想マシン「FuzzUbuntu」を起動し、ログインします。
- ⑥ 「/mnt/hgfs/vmshare」をls コマンドなどで確認します。
- ⑦ ホスト OS の「D:\vmshare」とゲスト OS の「/mnt/hgfs/vmshare」が共有されます。

■ ツールをインストールする。

(6) Wireshark のインストール

- ① 仮想マシン「FuzzUbuntu」の端末から下記コマンドを実行し、Wireshark をインストールします。

```
$ sudo apt-get install wireshark
```

- ② 端末から下記コマンドを実行し、一般ユーザ権限が wireshark でキャプチャできるように設定します。

```
$ sudo setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/dumpcap
```

- ③ [アプリケーション]→[インターネット]→[Wireshark]を選択し、Wireshark を起動します。
- ④ Wireshark のメニューから[Capture]→[Interfaces]を選択しデバイス (eth0 など) が表示されていることを確認します。

(7) tcpreplay, tcpreplay-edit, tcprewrite のインストール

- ① 仮想マシン「FuzzUbuntu」の端末から下記コマンドを実行し、tcpreplay, tcpreplay-edit, tcprewrite をインストールします。

```
$ sudo apt-get install tcpreplay
```

- ② 端末から下記コマンドを実行し、tcpreplay などがインストールされたことを確認します。

```
$ tcpreplay -V
$ tcpreplay-edit -V
$ tcprewrite -V
```

(8) Subversion のインストール

- ① 仮想マシン「FuzzUbuntu」の端末から下記コマンドを実行し、Subversion をインストー

ルします。この Subversion は Wireplay の最新のソースコードを取得するために使います。

```
$ sudo apt-get install subversion
```

② 端末から下記コマンドを実行し、Subversion がインストールされたことを確認します。

```
$ svn --version
```

動作環境がプロキシサーバを経由してインターネットに接続する場合、
「/etc/subversion/servers」の[global]項目にプロキシサーバ情報を設定します。

```
[global]
http-proxy-host = プロキシサーバのホスト名
http-proxy-port = プロキシサーバのポート番号
```

(9) Wireplay のインストール

① 仮想マシン「FuzzUbuntu」の端末から下記コマンドを実行し、必要なライブラリをインストールします。

```
$ sudo apt-get install ruby ruby-dev libruby
$ sudo apt-get install libpcap0.8 libpcap0.8-dev
$ sudo apt-get install libnet1 libnet1-dev
```

② 端末から下記コマンドを実行し、Wireplay の最新のソースコードを取得します。

```
$ svn checkout http://wireplay.googlecode.com/svn/trunk/ wireplay-read-only
```

③ 「./wireplay-read-only/libnids-1.23」に移動し下記コマンドを実行します。

```
$ ./configure --enable-shared --disable-libglib
```

④ 「./wireplay-read-only/libnids-1.23/src/killtcp.c」をテキストエディタ (vi 等) で編集します。

```
121 行目
変更前: #elif
変更後: #else
```

⑤ 「./wireplay-read-only/libnids-1.23」で make コマンドを実行します。

```
$ make
```

⑥ 「./wireplay-read-only/Makefile」をテキストエディタで編集します。

```
3 行目
変更前: RUBYINC := /usr/lib/ruby/1.8/i486-linux
変更後: RUBYINC := /usr/lib/ruby/1.8/i686-linux
```

⑦ 「./wireplay-read-only」で make コマンドを実行します。

```
$ make
```

⑧ 「./wireplay-read-only」で下記コマンドを実行し Wireplay をインストールします。

```
$ sudo make install
```

⑨ 下記コマンドを実行します。

```
$ sudo ln -s /opt/wireplay/bin/wireplay /usr/bin
```

⑩ 下記コマンドを実行し、Wireplay がインストールされたことを確認します。

```
$ ldd /usr/bin/wireplay
```

以上で、ファジング結果の再現環境の作成は完了です。

3.3. ファジング結果の再現手順

3.2.2 節で作成した再現環境を使って、ファジング結果の再現手順を説明します。この節における再現環境は図 3-1 を想定します。

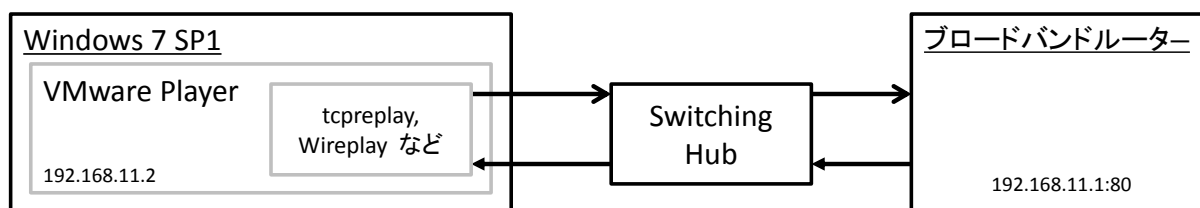


図 3-1 再現環境

ファジング結果の再現手順は 3 つの工程から成り立ちます (図 3-2)。3.3.1 節からこの 3 つの工程を説明します。

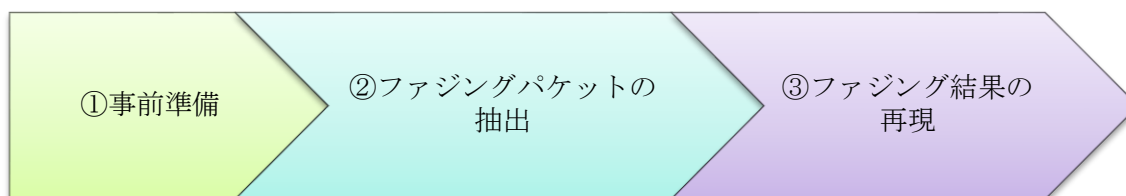


図 3-2 ファジング結果の再現手順

- ① 事前準備 (3.3.1 節)
- ② ファジングパケットの抽出 (3.3.2 節)
- ③ ファジング結果の再現 (3.3.3 節)

3.3.1. 事前準備

ファジング結果を再現する前に、主に次の点を確認してください。

- VMware Player の仮想マシン設定の共有フォルダが有効になっているか。
- 再現環境の Windows7 および仮想マシンからブロードバンドルーターにネットワークを経由してアクセスできるか。

3.3.2. ファジングパケットの抽出

事前準備が完了したら、ネットワークを介したファジングで取得したパケットキャプチャファイルから実際に再現するパケットを抽出します。

どのようなパケットを抽出する必要があるのかは、ファジングの内容により異なります。ネットワークを介したファジングには大きく 2 種類のパケットパターンがあります。

- (1). 一方的にパケットを送るファジング（例えば、IPv4 におけるファジング）
- (2). クライアント／サーバ間でパケットを相互にやり取りする（セッションを確立する）ファジング（例えば、HTTP におけるファジング）

本節および 3.3.3 節では、(1)を「IPv4 タイプ」、(2)を「HTTP タイプ」と定義して、2 つのタイプごとに説明していきます。

■ IPv4 タイプ

ここでは IPv4 におけるファジングを例にとり、ファジングパケットの抽出手順を説明します。ファジング結果を再現する前に、図 3-3 の環境で実施した IPv4 におけるファジングのパケットキャプチャファイルを取得していると仮定します。

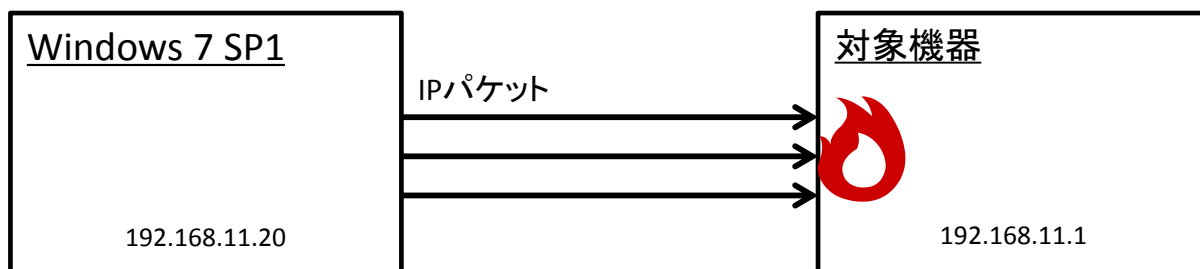


図 3-3 IPv4 におけるファジングを実施した環境

- ① Wireshark でパケットキャプチャファイルを開きます。
- ② Wireshark のフィルタ機能で必要なパケットのみ抽出します。例えば、192.168.11.20 と 192.168.11.1 の IP 通信（ICMP パケットを除く）を抽出したい場合、以下のようなフィルタとなります。

フィルタ例「`ip.addr eq 192.168.11.20 and ip.addr eq 192.168.11.1 and !icmp`」

- ③ [File]→[SaveAs]で保存画面を表示し、その画面中の[Packet Range]で[Displayed]にチェックをつけて保存します。この結果、フィルタしたパケットのみ抽出できます。

■ HTTP タイプ

ここでは HTTP におけるファジングを例にとり、ファジングパケットの抽出手順を説明します。ファジング結果を再現する前に、図 3-4 の環境で実施した HTTP におけるファジングのパケットキャプチャファイルを取得していると仮定します。

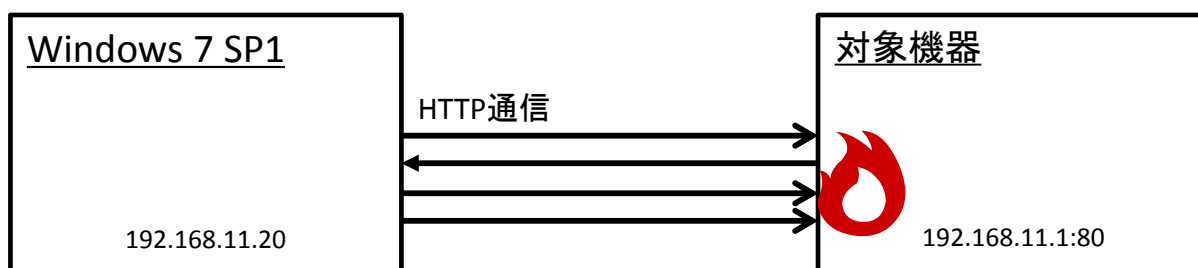


図 3-4 HTTP におけるファジングを実施した環境

- ① Wireshark でパケットキャプチャファイルを開きます。
- ② Wireshark でフィルタをかけ必要なパケットのみを抽出します。例えば、HTTP の標準ポートである 80/tcp が関連する HTTP 通信のみ抽出したい場合、以下のようなフィルタとなります。

フィルタ例「tcp.port == 80」

- ③ [File]→[SaveAs]で保存画面を表示し、その画面中の[Packet Range]で[Displayed]にチェックをつけて保存します。その結果、フィルタしたパケットのみ抽出できます。

3.3.3. ファジング結果の再現

3.3.2 節で再現したいファジングパケットを抽出したら、再現環境の仮想マシンでそのパケットを再現します。ファジングの IPv4 タイプと HTTP タイプによって、使うツールが異なります。次の 2 つのポイントからファジング結果の再現に使用するツールを決定します。

- パケットを加工する必要があるか (IP アドレスなどを変更する必要があるか)
- セッションを再現する必要があるか

ファジングを実施した環境と異なる環境でファジング結果を再現する場合、再現環境の IP アドレスや MAC アドレスがパケットキャプチャファイルに記録されているものと異なります。このときパケットキャプチャファイルの IP アドレスや MAC アドレスを再現環境と合わせる必要があります。パケットを加工する必要がある場合、さらに「ファジング結果再現時に加工するか」、「事前に加工するか」で使うツールが変わります。

また TCP を使用したネットワーク通信を再現する場合、ファジング PC とファジング対象機器の一連の通信 (セッション) を順番に再現する必要があります。

以下にまとめます。

- ① パケットを加工せず(IP アドレスなどを変更せず)に再現する場合
→ `tcpreplay`
- ② パケットの IP アドレスなどを変更して再現する場合
→ `tcpreplay-edit`
- ③ パケットの IP アドレスなどを変更したパケットキャプチャファイルを作成する場合
→ `tcprewrite`
- ④ セッションを再現する場合
→ `wireplay`

それでは「IPv4 タイプ」、「HTTP タイプ」ごとに実際のファジング結果を再現するコマンドを説明します。コマンド中の「`dump.pcap`」が 3.3.2 節で抽出したパケットキャプチャファイルになります。

■ IPv4 タイプ

ここでは IPv4 におけるファジングを例にとり、ファジング結果を再現します。

- ① パケットを加工する必要がない場合、「`tcpreplay`」を使用します。

```
$ sudo tcpreplay -i eth0 dump.pcap
```

- ② IP アドレスと送信元 MAC アドレス、送信先 MAC アドレスを加工する必要がある場合、「`tcpreplay-edit`」を使用します。この例では、IP アドレス「192.168.11.20」を「192.168.11.2」に、送信元 MAC アドレスを「00:0c:29:7b:88:c7」に、送信先 MAC アドレスを「00:0c:29:7b:76:c8」に変更しています。

```
$ sudo tcpreplay-edit -N 192.168.11.20:192.168.11.2 --enet-dmac=00:0c:29:7b:76:c8 --enet-smac=00:0c:29:7b:88:c7 -i eth0 dump.pcap
```

■ HTTP タイプ

ここでは HTTP におけるファジングを例にとり、ファジング結果を再現します。

- ① HTTP のセッションを再現する必要がある場合、「`wireplay`」を使用します。

```
$ wireplay -r client -t 192.168.11.1 -p 80 -F dump.pcap -K
```

更新履歴

更新日	更新内容
2012年3月27日	第1版 発行。
2013年3月18日	第1版 第2刷発行。 「2.6 Peach が生成するファズデータ」を更新。
2016年3月31日	第1版 第3刷発行。 「2 ファジングツール「Peach」の使い方」を更新。 本書ダウンロード URL を更新。
2017年3月3日	第2版 発行。 「1 ファジングツール「Taof」の使い方」を更新。 「2 ファジングツール「Peach Fuzzer」の使い方」を更新。 本書ダウンロード URL を更新。

著作・制作 独立行政法人情報処理推進機構（IPA）

編集責任 金野 千里

執筆者 小林 桂
岡崎 圭輔
鹿野 一人
熊谷 悠平

協力者 桑名 利幸
渡辺 貴仁
板橋 博之
扇沢 健也
大道 晶平
菅原 尚志
田村 智和

※独立行政法人情報処理推進機構の職員については所属組織名を省略しました。

「ファジング活用の手引き」別冊

ファジング実践資料

— ファジングツールの使い方とファジング結果の再現手順 —

[発行] 2012年 3月27日 第1版
2013年 3月18日 第1版 第2刷
2016年 3月31日 第1版 第3刷
2017年 3月 3日 第2版

[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