実践で学ぶ

一歩進んだサーバー構築・運用術

written by 仙石 浩明

第10回 ssh(中編)

ssh(セキュア・シェル)の応用方法を解説します。sshでは，直接TCP/IP接続できないホストに接続するために，ProxyCommandを設定できます。ファイアウオールを越えるProxyCommandを作成すれば，ファイアウオールの向こう側にあるホストを，LAN上のホストと同様に扱えます。

情報処理振興事業協会(IPA)が通商産業省の補助を受けて「未踏ソフトウェア創造事業」を開始しました*1。新聞などでも取り上げられたのでご存じの方も多いと思いますが，その採択者の中に私の名前があることに気付いた人は多くはないでしょう。採択者の何人かは取材を受けたようですが，ニュース性の高い人だけのようで，幸か不幸か私のところには取材の話は舞い込みませんでした。まあ，国が支援する対象としては，会社勤めの人よりは学生さんなどの方が珍しいのは当然と言えますね。

私が提案したテーマは「携帯電話用アプレット開発ツール」です。より多くの人が手軽に携帯電話上のプログラムを開発できる環境を作りたい，というのがケイ・ラボラトリー*2設立の趣旨ですから，今回のIPAの事業の話は，まさに渡りに船で，それまで温めていた技術を実現する絶好の機会と言えます。幸い採択して頂けたので，これから実現に向けて頑張りたいと思います。開発期間は2001年2月までですが，この時期いよいよJavaを搭載した携帯電話が登場します。極めて忙しい年末年始になりそうです。

sshを使ったファイアウオール越え

12月号では，ファイアウオールの内側の社内LANからインターネット上のホストへ，Webプロキシを経由してssh接続する方法を紹介しました。この方法では接続先のホストのポート443番でsshサーバーが動いている必要があり，そのようなホストが無い場合は使えません。

telnetプロキシ

もし，telnetプロトコルを中継するtelnetプロキシがファイアウオール上で動いている場合は，Webプロキシの場合と同様にして，telnetプロキシ経由でssh接続することもできます。

telnetプロキシとは，例えば**図1**に示すような手順*3で，インターネット上のホストへtelnet接続するためのものです。この例ではtelnetプロキシproxy.klab.org経由でホストasao.gcd.orgに接続しています。図1の最後の行は，asao.gcd.orgのtelnetサーバーが送信したログイン・プロンプトです。

telnetプロキシにはいろいろなものがありますが，大抵はユーザーIDとパスワードを入力して認証を行った後，インターネット上の任意のホスト・ポートに対してTCP/IP接続する*4，という使い

＊1　情報技術，特にソフトウエア関連分野での優秀な開発者の発掘と支援を目的に，通商産業省の外郭団体である情報処理振興事業協会(IPA)が実施。今回が第1回目になる。http://www.ipa.go.jp/を参照。
＊2　次世代携帯電話のコンテンツ関連技術に特化した研究開発型企業です。http://www.klab.org/を参照。
＊3　この実行例は架空のもので実在しません。
＊4　大企業などの場合，事業所のファイアウオールと全社のファイアウオールの2段構成になっていて，事業所のtelnetプロキシの認証を行った後，全社のtelnetプロキシへ接続しなければならない場合もあるでしょう。

```
kamiya:/home/sengoku % telnet proxy.klab.org↵
Trying 10.0.0.1...
Connected to proxy.klab.org.
Escape character is '^]'.
proxy.klab.org telnet proxy ready:
Username: sengoku ↵
Password: ########↵
Login Accepted
telnet-proxy> c asao.gcd.org↵
Trying 210.145.125.162 port 23...
login:                                    ↵このマークで改行
```

図1 telnetプロキシ

```
telnet-proxy> c asao.gcd.org 22↵
Trying 210.145.125.162 port 22...
SSH-1.99-OpenSSH_2.2.0p1
```

図2 telnetプロキシ経由で22番ポートへつなぐ

```
(1) telnetプロキシ（図1の場合であればproxy.klab.orgの23番ポート）
    に接続する。
(2) 「Username:」を受信するまで待つ。
(3) 「sengoku」（ユーザーID）を送信する。
(4) 「Password:」を受信するまで待つ。
(5) 「########」（パスワード，伏せ字にしています）を送信する。
(6) 「telnet-proxy>」を受信するまで待つ。
(7) 「c $1 $2」（インターネット上のホストへの接続コマンド）を送信する。
    ただし，$1，$2はproxy-telnetコマンドの第1，第2パラメータ。
(8) 「...↵」を受信するまで待つ。
(9) (8) の「...↵」以降に受信したキャラクタを全てそのまま標準出力へ出力する一方で，
    標準入力から入力したキャラクタをすべてそのまま送信する。
```

図3 proxy-telnet コマンドの動作

```
Host *
        ProxyCommand    /home/sengoku/bin/proxy-telnet %h %p
```

図4 ~/.ssh/configでProxyCommandを指定

方になります。

図1では，ホストasao.gcd.orgの23番ポートへ接続していますが，telnetプロキシの多くは，接続するポート番号を指定できるようです。この例の場合には，「telnet-proxy>」プロンプトにおいて**図2**のように入力すれば，asao.gcd.orgの22番ポートにつなぐことができます。図2の最後の行は，asao.gcd.orgの22番ポートのsshサーバーが送信した文字列です。

proxy-telnetコマンド

以上のようなtelnetプロキシが利用できる場合，**図3**の動作を行うproxy-telnetコマンドを作成し，~/.ssh/configでProxyCommandを**図4**のように指定すれば，telnetプロキシ経由でssh接続できます（**図5**）。

proxy-telnetなどsshのProxyCommandに指定するコマンドは，どのようなプログラミング言語で記述しても良いのですが，手軽に書けるという理由から私はいつもPerlを使っています。Perlだと，TCP/IP接続，送受信などの手続きが簡潔に記述でき，また図3にあるような「～を受信するまで待ち」という

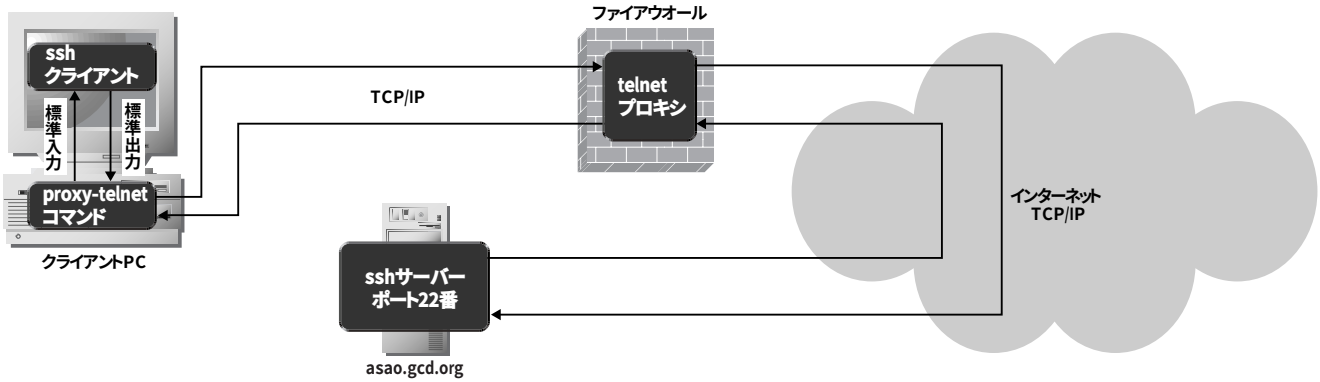

図5 telnetプロキシ経由のssh接続

```
#! /usr/bin/perl
$PROXY_TELNET = "proxy.klab.org";
$PROXY_PORT = 23;
$Verbose = 0;

while ($_ = shift) {
    last if ! /^-(.*)/;
    if ($1 =~ /^v+$/) { $Verbose += length($&); next; }
    print<<EOF;
Usage: proxy [option...] <host> <port>
Options:
    -v    Verbose mode
EOF
}
$HOST = $_;
if ($_ = shift) {
    $PORT = $_;
} else {
    $PORT = 23;
}
print "Verbose Level: $Verbose\n" if $Verbose;

use Socket;
($name, $aliases, $proto) = getprotobyname('tcp');
($name, $aliases, $type, $len, $thataddr) =
gethostbyname($PROXY_TELNET);
$that = sockaddr_in($PROXY_PORT, $thataddr);
socket(S, PF_INET, SOCK_STREAM, $proto) || die "socket: $!";
connect(S, $that) || die "connect: $!";

if ($Verbose > 1) {
    $Rin = &fhbits('STDIN S');
} else {
    $Rin = &fhbits('S');
}
&login;
&connect;
exit 0;

# login 処理
sub login {
    do { &receive(0.1); } until (/name:/);
    &send("sengoku\r");
    do { &receive(0.1); } until (/word:/);
    &send("########\r");
    do { &receive(0.1); } until (/proxy>/);
    &send("c $HOST $PORT\r");
    do { &receive(0.1); } until (/\.\.\.\n/);
    $Raw =~ m/\.\.\.[\r\n]+/;
    $Raw = $';
}

# connect
sub connect {
    my($rout,$len);
    print "CONNECT\n" if $Verbose;
    $Rin = &fhbits('STDIN S');
    syswrite(STDOUT,$Raw,length($Raw));
    while ((select($rout=$Rin,undef,undef,undef))[0]) {
        if (vec($rout,fileno(S),1)) {
            $len = sysread(S,$_,1024);
            return if $len <= 0;        # EOF
            syswrite(STDOUT,$_,$len);
        }
        if (vec($rout,fileno(STDIN),1)) {
            $len = sysread(STDIN,$_,1024);
            return if $len <= 0;        # EOF
            syswrite(S,$_,$len);
        }
    }
}

# send(str);  str を送る
sub send {
    undef $Buffer;
    undef $Raw;
    while( $_ = shift ) {
        print if $Verbose > 2;
        syswrite(S,$_,length);
    }
}

# receive(s);  s 秒入力が途絶えるまで待つ
sub receive {
    my($timeout) = shift;
    my($rout);
    while ((select($rout=$Rin,undef,undef,$timeout))[0]) {
        if (vec($rout,fileno(S),1)) {
           exit 1 if sysread(S,$_,1024) <= 0;          # EOF
            $Raw .= $_;
            tr/\r\000\012\021\023\032/\n/d;
            $Buffer .= $_;
            print if $Verbose > 1;
        }
        if (vec($rout,fileno(STDIN),1)) {
           exit 1 if sysread(STDIN,$_,1024) <= 0;      # EOF
            s/\n/\r/g;
            syswrite(S,$_,length);
        }
    }
    $_ = $Buffer;
}

sub fhbits {
    my(@fhlist) = split(' ',$_[0]);
    my($bits);
    for (@fhlist) {
        vec($bits,fileno($_),1) = 1;
    }
    $bits;
}
```

図6　Perlで書いたproxy-telnetコマンド
スクリプトには左端の行番号は不要である。

動作が正規表現を使って書けるので便利です。perlスクリプトで書いたproxy-telnetコマンドの例を**図6**に示します。

図6の22行目までは，パラメータの読み込みやデフォルト値の設定です。**表1**で示した変数が設定されます。

23行目から28行目までは，telnetプロキシへTCP/IP接続を行います。29行目以降では，ソケットSに対して読み書きすれば，telnetプロキシへ送受信することができます。これは図3の(1)の動作です。

35行目のloginサブルーチンの呼び出しにおいて，telnetプロキシ経由で接続先のsshサーバーへの接続を行います。これは図3の(2)から(8)までに対応します。

36行目のconnectサブルーチンの呼び出しは，標準入力から入力したキャラクタすべてをtelnetプロキシへ送信（ソケットSへ出力）し，telnetプロキシから受信（ソケットSから入力）したキャラクタすべてを標準出力へ出力します。これは図3の(9)に対応します。connectサブルーチンは，標準入力あるいはソケットSからの入力のいずれかがEOFになるまで続きます。

39行目から50行目までがloginサブルーチンです。この部分を書き換えれば，どのようなtelnetプロキシにも対応できます。図3に示したproxy-telnetコマンドの動作と，図6のperlスクリプトの対応関係を，**表2**に示します。

49行目において，変数「Raw」に「…↵」以降のキャラクタ列を代入しています。これは図3(9)にあるように，「…↵」以降に受信したキャラクタを標準出力へ出力する必要があるためです。実際の出力は，connectサブルーチン内の57行目で行います。

52行目から70行目までが，connectサブルーチンです。58行目のselectは，標準入力とソケットSのどちらかからキャラクタが入力されるまで待ちます。

60行目から62行目までが，ソケットSからキャラクタが入力された場合の処理です。つまり，telnetプロキシからキャラクタを受信した場合です。EOFであればサブルーチンを終了し(61行目)，そうでなければ受信したキャラクタを，そのまま標準出力に出力します(62行目)。

65行目から67行目までが，標準入力からキャラクタが入力された場合の処理です。EOFであればサブルーチンを終了し(66行目)，そうでなければ入力したキャラクタを，そのままソケットSへ出力します(67行目)。つまりtelnetプロキシへ送信します。

72行目から80行目までが，キャラクタ列をtelnetプロキシへ送信するsendサブルーチンです。単にキャラクタ列をソケットSへ出力するだけですが，後述する受信用バッファである変数BufferとRawをクリアします。また，デバッグのために送信キャラクタ列を標準出力へ出力することもできます。

82行目から101行目までが，telnetプロキシからキャラクタ列を受信するreceiveサブルーチンです。引数で指定した秒数の間,受信が途絶えるまで待ちます。受信したキャラクタ列は変数BufferとRawに設定されます。変数Rawは受信したキャラクタ列そのままであり，Bufferはキャラクタ列中の0x0D(改行)を「\n」へ置き換え，キャラクタ0x00，0x0A，0x11，0x13，0x1Aを削除したキャラクタ列です。loginサブルーチンで受信キャラクタ列を比較する際，以上の文字列が入っていると煩雑になるので，このような置き換えと削除を行っています。

86行目のselectはソケットSからの入力を待ちます。変数Verboseの値が2以上であれば(30行目から34行目)，selectはソケットSと標準入力からの両方の入力を待ちます。

88行目から92行目までが，ソケットSから入力された場合で，受信したキャラクタを変数BufferとRawに追加します。変数Verboseの値が2以上であれば，変数Bufferに追加した内容を標準出力に出力します。

95行目から97行目までが，標準入力か

表1　図6で用いる変数

変数	内容
PROXY_TELNET	telnetプロキシのホスト名
PROXY_PORT	telnetプロキシのポート番号
HOST	接続先（sshサーバー）のホスト名（第1パラメータ）
PORT	接続先（sshサーバー）のポート番号（第2パラメータ）
Verbose	デバッグ用フラグ（値が1以上の時，デバッグ・モード）

表2　proxy-telnetコマンドの動作とperlスクリプトの対応関係

proxy-telnetの動作（図3）	perl スクリプト（図6）の対応行
(2)「name:」を待つ	41行目
(3)「sengoku」を送信	42行目
(4)「word:」を待つ	43行目
(5)「########」（伏せ字）を送信	44行目
(6)「proxy>」を待つ	45行目
(7)「c $HOST $PORT」を送信	46行目
(8)「…↵」を待つ	47行目

らキャラクタが入力された場合で，入力されたキャラクタはそのままtelnetプロキシに対して送信されます。この部分はデバッグのためのものです。例えば，telnetプロキシからの応答が想定したものと異なっていた場合，telnetプロキシへ送るキャラクタ列をキーボードから入力できます。

103行目から110行目までは，selectの引数を求めるためのサブルーチンです。標準入力あるいはソケットSに対応するビットを立てます。

8ビット透過でない場合

Webプロキシの場合と異なり，telnetプロキシの種類によっては特定のキャラクタ(例えば，0x00)を通さなかったり，改行コードの変換が行われたり，特定のタイミングで特定のキャラクタが挿入されたりする場合があります。sshでは通信路が8bitキャラクタをすべて通すことを前提にしていますので，通さないキャラクタがあったり，変換や挿入が行われると，sshによる接続ができなくなってしまいます。

そこで，プロキシの種類によらず確実に通過することが期待できる英数字と一部の記号だけで通信を行う方法を考えます。アルファベット大文字と小文字で52文字，数字で10文字，これに記号文字を2つ付け加えれば64文字になり，6bitデータが表現できます。7bitを表現するには128文字必要ですが，0x80以降のキャラクタはプロキシを通過できるとは限らないので，128文字は確保できません。

1文字につき6bitのデータを送ることにすると，4文字で24bit分，すなわち3バイトのデータを送ることができます。このような変換方法はMIME*で実際に使われていて，base64エンコーディングと呼ばれています。

base64エンコーディングでは，「A」～「Z」，「a」～「z」，「0」～「9」，「+」，「-」の64文字が使われ，この順に0～63のデータを表現します。送信するデータの長さが3の倍数でない場合は，残りを「=」で埋めてエンコーディング後の文字列長が4の倍数になるようにします。例えば，「sengoku」というデータ列をbase64エ

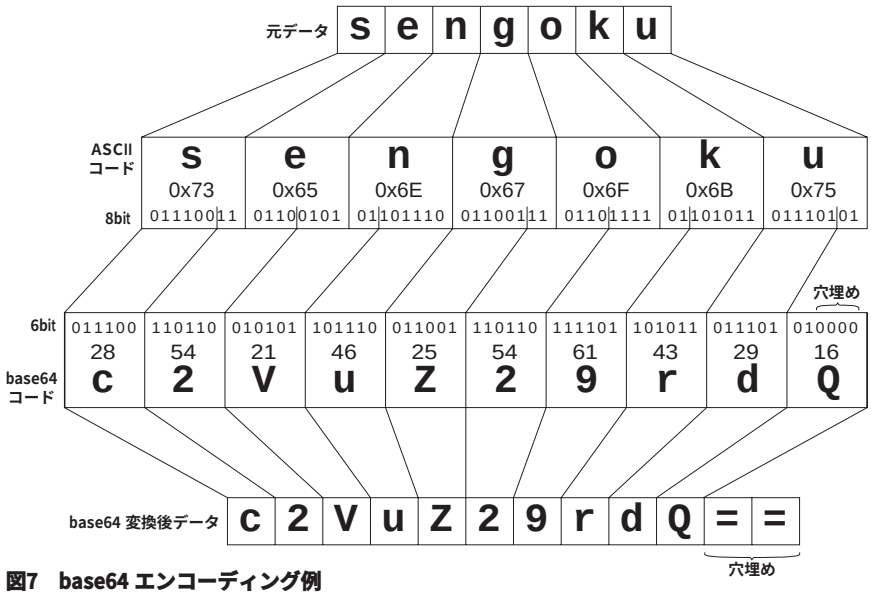

図7　base64 エンコーディング例

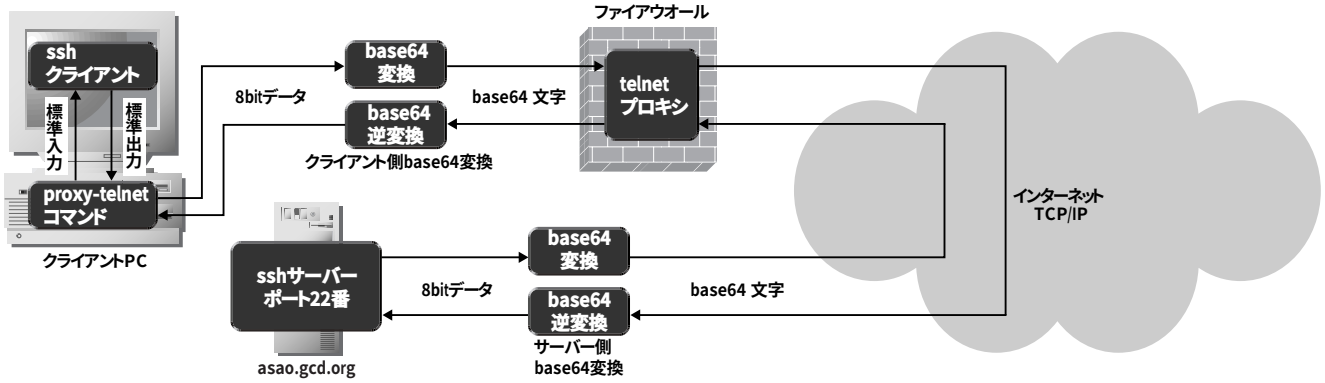

図8　8ビット透過でないtelnetプロキシ経由のssh接続

ンコーディングすると,「c2VuZ29rdQ==」というデータ列になります(**図7**)。以下,base64エンコーディングした結果のデータ列を「base64文字列」と呼びます。

base64でエンコーディングされた状態でtelnetプロキシをデータを通過させるためには,図5において,sshクライアントから送信されたデータを,base64エンコーディングで変換した上でtelnetプロキシを通過させ,sshサーバーが受信する前に逆変換を行って元のデータへ戻す必要があります。

さらに逆方向のデータ,つまりsshサーバーから送られたデータは,telnetプロキシが受信する前にbase64エンコーディングで変換し,sshクライアントが受信する前に逆変換を行って元のデータに戻さなければなりません。つまり**図8**のような構成になります。

図8から分かる通り,base64の変換および逆変換をsshクライアント側と,sshサーバー側の2カ所で行う必要があります。ただし,sshクライアント側においては,telnetプロキシの認証を行う間はbase64変換を行ってはいけません。プロキシに送信すべきユーザーIDやパスワードまでbase64変換してしまうと認証ができなくなってしまいます。

認証が終わって,sshサーバー側との接続が確立した時点(図3の(9)の段階)で変換を始める必要があります。そのためには,proxy-telnetコマンドにbase64変換機能を組み込んでしまうのが一番手っ取り早いでしょう。幸い,perlにはbase64変換のための関数が用意されていますから,組み込みは至って簡単です。ここではbase64変換機能付のproxy-telnetコマンドをproxy-baseコマンドと呼ぶことにします。

クライアント側のbase64変換

proxy-baseコマンドは,次に示すように図6のproxy-telnetコマンドに4カ所修正を加えるだけで作ることができます。

まず,1行目と2行目の間に次の行を挿入します。

```
use MIME::Base64;
```

この行は base64 変換関数を使うために必要です。

sshサーバーとの接続を完了すると,connectサブルーチン(52行目～70行目)に処理が移りますから,base64変換と逆変換は,このサブルーチン内のみで行えばOKです。

まず,loginサブルーチンの最後でsshサーバー側から受信したbase64文字列が変数Rawに設定されていますから,これを元の8bitデータ列に戻して標準出力へ出力します。そのためには,57行目を次の3行で置き換えます。

```
$_ = decode_base64 ($Raw) ;
syswrite (STDOUT,$_,length) ;
$buf = "";
```

これ以降,sshサーバー側から受信したデータ列は,62行目で標準出力に出力されますから,この62行目を図9の7行で置き換えることによって,出力する前にdecode_base64関数によって逆変換を行い,元の8bitデータ列に戻します。

一方,標準入力から入力した8bitデータ列は,97行目でサーバー側へ送信されますから,この97行目を次の2行で置き換えることによって,送信する前にencode_base64関数によってbase64文字列へ変換します。

```
$_ = encode_base64 ($_) ;
syswrite (S,$_,length) ;
```

サーバー側のbase64変換

図8のサーバー側のbase64変換は,クライアント側のbase64変換と異なり,常に変換と逆変換を行うだけで良いので

```
            print "received: $_\n" if $Verbose > 1;
            $buf .= $_;
            while ($buf =~ /=+/) {
                $buf = $';
                $_ = decode_base64("$`$&");
                syswrite(STDOUT,$_,length);
            }
```

図9　図6の62行目をこの7行で置き換える

```
stone -l localhost:22 10022/base &
```

図10　stoneを使ってサーバー側のbase64変換を行う

```
Host asao.gcd.org
        ProxyCommand    /home/sengoku/bin/proxy-base %h 10022
```

図11　クライアント側の~/.ssh/configでproxy-baseの接続先ポートを10022番に設定する

簡単です。適当な変換プログラムを書くだけで良いのですが，stone[*5]にbase64変換の機能があるので，stoneを使うことにします。sshサーバーと同じホスト上で**図10**のように実行すれば，proxy-baseコマンドからtelnetプロキシおよびインターネットを経由してポート10022番に届いたパケットをbase64逆変換してポート22番のsshサーバーに転送し，逆にsshサーバーから受信したパケットをbase64変換してproxy-baseコマンドへ返します。

この場合，proxy-baseの接続先ポートを10022番に設定する必要があるので，クライアント側の~/.ssh/configで例えば**図11**のように設定します。

ファイアウオールの外から中へ

以上は，ファイアウオールの内側からインターネット上のホストへのssh接続ですが，逆にインターネット上の任意のホストからファイアウオールの内側のホストに対する接続もまったく同様に設定できます。まず図1，図2に示したように手作業でログインを行ってみて，Proxy Commandに指定するコマンドの仕様を図3に示したように決定します。後はperlスクリプトなどでその仕様を満たすコマンドを記述するだけです。多くの場合，図6で示したスクリプトのloginサブルーチンの部分を変更するだけで済むでしょう。

多くのサイトにおいて，外部からのログインには，OTP（One Time Password）などログインごとに異なるパスワードの入力が必要だと思いますが，perlスクリプトなら認証サーバーが送ってくるチャレンジを読み取って対応するレスポンスを生成するプログラムを書くことも難しくありません。OTPのパスフレーズはssh-askpassコマンドなどを使えば，その都度ユーザーに入力させることができます。

ここで注意すべきなのは，OTPを計算するためのパスフレーズをスクリプトに埋め込んではいけない，ということです。例えそのスクリプトが個人用のPCだけに入れてあって，しかもPC自体にパスワードをかけて他の人が使えないようにしてあったとしても，その PCごと盗んでしまえばスクリプトの内容を読む方法はいくらでもあります。OTPのパスフレーズが盗まれれば，ssh接続先の自分のサーバーだけでなく，サイト全体を危険にさらすことになるわけですから，やはり避けるべきでしょう。

ssh接続のたびにOTPのパスフレーズを入力するのが面倒であれば，sshのポート・フォワード機能を利用するのが良いでしょう。例えば，ssh-askpassを呼び出してユーザーにOTPのパスフレーズを入力させ，ファイアウオールの内側のホストへssh接続するためのコマンドproxy-intoを書いたとして，これを~/.ssh/configに**図12**のように設定しているとします。**図13**のように実行すると，ssh-askpassコマンドのウィンドウが開いてパスフレーズの入力を求めるので，OTPのパスフレーズを入力すると社内LAN上のサーバー，kamiya.klab.orgへログインできます。

図13の「-L 10022:localhost:22」はポート・フォワードの設定で，sshクライアントを実行したホストの10022番ポートをsshサーバー経由でlocalhost[*6]の22番ポートへ転送します。つまり，sshクライアントを実行したホストのポート10022番に接続すると，ホストkamiya.klab.orgのポート22番へつながります。つまり，~/.ssh/configで，**図14**のように設定しておけば，図13のssh接続が続いている限り，

```
ssh kamiya
```

でOTPのパスフレーズを入力する必要なく，kamiya.klab.orgにログインできるようになります。もちろん，kamiya.klab.org上で任意のコマンドをリモート実行したり，scpを使ってファイルをコピーすることもできます。

```
Host kamiya.klab.org
        ProxyCommand    /home/sengoku/bin/proxy-into %h %p
```

図12　~/.ssh/configの設定例

```
ssh -L 10022:localhost:22 kamiya.klab.org
```

図13　ポート・フォワード機能付でファイアウオールの内側のホストへssh接続

```
Host kamiya
        HostName        localhost
        Port            10022
```

図14　~/.ssh/configの設定例

＊5　http://www.gcd.org/sengoku/stone/Welcome.ja.htmlを参照。本連載の第5回と第6回でstoneの解説を行っています。
＊6　サーバー側から見てlocalhostです。つまり，sshサーバーが動いているホストになります。