SSH-2 サーバ向けファジングツール

兒島 尚† 中田 正弘†

†株式会社富士通研究所
211-8588 神奈川県川崎市中原区上小田中 4-1-1
{hkojima, nakada.masahiro}@jp.fujitsu.com

あらまし SSH(Secure Shell)は通信路の認証や暗号化を実現する標準プロトコルであり，現在主流のバージョンであるSSH-2を実装したサーバは様々な製品の管理コンソールの保護等に広く利用されている．SSH-2サーバの実装に問題があれば安全性が損なわれるため十分なセキュリティ評価が必要である．このため我々はSSH-2サーバを対象としたファジングツールを開発した．本ツールはプロトコル実装評価で近年利用が増えているファジング手法に基づき，正常パケットを変異させたテストパケットを多数送信してサーバの挙動を監視する．本ツールは従来ツールが対応していなかったSSH-2メッセージに対応し，複数のネットワーク製品で脆弱性を検出した．

A Fuzzing Tool for SSH-2 Servers

Hisashi Kojima† Masahiro Nakada†

†Fujitsu Laboratories Ltd.
1-1, Kamikodanaka 4-chome, Nakahara-ku, Kawasaki, Kanagawa 211-8588 Japan
{hkojima, nakada.masahiro}@jp.fujitsu.com

Abstract SSH (Secure Shell) is a protocol to protect network channels and the mainstream version SSH-2 is widely used such as for management consoles of various products. SSH-2 server implementations should be sufficiently evaluated since defects of them hurt security of such products. Therefore we have developed a fuzzing tool for SSH-2 servers. The tool sends many test packets mutated from normal packets to servers and observes their behaviors, based on fuzzing techniques recently being used for evaluation of protocol implementations. It supports more kinds of SSH-2 messages than similar tools and has revealed vulnerabilities of multiple network products.

1 はじめに

SSH（Secure Shell）[1]は通信路の認証や暗号化を実現するネットワークプロトコルである．同種の技術である TLS[2]と比べて標準でリモートシェルに対応するなどの特徴があり，現在主流のバージョンである SSH-2 を実装したサーバが様々な製品のコマンドラインベースのリモート管理コンソールの保護などに広く利用されている．SSH-2 サーバの実装に問題があれば依存する通信の安全性が損なわれるため，製品の出荷前に十分なセキュリティ評価を実施する必要がある．近年，ネットワークプロトコルのセキュリティ評価でファジング[3]の有効性が

注目されている. ファジングとは対象が通常想定しないような入力データを大量に与えることでソフトウェアの不具合を発見するためのテスト手法であり, SSH-2 でも多数の製品で脆弱性が検出されている[6]. しかし我々の知る限り, 従来のファジングツールは SSH-2 メッセージの全てには対応していなかった. 本稿ではより多くのメッセージに対応する, 我々が開発した SSH-2 サーバ向けファジングツールについて説明する.

本稿の構成を以下に示す. まず 2 章で関連研究として従来ファジングツールの SSH-2 対応状況を説明し, 本ツールが対象とする SSH-2 メッセージの種類を示す. 3 章では本ツールによる具体的なテスト方法について説明し, 4 章でその実装について述べる. 5 章では実際のネットワーク製品を対象に行った評価について説明し, 多くの製品で脆弱性を検出した実績を示す. 最後に 6 章でまとめと今後の課題を述べる.

2 関連研究

SSH-2 では 29 種類のメッセージが定義されており, このうち 19 種類がクライアントからサーバに送られる. また, これらとは別にメッセージを暗号化する際のパケット構造が BP（Binary Packet）として定義されている. 他に SFTP（SSH File Transfer Protocol）[5]という SSH 上で FTP を実現するプロトコルがあるが, 実装依存とされている. よって, SSH-2 サーバー般の評価では, 19 種類のサーバ向けメッセージと BP に対応することがまず必要と考えられる.

SSH-2 のファジングにおいては幾つかの無償ツールが公開されている. SSHredder[6]は生成済みのテストパケット群として公開され, KEXINIT のファジングに対応しており, 多数の SSH-2 サーバで脆弱性を検出している. SSHFuZZ[7]は USERAUTH_REQUEST に含まれるユーザ名, ホスト名, 公開鍵に加えて, SCP[4][5]及び SFTP における各コマンドのパラメータのファジングを行う. また, 侵入テストフレームワークの Metasploit には SSH ファジング用モジュールがあり[8], バージョン文字列や KEXINIT のファジングに対応している. しかしこれらでは 3 種類のメッセージにしか対応できないため不十分である. 一方, 商用のファジングツールでは多くの製品が SSH-2 ファジングに対応しているが, その仕様の詳細が明らかにされている Codenomicon 社のツールを例に取ると, 対応メッセージは 15 種類で全てのメッセージはカバーしていない[9].

表 1 既存ツールと開発ツールの SSH-2 対応状況

ID	メッセージ or BP	SSHredder	SSHFuZZ	Metasploit	Codenomicon	開発ツール
--	バージョン文字列			X	X	
1	DISCONNECT				X	x
2	IGNORE				X	x
3	UNIMPLEMENTED					x
4	DEBUG				X	x
5	SERVICE_REQUEST				X	x
20	KEXINIT	x		X	X	
21	NEWKEYS				X	x
30	KEXDH_INIT				X	x
50	USERAUTH_REQUEST		X		X	x
53	USERAUTH_BANNER					x
80	GLOBAL_REQUEST				X	x
90	CHANNEL_OPEN				X	x
93	CHANNEL_WINDOW_ADJUST					x
94	CHANNEL_DATA				X	x
95	CHANNEL_EXTENDED_DATA				X	x
96	CHANNEL_EOF				X	x
97	CHANNEL_CLOSE					x
98	CHANNEL_REQUEST				X	x
--	BP					x

以上のことから，我々は評価すべきメッセージを網羅した新たな SSH-2 サーバ向けファジングツールが必要と考え，これを開発した．バージョン文字列と KEXINIT については無償ツールで既に対応されていることから，本ツールはこれらを除く 17 種類のサーバ向けメッセージと BP に対応する．

表 1 に既存ツールと本ツールの SSH-2 対応状況を示す．ID は SSH-2 の仕様で定義されたメッセージ毎に一意の番号である．

3 テスト方法

本章では開発ツールによる具体的なテスト方法について説明する．本ツールでは型テストとランダムテストの 2 つのテスト方法を用いている．

3.1 型テスト

型テストとは，各対象 SSH-2 メッセージの構造を考慮して，各フィールドをその型の種類に基づいて不正な値に変異させ，テストパケットを生成する方法である．開発ツールでは，図 1 に示す DISCONNECT の例のように C 言語の構造体に似た形式で記載することにより，テストパケットの内容を指定できる．

```
struct  DISCONNECT {
  const  byte      message_id      = 1;
         uint32    reason          = 2;
         string    description     = "";
         string    language        = "";
}
```

図 1 DISCONNECT のパケット構造

ここで，message_id フィールドは 1 バイトの符号なし整数の BYTE 型として定義されており，初期値は 1 である．ただし先頭に const 属性が付いているため，変異の対象とならず常に初期値が設定される．一方，他のフィールドは一つずつ順番に変異され，それぞれ独立したテストパケットとして生成される．

我々は BP を除く 17 種類のサーバ向け SSH-2 メッセージについて同様にパケット構造を記述した．ここで，各メッセージの先頭には必ずメッセージの ID を示す message_id フィールドがあるが，ここを変異させると目的のメッセージとして対象サーバに認識されない可能性があるため，message_id フィールドには全て const 属性を付与して変異させないようにしている．

3.1.1 SSH-2 の型と変異方法の対応

SSH-2 で定義された型を表 2 に示す．

表 2 SSH-2 で定義された型

型	説明	補足
byte	1 バイト値	
byte[n]	長さ n のバイト列	
boolean	1 バイト真偽値	0:偽 0 以外:真
uint32	32bit 符号なし整数	Big endian
uint64	64bit 符号なし整数	Big endian
string	文字列	uint32(長さ) +文字列(ヌルなし)
mpint	多バイト長整数	uint32(長さ) +整数(2 の補数)
name-list	文字列リスト	「,」(カンマ)で 区切られた string

これらの型に対して以下の変異方法を適用する．ファジングにおいては一般的に利用されている変異方法である．

- 文字列長：文字列の長さを整数の境界値付近の値(例:256, 65536)に変異させる．増やした分はパディング(例:'x')で埋める．主にバッファオーバーフローの発生が狙いである．
- 数値：数値の値を整数の境界値付近の値に変異させる．主に整数オーバーフローの発生が狙いである．
- バイト置換：文字列や数値の各バイトを不正な値(例:ヌル(0x00))に変異させる．主に文字列処理に伴うメモリエラーの発生が狙いである．
- 長さ不整合：長さ情報を含む場合

(string, mpint, name-list)に長さ情報と実際の長さを合わせないようにする. 主にバッファオーバーフローの発生が狙いである.

- ランダム: ランダムな長さのランダムな値に変異させる. 予測不可能な入力による誤動作の発生が狙いである.

表 3 に SSH-2 の型と変異方法の対応を示す. この対応に従って図 1 で定義した各メッセージのパケット構造を変異させてテストパケットを生成する.

表 3 SSH-2 の型と変異方法の対応

型	文字列長	数値	バイト置換	長さ不整合	ランダム
byte			x		x
byte[n]	x		x		x
boolean			x		x
uint32		x			x
uint64		x			x
string	x		x	x	x
mpint	x		x	x	x
name-list	x		x	x	x

3.1.2 SSH-2 プロトコル状態の考慮

SSH-2 は stateful なプロトコルであり, 表 4 に示すプロトコル状態が定義されている.

表 4 SSH-2 のプロトコル状態

プロトコル状態	説明
kexinit	鍵交換開始済
kexdh_reply	DH 鍵交換開始済
kex	鍵交換済
auth	認証済
channel	チャンネル接続済
shell	シェル確立済

各メッセージにはサーバで受理されうる適切な状態が定義されており, 不適切な状態でテストパケットを送信しても単に拒絶されて有効な評価が行えない恐れがある. よって, 各メッセージの送信前にサーバとある程度の正常通信を行って適切な状態にする必要がある. 表 5 に各メッセージと受理されうるプロトコル状態の対応を示す. 複数の状態で受理されうるメッセージについては, 同じテストパケットを状態毎に送信することで評価の網羅性を高めることとした.

また, SSH-2 では 1 つのコネクションの中で, チャンネルとよばれる論理的な通信路を複数持つことが可能であり, 各チャンネルはチャンネル ID という uint32 の値で識別される. そこで, テストパケットの生成の際にチャンネル ID を維持させるか変異させるかの 2 通りの方法が考えられるが, 網羅性向上のためどちらの場合も評価することとした. 該当メッセージを表 5 に示す.

表 5 各メッセージの受理されうる状態とチャンネル ID 考慮の必要性

メッセージ	受理されうる状態	Ch ID
DISCONNECT	all	
IGNORE	all	
UNIMPLEMENTED	all	
DEBUG	all	
SERVICE_REQUEST	kex / auth	
NEWKEYS	kexdh_reply	
KEXDH_INIT	kexinit	
USERAUTH_REQUEST	kex	
USERAUTH_BANNER	kex	
GLOBAL_REQUEST	all	
CHANNEL_OPEN	auth	
CHANNEL_WINDOW_ADJUST	channel	x
CHANNEL_DATA	channel	x
CHANNEL_EXTENDED_DATA	channel	x
CHANNEL_EOF	channel	
CHANNEL_CLOSE	channel	
CHANNEL_REQUEST	channel	x

3.1.3 コネクションの接続方法

型テストでは基本的に1つのテストパケットに対するサーバの反応を評価することが狙いのため, テストパケット毎にサーバとのコネクションを再接続して状態をリセットすることが考えられる. しかし, 再接続せずに複数テストパケットに渡って同一コネクションを使用し続けた場合, 高負荷状態での評価が期待できるという利点もある. そこで, 毎回再接続する方法と, 同一メッセージのテストパケットでは接続を維持する方法の 2 通りを評価することとした. 表 6 に両者の特徴を示す.

表 6 コネクションの接続方法による違い

接続方法	説明	メリット	デメリット
同種類維持	同一メッセージで接続維持	高負荷状態の評価が可能	接続直後以外は無視の恐れ
毎回再接続	テストパケット毎に再接続	各パケットが確実に処理	高負荷状態になりにくい

3.2 ランダムテスト

ランダムテストは型テストとは異なり，元のメッセージの構造を基本的に考慮せずに，ランダムな長さのランダムな値に変異させる方法である．対象とするメッセージの違いにより 3 つの異なる方法で評価を行う．

3.2.1 通常ランダム

3.1.2 節で述べた各プロトコル状態においてランダムパケットを送信する．この際，状態が channel / shell の場合はチャンネル ID の維持／変異を考慮する．また，3.1.3 節で述べたコネクションの接続方法も考慮する．送信されるテストパケットの例を図 2 に示す．

```
struct NORMAL_RANDOM {
  byte data[] = { 0x03, 0x52, 0x22, ... };
}
```

図 2 通常ランダムのテストパケットの例

3.2.2 ID 固定ランダム

通常ランダムと同様だが，各メッセージの ID を示す message_id フィールドだけを固定とし，残りをランダムデータとする．サーバが各メッセージの処理ルーチンに入った後でランダムデータを処理させることが狙いである．図 3 に例を示す．

```
struct FIXEDID_RANDOM {
   const byte message_id = 1;
         byte data[] = { 0x39, 0xf2, ... };
}
```

図 3 ID 固定ランダムのテストパケットの例

3.2.3 BP ランダム

メッセージではなく BP としてランダムデータを送信する．メッセージとして処理される可能性は低いが，その前段階のサーバでの復号処理において誤動作を発生させることが狙いである．チャンネル ID を考慮しないこと以外は，テストパケットの構造，プロトコル状態，コネクション接続方法は通常ランダムと同様である．

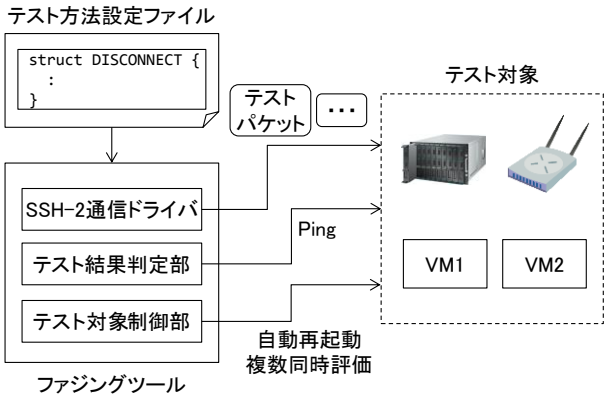

図 4 開発したファジングツールの構成

4 実装

本ツールの開発には Visual Studio を利用し，Windows で動作する GUI アプリケーションとして実装した．本ツールは主に SSH-2 通信ドライバ，テスト結果判定部，テスト対象制御部で構成される．ツールの構成を図 4 に示す．SSH-2 通信ドライバは 3 章で述べたパケット構造などを定義したテスト設定方法ファイルに基づき，SSH-2 メッセージの作成及び送信，プロトコル状態の準備などを行う．ツール自体は汎用的に設計されており，パケット構造を定義しテスト対象との通信用のドライバを追加することで，SSH-2 の他にも様々なプロトコルに対応できるようになっている．4.1 節では今回開発した SSH-2 通信ドライバについて説明する．テスト結果判定部はテストパケット送信ごとにテスト対象の挙動を監視して不具合発生の有無を判定する．4.2 節でテスト結果の判定方法について説明する．テスト対象制御部はテスト対象が応答停止になった場合に再起動するほか，同時

に複数のテスト対象を評価することでテストを効率化する. 4.3 節でテスト対象の制御方法について説明する.

4.1 SSH-2 通信ドライバ

SSH-2 サーバとの通信用のドライバの開発には libssh[10]を利用した. libssh は C 言語で開発されているオープンソースの SSH 通信ライブラリであり, SSH-1 及び SSH-2 のクライアント／サーバ通信に対応する. 3.1.2 節で述べたように, 本ツールでは各プロトコル状態を維持しつつテストパケットを送信する必要がある. libssh は標準で kex, auth, channel, shell でのプロトコル状態の維持には対応しているが, kexinit と kexdh_reply には未対応のため維持できるように改変を行った. また, auth, channel, shell の各プロトコルでは認証が必要になるため, 予めテスト対象として登録したユーザ ID 及びパスワードを送信するようになっている. さらに, 各プロトコル状態からそのままパケットを送信する機能(例:BP としてのパケット送信)も不足していたため新たに実装した.

4.2 テスト結果の判定

テスト対象における不具合の発生の検出のため, 本ツールではテストパケット送信毎に Ping の応答を調べる. もし応答がなければそのテストパケットが原因である可能性が高い. また, ネットワーク機器等, シリアルインターフェイス(例:RS-232c)経由のコンソール出力を持つ場合は, それらの出力も監視する.

4.3 テスト対象の制御

不具合発生等が原因でテスト対象機器が応答不能になった場合, テスト継続のためにテスト対象を再起動することが必要である. 本ツールでは物理的な機器についてはリブーター[11]とよばれる電源制御装置を利用して実現している. 一方, テスト対象は物理的な機器だけでなく VM(仮想マシン)の場合もあるため, VMware 社の VM 操作 API である VIX[12]に対応し, VMware Workstation で動作する VM については再起動できるようにした. レジュームにも対応するため, VM のテスト開始直前状態のスナップショットを用意しておけば高速な再起動及びテスト再開が可能である.

また, 複数のテスト対象を同時に制御することが可能であり, テストパケットを複数のテスト対象に割り振って並列に実行したり, 複数の異なるテスト対象に同じテストパケットを送信したりすることができる. 前者は VM のように同等のテスト対象を容易に複製できる場合に, 後者は物理的な異なる機器が複数ある場合に, いずれもテスト時間を大幅に短縮できる.

5 評価

本章では本ツールで行った評価結果について説明する. いずれの評価においても, 本ツールは CPU:Intel Core i7-2600 3.4GHz, メモリ:16GB, OS:Windows 7 Pro 64bit の PC 上で実行した.

5.1 OpenSSH

まず, コミュニティベースで開発されているデファクト実装である OpenSSH[13]を対象に評価を行った. OpenSSH は広く利用されていて SSH の評価対象としての重要性が高いことに加え, 2 章で述べたようにこれまでにも他ツールの評価対象となってきた経緯がある. よって, 比較的実装が頑強であると予想されるため, 本ツールが適切にテストを完了できるかどうかの確認も狙いとしている. 使用した OpenSSH のバージョンは 5.8p1 であり, OS には Ubuntu 11.10 x64 を使用し, 本ツールと同じ PC 上にインストールされた VMware Workstation で動作させている. こ

こでは，4.3 節で述べたテスト対象並列制御の機能を利用し，6個に複製したVMにテストパケットを割り振ってテストの高速化を図った．評価結果を表 7 に示す．

表 7 OpenSSH の評価結果

テスト方法	テスト数	接続方法	所要時間	評価結果
型テスト	368407	維持	約 5 時間	問題なし
		再接続	約 22 時間	問題なし
通常・BP ランダム	141399	維持	約 2 時間	問題なし
		再接続	約 7 時間	問題なし
ID 固定 ランダム	255199	維持	約 4 時間	問題なし
		再接続	約 13 時間	問題なし

結果が示す通り，特に問題は検出されなかった．よって，少なくとも今回，本ツールで定義したようなテスト方法に対してはOpenSSH の耐性は高いといえる．一方，本ツールがテストを適切に完了できることは確認できた．また，6 個の VM で並列化してはいるが，1 台の PC だけで約 76 万件のテストパケットを約 53 時間（約 2.2 日）で完了できており，一般的な計算機環境で実施可能な程度のテスト時間といえる．

5.2 ネットワークスイッチ製品

次に，一般に販売されているネットワークスイッチ製品について評価を行った．ここでは複数のベンダーを対象に 6 種類の製品（D1#1，D1#2，D1#3，C#1，D2#1，H#1）を選択した．いずれも 10/100M の Ethernet に対応し，管理コンソールのための SSH-2 サーバを搭載している．我々はこれらの製品を本ツールが動作する PC と同一セグメントのLAN に配置して評価した．VM のように複製はできないため，各製品を 1 台ずつ用意して計 6 台の製品を同時に並列に評価することでテスト時間を短縮できるようにした．また，シリアルインターフェイスを持つ場合はその出力も記録するようにした．評価結果を表 8 に示す．

結果が示す通り，H#1 を除く 5 製品で脆弱性を検出できた．いずれの場合もテストパケット送信直後に製品が応答不能になり，Ping も返さず，しばらくすると自動的に再起動された．そしてシリアルコンソールには例外発生を示すエラーコードやスタック・レジスタの値が出力されていた．脆弱性は全て型テストでコネクション接続方法を再接続にした場合にのみ検出されており，接続方法を維持にした場合やランダムテストでは検出されなかった．製品の安全性のため脆弱性の詳細はここでは述べられないが，本ツールで新たに対応したものを含む複数の SSH-2 メッセージのテストパケットで検出されており，特定のフィールドが存在しなかったり（つまり長さが 0），不正な値が入力されていたり（例えば 0xffffffff）した場合に発生していた．よって，これらネットワークスイッチのような製品に対して本ツールは効果的といえる．これらの脆弱性は情報セキュリティ早期警戒パートナーシップ[14]に基づき IPA に報告済であり，一部は公開され[15]，他は現在ベンダーと調整中となっている．

また，各製品の SSH サーバの種類は返されたバージョン文字列の情報から推測したものである．D1#1～3 は独自実装と考えられる．一方，C#1 と D2#1 では OpenSSH にも関わらず脆弱性が検出されているが，H#1 では検出されていないため，バージョンが「3.4p1」と古いことが原因と考えられる．ベンダーが改変している可能性もあるが，このバージョンの OpenSSH 自体が脆弱である可能性がある．これは 5.1 節で対象とした 5.8p1 等の最近のものと比べると大幅に古いが，組み込み機器においては古いバージョンを使い続けている場合があるため注意が必要といえる．

表 8 ネットワークスイッチ製品の評価結果

製品	D1#1	D1#2	D1#3	C#1	D2#1	H#1
SSH サーバ	独自	独自	独自	OpenSSH 3.4p1	OpenSSH 3.4p1	OpenSSH 3.7.1p2
脆弱性	6 件検出	2 件検出	3 件検出	2 件検出	2 件検出	問題なし

6 まとめ

本稿では我々の開発したSSH-2サーバ向けファジングツールについて述べた．本ツールは従来のファジングツールが対応していなかった SSH-2 メッセージの評価に対応し，SSH-2 の仕様を考慮した型／ランダムテストという方法により，SSH-2 サーバの評価に必要と考えられる約 76 万件のテストパケットを評価できる．本ツールによる評価では，OpenSSH については特に問題は検出されなかったが，ネットワークスイッチ製品においては評価した 6 製品中 5 製品で脆弱性を検出することに成功した．

今後の課題としては，テスト方法の追加と様々なSSH-2サーバへの適用が考えられる．例えば受理されないはずのプロトコル状態でテストパケットを送信したり，型テストにおける変異数を増やしたりすることが考えられる．また，SSH-2 による管理コンソールはネットワークスイッチ製品だけでなく，ダイヤルアップルータ，ストレージなど，様々な製品に搭載されており，本ツールでの評価が有効と考えている．

参考文献

[1] Wikipedia, “Secure Shell,” http://en.wikipedia.org/wiki/Secure_Shell

[2] Wikipedia, “Transport Layer Security,” http://en.wikipedia.org/wiki/Transport_Layer_Security

[3] Wikipedia, “Fuzz testing,” http://en.wikipedia.org/wiki/Fuzz_testing

[4] Wikipedia, “Secure Copy,” http://en.wikipedia.org/wiki/Secure_copy

[5] Wikipedia, “SSH File Transfer Protocol,” http://en.wikipedia.org/wiki/SSH_file_transfer_protocol

[6] Rapid7, “Rapid7 Advisory R7-0009: Vulnerabilities in SSH2 Implementations from Multiple Vendors,” http://www.rapid7.com/resources/advisories/R7-0009.jsp

[7] Jeremy Brown, “SSHFuZZ - SSH Fuzzer,” https://github.com/wireghoul/sploit-dev/blob/master/sshfuzz.pl

[8] Rapid7, “metasploit-framework,” https://github.com/rapid7/metasploit-framework/tree/master/modules/auxiliary/fuzzers/ssh

[9] Codenomicon, “Codenomicon SSH2 Test Tool Data Sheet,” http://www.codenomicon.com/products/ssh2x.shtml

[10] libssh, http://www.libssh.org/

[11] 明京電機, ”TIME BOOT,” http://www.meikyo.co.jp/products/mt8f.html

[12] VMware, “VIX API Documentation,” http://www.vmware.com/support/developer/vix-api/

[13] OpenSSH, http://www.openssh.org/

[14] IPA, “脆弱性関連情報の届出,” http://www.ipa.go.jp/security/vuln/report/index.html

[15] CVE-2013-1154, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1154