Linuxによるセキュリティ入門（1）
— ssh —

西　村　竜　一

I．はじめに

こんにちは。名古屋大学情報連携基盤センター改組の関係で，この連載もしばらく間が開いてしまいました。仕切り直しの意味を込めて，タイトルも少し変えてお送りしたいと思います。今後ともよろしくお願いいたします。

この連載では，主に大学等で計算機を道具として利用しているLinuxユーザに，セキュリティ問題に対する意識と必要なるスキルをより高めてもらうことを目標としています。ウイルスやクラッキングなどに対するセキュリティレベルの確保は，インターネットを利用するすべてのユーザが常に意識しないといけない問題です。本連載を，より安全にインターネットを利用するための参考にしていただければと思います。

また，Lunuxユーザには，自分自身でLinuxが動いている計算機を管理されている方も多いでしょう。しかしながら，その管理のために必要な時間や労力が十分に割けず，管理を怠っている方も多いのではないかと思います。そのような，なんとなく使っているけどメンテナンスがされず放置されているも同然な計算機は，クラッキングの対象となり，自分だけではなくインターネット上の他のユーザにも多大な迷惑を与えかねません。そんなことにはならないために，できるだけ楽にセキュリティレベルを保ったシステムを運用するための知恵を，プロではないシステム管理者のみなさんにお届けできればと思っています。

Linuxと言っても，数多くのディストリビューションがあり，その開発コンセプトや中身はさまざまです。すべてのディストリビューションを対象にして話をすすめていくことはできませんので，この連載の中ではDebian GNU/Linux[1] を利用することにします。Debian GNU/Linux（以下，Debianと略す）は，Debian Projectによって開発されているディストリビューションで，高性能なパッケージ管理機能を持つのが特徴です。一般的には，RedHat Linux[2] やVine Linux[3] などのLinuxディストリビューションが使われることがまだ多いようですが，Debianは除々にユーザを増やしており，ちょっとマニアックなLinuxユーザ：-)には非常に高い支持を得ています。著者も過去はいろいろ使いましたが，現在はDebianユーザであり，みなさまにも自信を持っておすすめします。初心者にはちょっとはじめの敷居が高いと言われるDebianですが，慣れれば

1　http://www.debian.org/
2　http://www.jp.redhat.com/
3　http://www.vinelinux.org/

```
## for woody
deb http://ftp.ring.gr.jp/pub/linux/debian/debian woody main contrib non-free
deb http://ftp.ring.gr.jp/pub/linux/debian/debian-non-US woody/non-US main contrib non-free
deb http://ftp.ring.gr.jp/pub/linux/debian/debian-jp woody-jp main contrib non-free
## proposal updates
deb http://ftp.ring.gr.jp/pub/linux/debian/debian woody-proposed-updates main contrib non-free
deb http://ftp.ring.gr.jp/pub/linux/debian/debian-non-US woody-proposed-updates/non-US
          (紙面の都合上改行しています。上の行とつなげて記述してください) main contrib non-free
## security updates
deb http://security.debian.org/ woody/updates main contrib non-free
```

図1　sources. list の設定例：woody 用の設定例です。この例では ftp.ring.gr.jp のミラーサーバからパッケージを取得します。近くに Debian アーカイブのミラーサーバがあればそちらに書き換えてください。

最強の道具になるのは間違いなしですので，ぜひお試しください。なお，インストール方法は，これまでの連載や市販の本などを参考にしてください。また，この連載で述べることは他のディストリビューションへも応用できることばかりなので，どうしても Debian にできない人にも参考にしていただければと思います。

それでは Debian に関するニュースからはじめます。

II．Debian GNU/Linux 3.0（woody）リリース

ニュースといってもかなり古いニュースになってしまいますが，Debian GNU/Linux の最新安定版であるバージョン 3.0 が 2002 年 7 月 19 日にリリースされました（コード名で" woody" と呼ばれることが多い）。これまでの安定版は，バージョン 2.2（" potato"）でした。もうしばらくは potato のセキュリティアップデートも提供されるようですが，今後のパッチの提供は，主に woody が対象になります。楽に計算機のセキュリティレベルを保つためには，woody へのアップグレードを強くおすすめします。potato を使い続けるには，ユーザが自分自身でセキュリティパッチを作成し，適用をする必要があります。もしくは，potato を使用する場合は，ファイヤウォールなどで外からの攻撃に対する防御を忘れないようにしましょう。

woody の新規インストールや potato からのアップグレード方法については，紙面の都合で，この連載ではとりあげることができません。リリースノート[4] やそろそろ出版されるであろうインストール本などを参考に挑戦してください（おそらくそんなに難しくありません）。

なお，以下の説明では，/etc/apt/sources. list には，図 1 に示す woody 用の設定が記述されているものとします。woody はリリースされてまだ間がないのですが，すでにいくつかのセキュリティアップデートが出されています。Debian のセキュリティアップデートは http://security.debian.org/で公開されているので，このサイトへの設定を/etc/apt/sources.list では忘れず設定するように心がけましょう。また，こまめに apt-get update と apt-get upgrade を実行して，パッケージのアップグレードを忘れないようにしてください。これまでも何度か述べましたが，

4　http://www.debian.org/releases/woody/releasenotes

Debian ではこれだけで高いセキュリティレベルを保つことができます。

III. ssh 活 用 法

今回は，連載の前回でも述べたように ssh(Secure SHell)について説明します。ssh は，telnet や rlogin，rsh などのインターネット上の他の計算機を遠隔操作するツールを置き換えるものとして開発されました。telnet や rlogin は，サーバなどの他の計算機にリモートログインする際に広く使われています。しかし，その通信は暗号化されないため，パスワードや通信内容は，他人に盗聴されてしまう危険性があります（実際，盗聴はすごく簡単なことなのです)。ssh は通信内容の暗号化や公開鍵暗号を用いた高度な認証などにより，高いセキュリティレベルを持ったリモートログインを可能とします。また，ツールとしてさまざまな便利な機能を持ち，telnet や rlogin に比べて使い勝手は良くなっています。

ssh は，すでに標準的なツールとして広く認知されています。多くの方はすでにお使いでしょうが，もしかしたら，まだお使いでない方もいるかもしれません。そんな人のために，今回は，ssh の基本的な使い方を中心に説明したいと思います。

なお，Debian で使用する ssh は，よりフリーなライセンスで提供されている OpenSSH[5] ですが，基本的にはオリジナルの ssh と同じ使い方ができるので，以下の説明では区別なく記述することとします。

1．ssh のインストール

/etc/apt/sources.list の設定が正しければ，Debian パッケージは，apt-get で簡単にインストールすることができます。ssh の Debian パッケージの名前は ssh なので，ssh のインストールは以下の操作で行うことができます。

```
# apt-get install ssh
```

ここで#は，root のシェルのプロンプトをあらわしており，この操作の意味は，root 権限で apt-get install ssh というコマンドを実行するという意味です。同様に，一般ユーザ権限でコマンドを実行するときは，%でシェルプロンプトをあらわすこととします。

上記コマンドでインストールを実行すると，apt は，/etc/apt/sources.list で設定された Debian アーカイブからパッケージのファイルを取得し，システムへのインストールを開始します。すると，設定にもよりますが，システムはいくつかのインストールに関する質問をしてきます。今回，実際に woody のシステムに apt を用いて ssh をインストールしたところ，聞かれた質問は以下の 4 つでした。これらの質問は設定によっては聞かれない場合もあります。そのような場合にはあらかじめ定められたデフォルトの設定が利用されます。それでは質問の中身を見てみましょう。

5 http://www.openssh.org/ja/index.html

・Do you want to continue?

言うまでもなく，「インストールを続行するか？」という意味です。もちろんYを選んで続行します。この際，新たにパッケージをインストールすることによって，依存関係のためインストールされる他のパッケージ，削除されてしまうパッケージが表示されます。パッケージのインストールをするときには，それらを確認するようにしましょう。

・Allow SSH protocol 2 only

ssh の通信プロトコルには，大きくわけてバージョン1（SSH 1 プロトコル）とバージョン2（SSH 2 プロトコル）の2つのプロトコルがあります。実は，SSH 1 プロトコルには安全性に欠陥があることが判明しています。SSH 2 プロトコルでは，その欠陥の本質的な修正と特許問題の解決及びプロトコルの見直しが行われています。しかし，そのSSH 1 プロトコルの欠陥もさまざまな改良により，現在ではおおよそ問題ではなくなってきています。特許問題も現在は特許が切れているので問題にはなりません。また，SSH 1 プロトコルしかサポートされていない環境も依然として多くありますので，ここではSSH 1 プロトコルも使えるように設定したいと思います。この場合には，〈No〉を選択してください。より安全なSSH 2 プロトコルのみしか使わない場合は，〈Yes〉を選びます。たとえば，クライアントとサーバがどちらもDebianのみであるならSSH 2 プロトコルのみの利用で問題がなく，より安全な通信を行うことができます。

・Do you want /usr/lib/ssh-keysign to be installed SUID root?

ここでの詳しい説明は省略しますが，“If in doubt, I suggest install it with SUID.”らしいので，〈Yes〉を選択しましょう。

・Do you want to run the sshd server?

ssh をクライアントだけではなくサーバとしても起動する（sshd を利用する）場合には，〈Yes〉を選択します。

これらのパッケージのインストール時に聞かれる設定をインストール後に変更するには dpkg-reconfigure コマンドを用います。たとえば以下のようになります。

```
# dpkg-reconfigure -plow ssh
```

ただし，この方法ではさきほどの”Allow SSH protocol 2 only”に関しては変更することはできません。このようなdpkg-reconfigureで変更できない設定に関しては，直接エディタで設定ファイルを書き換え，デーモンを再起動することになります。sshd の設定ファイルは，/etc/ssh/sshd_config です。このファイル中の Protocol の行を”Protocol 2，1”にすればSSH 2 とSSH 1 両方，“Protocol 2”に変更すればSSH 2 のみになります。書き換えができたら，

```
The authenticity of host 'docchi (192.168.0.1)' can't be established.
RSA key fingerprint is 2f:ea:5f:ad:b2:67:87:ae:5a:54:6a:c6:87:c1:8b:d8.
Are you sure you want to continue connecting (yes/no)?
```

図2　はじめてdocchiという名前の計算機にsshでアクセスしたとき

```
# /etc/init.d/ssh restart
```

でsshdを再起動して設定の変更は完了です。

ちなみに，sshをアンインストール（削除）するには，

```
# apt-get remove ssh
```

とします。

2．sshの基本的な使い方

sshの最も基本的な使い道は，遠隔の計算機へのリモートログインです。たとえば，acchi.example.ac.jpという名前の計算機にhanakoという名前でログインするときは以下のようになります。

```
% ssh acchi.example.ac.jp -l hanako
```

このとき，もちろんacchi.example.ac.jpではsshdが動いてなければなりません。また，-lによるユーザ名の指定のオプションは，ログイン元の計算機とログイン先の計算機のユーザ名が同じときは省略することができます。はじめてログインする計算機に接続したときには，図2のようなメッセージが表示されますので，"yes"と答えます。このとき，ログイン元の計算機へは，ログイン先の計算機の公開鍵が渡されます。渡された公開鍵は記録され，次回，接続するときに，この記録された公開鍵と接続時のログイン先計算機の秘密鍵を用いて，接続相手は前回と同じ計算機かのチェックを行います。これをホスト認証と呼びます。もしログイン先の計算機が他のものに変わっていて，鍵の組合せが正しくなければ，"WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!"と警告が表示され，ログインすることはできません。この場合，ログイン先の計算機は偽装されている可能性があるので注意が必要です。sshは，このような仕組みでDNSの不正改竄などによる悪意を持ったホストのなりすましに対処しています。

ただし，OSの再インストールなどでログイン先の計算機の鍵が本当に変更されてしまった場合も，この警告によりログインすることはできません。この場合は，ログイン元の計算機のホー

ムディレクトリの下の.ssh/known_hostsというファイルにログイン先の計算機の鍵が記録されているので，該当の計算機の行を削除することによって再びログインすることができるようになります。

ホスト認証が無事に終了すると，以下のようにパスワードが求められます。ここでUNIXのログインパスワードを正しく入力すれば，ログインは完了です。

```
hanako@acchi.example.ac.jp's password:
```

このパスワードを入力する時点で，すでに通信は暗号化されているので安全です。

さて，sshには便利な機能があると述べましたが，まずは，通信内容の圧縮機能を紹介しましょう。使い方は簡単で，sshの起動時にオプションで-Cをつけるだけです。たとえば，前述の例では，

```
% ssh -C acchi.example.ac.jp -l hanako
```

のようになります。圧縮処理が必要なので多少のCPUパワーは奪われてしまいますが，遅いネットワークを経由してリモート操作する場合などには有効です。お試しください。ただし，速いネットワークでは逆に遅くなることがあるそうです。

3．鍵を作ろう

さきほどの説明では，ホスト認証が終了したのち，ログインパスワードを使ってユーザの認証を行いました。この場合でも，sshはホスト認証や暗号化された通信などによって，telnetやrloginと比較して，非常に高いセキュリティレベルを保った通信を行うことができます。そして，より安全なユーザ認証として公開鍵暗号による認証を利用することもできます。公開鍵暗号によるユーザの認証を利用すると以下のような利点が考えられます。

- 秘密鍵の管理が完全なら，第3者になりすまし認証をされる可能性が限りなく少ない。
- 通常のUNIXパスワードよりも長くて複雑なパスフレーズを使用できる。
- ネットワーク上にパスワードやパスフレーズを一切流さない。

公開鍵暗号による認証では，秘密鍵と公開鍵の2つの鍵（実際はある長さを持ったテキスト）を1つの組にして使用します。秘密鍵は文字通り絶対に秘密にしなくてはならない鍵です。絶対に，だれかに見せたり，コピーして渡したりしてはいけません。公開鍵は逆に公開されている鍵ですから，だれに見せても構いません。sshのユーザ認証で使用する際には，秘密鍵をログイン元の計算機の中のだれにも見られてはいけないファイルに，公開鍵をログイン先の計算機のファイルに保存しておきます。そして，それら鍵を生成するプログラムがssh-keygenコマンドです。

ところで，SSH 1プロトコルとSSH 2プロトコルでは使用する鍵の種類が異なり，保存するファイル名も異なっています。SSH 1プロトコルではRSA鍵，SSH 2プロトコルではRSAもし

くはDSA鍵を用います。表1に生成する鍵とそれぞれの鍵ファイルのファイル名を示します。この表の中で$HONEは，ホームディレクトリをあらわしています。

表1　生成される鍵ファイル

プロトコル	公開鍵のファイル名	秘密鍵のファイル名
SSH1	$HOME/.ssh/identity.pub	$HOME/.ssh/identity
SSH2(DSA鍵)	$HOME/.ssh/id_dsa.pub	$HOME/.ssh/id_dsa
SSH2(RSA鍵)	$HOME/.ssh/id_rsa.pub	$HOME/.ssh/id_rsa

それでは鍵を生成してみましょう。以下では，SSH 1プロトコルのためのRSA鍵の生成を例にとって説明します。

```
% ssh-keygen -t rsa1
```

ssh-keygenを実行すると，まず”Enter file in which to save the key”と入力を求められます。ここに鍵を出力するファイル名を入力するのですが，今回はデフォルトのファイル名を使用しますので，そのままエンターキーを押してください。

続いて求められるのがパスフレーズの入力です(“Enter passphrase”)。このパスフレーズは一種のパスワードのようなものですが，(ログイン元にある)秘密鍵によるメッセージの復号のために利用します。このため，sshのユーザ認証のときに，パスフレーズ自体がネットワークに流れることはなく，非常に安全性の高い認証を可能にしています。また，パスフレーズの長さは，UNIXのログインパスワードより長いものを設定できるので，より安全であるといえます。ちなみに，良いパスフレーズは，10～30文字程度の長さを持つ，第3者が容易に想像しにくい文字列です。なお，パスフレーズはさきほども述べたように秘密鍵と一緒に利用されます。秘密鍵とパスフレーズのどちらかがなくては認証を行うことはできません(もちろん公開鍵も使いますが…)。このため秘密鍵の管理さえしっかりしていれば，0文字のパスフレーズを用いることも可能です。一般的には短いパスフレーズは危険ですが，場合によっては便利なときもあります。ただし，なんども言いますが，秘密鍵は絶対に秘密でなければなりません。

“Enter passphrase”でパスフレーズを入力すると，もう一度，確認のために同じパスフレーズの入力が求められます(“Enter same passphrase again”)。正しくパスフレーズの入力ができると，$HONE/.ssh/identity.pubに公開鍵，$HOME/.ssh/identityに秘密鍵が出力されます。

同様に，SSH 2プロトコルのためにRSA鍵の生成は，

```
% ssh-keygen -t rsa
```

SSH 2プロトコルのためのDSA鍵の生成は，

```
% cat $HOME/.ssh/identity.pub |ssh acchi.example.ac.jp 'cat >> $HOME/.ssh/authorized_keys'
```

図3　ssh経由のパイプを応用した公開鍵の登録方法：ただし，ログイン先の計算機でsshのUNIXログインパスワードでのユーザ認証を許可している必要があります。

```
% ssh-keygen -t dsa
```

と実行します。

それでは生成した鍵を用いたユーザ認証をしてみましょう。そのための準備として少々やっかいなのは，公開鍵をログイン先の計算機に登録しなければならないということです。

その具体的な操作は，ログイン元計算機にある公開鍵ファイル(identity.pubやid_dsa.pub)の内容をログイン先の計算機の$HOME/.ssh/authorized_keysに追加することになります。$HOME/.ssh/authorized_keysへの追加はテキストエディタを使って，公開鍵ファイルの内容をコピーアンドペーストしても良いですし，少し高度なテクニックとしては，図3に示すような操作で行うこともできます。この操作は，ログイン先であるacchi.example.ac.jpのホームディレクトリ下の.ssh/authorized_keysに，ログイン元の$HOME/.ssh/identity.pubの内容をssh経由のパイプを用いて追加しています。

SSH 2プロトコルを用いるときも，同様の手順でログイン先のauthorized_keysファイルに，使用する公開鍵を追加してください。$HOME/.ssh/authorized_keysの内容は，1行ごとに1つの公開鍵である必要があります。複数の公開鍵を行を連結して記述したり，余計な改行が鍵の途中で含まれてはいけません。

また，よりセキュリティの厳しい設定の計算機では，sshの公開鍵ユーザ認証でのみログインを許可している場合があります。このときは，公開鍵を登録するためにログインすることができませんので，その計算機の管理者に公開鍵ファイルをメールなどで送付して，鍵を登録してもらうことになります。

準備ができたところで，公開鍵によるユーザ認証を用いたリモートログインをしてみましょう。と，言っても普通にsshを使うだけです(図4を参照)。違うのは，認証時にUNIXのログインパスワードではなく，鍵の生成時に入力したパスフレーズの入力を求められることです。正しいパスフレーズを入力すればログインは完了です。また，Debianのデフォルト設定では，何度かパスフレーズを間違えた場合は，UNIXログインパスワード認証に切り替わるようになっています。

ただし，Debianの標準の設定では，sshはSSH 2プロトコルを優先して使います。SSH 1プロトコルを利用するときには，sshの実行時てに，-1オプションをつけて使用してください。たとえば，以下のようになります。

```
% ssh acchi.example.ac.jp -l hanako
Enter passphrase for key '/home/hanako/.ssh/id_dsa':
（ここで正しいパスフレーズを入力）

（パスフレーズを間違えて入力すると…）
hanako@acchi.example.ac.jp's password:
（のようにUNIXパスワード認証に切り替わります）
```

図4　公開鍵ユーザ認証の時のssh実行画面：この例ではSSH 2プロトコルのDSA鍵を用いています。

```
[例1] % scp ./test.txt hanako@acchi.example.ac.jp:/tmp/test2.txt
[例2] % scp hanako@acchi.example.ac.jp:public_html/index.html .
[例3] % sftp hanako@acchi.example.ac.jp
[例4] % scp -r hanako@acchi.example.ac.jp:public_html .
[例5] % rsync -auvzb -e ssh ~/public_html hanako@acchi.example.ac.jp:
```

図5　scp，sftp，rsyncの使用例

```
% ssh -1 -C  acchi.example.ac.jp
```

4．ファイルのコピー

ここでは，sshのプロトコルを用いたネットワーク経由の安全なファイル転送機能について説明します。ファイルのコピーをするためのコマンドは，scpです。まずは図5の例1を見てください。これはカレントディレクトリのtest.txtというファイルをacchi.example.ac.jp上の/tmp/test2.txtにコピーする操作です。scpは基本的には，

```
% scp [コピー元] [コピー先]
```

のように使います。コピー元，コピー先の指定は，“ユーザ名@ホスト名：ファイルのパス”となります。“ユーザ名”や“ホスト名”は，ローカルの計算機に対しては省略します。2つのリモートホスト間でファイルをコピーすることもできます。また，“ファイルのパス”は，絶対パスで書くか，もしくは“ユーザ名@ホスト名：”に続けて書いた場合は，ホームディレクトリからの相対パスになります。たとえば，acchi.example.ac.jpのホームディレクトリ下にあるpublic_html/index.htmlをローカル計算機のカレントディレクトリへコピーする操作は，図5の例2のように

なります。scpは，sshと同様に-Cや-1のオプションをつけることができます。

scpの他にftpに似たインタフェースでネットワーク経由のファイルコピーを提供するツールがあります。それがsftpです。sftpは，図5の例3のように起動し，リモート計算機に接続後は，普通のftpと同様にgetやputのコマンドを使ってファイルの転送を行うことができます。mgetにより複数のファイルをまとめてコピーすることもできます。ftpのコマンドの使い方の詳細は，ftpもしくはsftpのオンラインマニュアルを参照してください。sftpは，ftpに似ていますが，あくまでsshに含まれるツールなので，リモート計算機では，ftpサーバではなくsshdが動いていなければなりません。

また，ディレクトリ単位のコピーは，scpにディレクトリ全体を再帰的にコピーするオプション-rをつけて行います。実行例は，図5例4のようになります。そして，rsync[6]とsshと組合わせて使うことにより，さらに強力なディレクトリ単位のコピーができます。rsyncは，大量のファイルを高速かつ差分のみを効率よく転送することがてきるツールです。それでは，おなじみのapt-getを使ってrsyncのインストールをしましょう。

```
# apt-get install rsync
```

なお，ファイル転送をする送信側，受信側の両方の計算機にrsyncをインストールする必要があります。

rsyncも多くのオプションを持ったソフトウェアなので，詳細はオンラインマニュアルを読んでいただくことにして，ここでは実際の使用例を紹介します。図5の例5を見てください。この例では，ローカルの計算機のホームディレクトリの下にあるpublic_htmlディレクトリの下をacchi.example.ac.jp上のユーザhanakoのホームディレクトリの下にコピーします。コピー元，コピー先の指定はscpと同様です。オプションauvzbの意味は以下のようになります。

-a　アーカイブモードで転送。ファイルの持つ情報を保つ。
-u　コピー元よりコピー先のファイルの方が新しい時はスキップ。
-v　転送の間の情報を詳しく表示。
-z　通信を圧縮して転送。
-b　コピー先にコピー元と同名のファイルがあるときはバックアップを作成。

また，-e sshオプションによって，sshを使った接続を利用するよう指定しています。public_htmlディレクトリの下のディレクトリも再帰的にコピーするには，さらに-rオプションを指定してください。

6　http://rsync.samba.org/

```
% telnet localhost 10110
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK <17395.1030293525@mail.example.ac.jp>
```

図6　sshのポート転送を用いた安全なPOP 3へのアクセス

5．ポート転送の利用

sshの便利な機能として重要なのがポート転送機能です。sshは，TCP/IP通信で使われるある特定のポートへのアクセスを，暗号化された安全な通信によって，他の計算機の任意のポートに転送することができます。

たとえば，メールを受信するときに使用するPOP 3プロトコルでの利用を考えましょう。POP 3は，そのままでは通信内容やパスワードの暗号化などは行いません。よって，メールの内容やパスワードが第3者によって盗聴されてしまう可能性があります。このままでは危険なので，sshを用いてローカル計算機の10110番ポートをメールサーバの110番に転送します。110番は，POP 3が使用すると定義されたポート番号です。一方で，ローカル計算機側の10110番は適当に決めた数字です。このとき，メールソフトはローカルの計算機の10110番ポートへ接続して，メールを受信するように設定します。しかし，実際にはsshによって通信は転送されて，メールサーバのPOP 3サーバにアクセスすることになります。この結果，ローカルの計算機とメールサーバ間の通信の安全性は格段に向上します。

それでは実際にポート転送を行ってみましょう。以下の例では，ローカルの計算機の10110番ポートをメールサーバであるmail.example.ac.jpのPOP 3（110番）ポートに転送します。ただし，mail.example.ac.jpにsshでログインできるアカウントが必要です。

```
% ssh -L 10110:mail.example.ac.jp:110 mail.example.ac.jp -l hanako
```

続いて認証が行われ，一見，普通にログインしたように見えますが，これでポート転送は完成です。

ためしにtelnetコマンドでローカル計算機の10110番ポートにアクセスします(図6参照)。このようにmail.example.ac.jpのPOP 3サーバにアクセスできていることがわかります。

```
telnet stream tcp nowait telnetd.telnetd /usr/sbin/tcpd  /usr/sbin/in.telnetd
↓ この行を以下のように変更（-z secureを追加）
telnet stream tcp nowait telnetd.telnetd /usr/sbin/tcpd  /usr/sbin/in.telnetd -z secure
```

図7　/etc/inetd.conf内のtelnetdの設定変更

なお，この例ではローカル計算機の10110番ポートへのアクセスはlocalhostからのみ許されます。このポートへの他の計算機からのアクセスも許可するには，sshの実行時に-gオプションを指定します。また，ネットワーク環境によっては-Cも使うのが良いと思います。

sshのポート転送機能はたいへん強力で，これ以外にもさまざまな応用が考えられます。今回は簡単な例の紹介程度しかできませんでしたが，オンラインマニュアルなどを参考に，ぜひ他の応用を試してみてください。

IV. おまけ：SSL対応telnet

ここまでsshについて説明をしましたが，みなさんの中には使いなれたtelnetコマンドを使いたいという方もいるのではないでしょうか。そんな方にもsshへの移行をおすすめしますが，Debianでは通信の暗号化をサポートしたtelnetを利用することもできますので，おまけに紹介します。

telnetの暗号化は，SSL[7]（Secure Sockets Layer）という技術を利用します。SSLはsshと同様なホスト認証や通信の暗号化により，通信の安全を確保する技術であり，telnet以外にもWebのHTTPやメールのPOPなどのプロトコルを安全なものにするのに広く使われています。SSLについては，また機会があればこの連載でとりあげようと思いますが，今回はtelnetに関してのみ説明をします。

SSLを用いたtelnetを利用するには，サーバ，クライアント両方がSSLに対応している必要があります。Debianでは，SSL対応telnetサーバ（telnetd）のインストールは，

```
# apt-get install telnetd-ssl
```

そして，クライアントのインストールは，

```
# apt-get install telnet-ssl
```

で行うことができます。telnetdは，リモートからログインする必要がある計算機のみにインストールするようにしましょう。余分なサーバプログラムはインストールしてはいけません。

7 http://www.openssl.org/

```
% telnet acchi.example.ac.jp
Trying 192.168.1.100...
Connected to acchi.example.ac.jp.
Escape character is '^]'.
[SSL - attempting to switch on SSL]
[SSL - handshake starting]
[SSL - OK]
Password:
```

図8　SSL対応telnetによる安全なtelnet接続

インストールが完了したら，普通にtelnetを使って計算機にログインしましょう。図8のように表示されれば，SSLによって安全なtelnetが利用できています。

ただし，気をつけなければいけないことは，標準設定において，このtelnetdはSSL非対応のtelnetクライアントからのアクセスも受け付けてしまうということです。これでは危険極まりないので，SSL非対応のtelnetのアクセスは拒否するよう変更します。まず，/etc/inetd.confの中のtelnetdに関する設定を図7のように変更します。続いて以下のようにしてinetdを再起動します。

```
# /etc/init.d/inetd restart
```

これで設定が有効になっているはずです。

試しにSSL非対応のtelnetを用いてログインしようとすると，"telnetd: [SSL required - connection rejected]."とログインが拒否されることがわかります。

V. おわりに

sshの話，いかがでしたでしょうか？　もし，まだsshを使わずリモートログインをしている人がいたら，この機会に乗り換えることを強くおすすめします。いや，心情的には「乗り換えろ！」と命令したいです。認証や公開鍵暗号の仕組みなどは少し複雑ですが，まず大事なのはツールとして使うことです。慣れるまで根気よく使ってみてください。仕組みや動作原理を詳しく知りたい方は，詳細な解説のある本を読むのも良いですが，OpenSSHもオープンソースなソフトウェアなので直接ソースコードを読んでみるのはいかがでしょうか（と言っている私は読んだことありませんが）。

さて，次回はやっとLinuxらしい話題で，iptablesなどを用いたアクセス制御の方法について説明したいと思います。

（にしむら　りゅういち：奈良先端科学技術大学院大学情報科学研究科）
（nisimura@linux.or.jp）