# An Introduction to Secure Shell for Non-Server Engineers

*——非サーバーエンジニアのためのSSH入門*

2014/01/18
姫路IT系勉強会
Hajime MIZUNO @ Ubuntu Japanese Team

NRKbeta 2008

# Self Introduction

*——お前誰よ？*

- 水野 源 (Hajime MIZUNO)
- @mizuno_as
- Ubuntu Japanese Team所属
- Ubuntu Member
- Debian Package Maintainer
- 本業はサーバー/インフラエンジニア
- 書籍執筆など少々

ubuntu

# 今日のお題

Secure Shell = SSH

# What is Secure Shell?

*——SSHってなんだろう?*

- セキュアに通信するためのプロトコル
- サーバーを安全に操作したり
- 安全にファイルを転送したり
- さまざまな通信のバックエンドにも

サーバーにログインするための
ツール

というのが一般的な認識

# Install OpenSSH Server

*——SSHサーバーを構築する*

## Ubuntu or Debian

apt-getでopenssh-serverをインストールするだけ。
taskが用意されているので、taskselでインストールしても可。

```
$ sudo apt-get install openssh-server
$ sudo tasksel install openssh-server
```

## RHEL or CentOS

インストールはyumで。
パッケージ名はDebianと異なる。

```
# yum install openssh
```

# Let's get the show on the road

*——はじめの一歩*

## サーバー(hostname)にログインする

ユーザー名は-lオプションか、メールアドレスのように@をつけて表記する。
省略すると、ローカルホストの現在のユーザー名が使われる。

*$ ssh -l name hostname*
*$ ssh name@hostname*

## SSH越しにコマンドを投入する

ホスト名の後にコマンドを書くことで、任意のコマンドを投入できる。
この場合対話的シェルには入らず、実行後にセッションが切れる。

*$ ssh -l name hostname command*

# Let's get the show on the road

*——はじめの一歩*

SSHクライアントでサーバーに接続


SSH Client

Secure Channel

SSH Server

シェルはサーバー上で実行される

- zsh
- tcsh
- bash

# File copy over Secure Channel

*——安全なファイルコピーのために*

## セキュアなcp ーー scp

ネットワーク越しに使えるcpコマンド。
ユーザー名は省略可能。
ファイルパスの”/”を省略すると、ホームディレクトリからの相対パスになる。

*$ scp name1@host1:file1 name2@host2:file2*

## SSHごしのftp ーー sftp

ftpコマンドと同様に、対話的なput/getが可能。
例によってユーザー名は省略可能。

*$ sftp name@host*

# Dig! Dig! the Tunnel

*——奥の奥まで*

- SSHはリモートログインするだけのツールではない
  → SSHは安全な経路を作るためのプロトコル
- SSHセッションの上に、別のプロトコルを流せる
- 本番の通信に先立ち、SSHで経路を確保する
  → トンネルを掘る

# Local Forwarding

*——ローカルポート転送の例*

**sshコマンドがポートを待ち受け、サーバーが転送する**

*$ ssh -L port:host:hostport hostname*

sshコマンドが、クライアントホスト上でportに指定したポートを待ち受ける。
SSHサーバーが、その通信をhostに指定したホストのhostportへ転送する。

転送先のhostは、SSHサーバーから見たホスト。

**例:**

*$ ssh -L 8080:localhost:80 192.168.1.1*

1. sshコマンドを実行するマシンでポート8080を待ち受ける。
2. SSHのセッションを経由して、その通信が192.168.1.1へ送られる。
3. 192.168.1.1がlocalhost(192.168.1.1自身）のポート80へ転送する。

トンネルを掘るだけなら-f -Nとかが便利。
専用のフロントエンドもある。

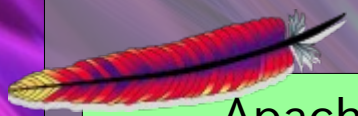
# Local Forwarding

*——ローカルポート転送の例*

SSH Client

SSH Server



Apache Web Server

127.0.0.1:80からのみ
アクセス可能

2. Localhost:8080へ接続

4. Localhost:80へ転送

127.0.0.1:8080
をLISTEN

3. SSHサーバーへ転送

SSH Server

(192.168.1.1)

sshコマンド

1. SSHセッションの開始

*$ ssh -L 8080:localhost:80 192.168.1.1*

# Local Forwarding

*——ローカルポート転送の例*

- 何の役に立つの？
  - クライアントから直接アクセスできないホストへ到達できる。
  - LAN内部のサーバーとか。
  - アクセス制限の回避にも。

# Remote Forwarding

*——リモートポート転送の例*

**SSHサーバー上でポートを待ち受け、SSHクライアントへ転送する**

*$ ssh -R port:host:hostport hostname*

接続したSSHサーバー上で、portに指定したポートを待ち受ける。
サーバー上のそのポートへの接続は、SSHクライアントへ送られる。
SSHクライアントが、hostのhostportへ通信を転送する。

転送先のhostは、SSHクライアントから見たホスト。
基本的にローカルポート転送の逆。

**例:**

*$ ssh -R 8080:localhost:80 192.168.1.1*

1. SSHサーバー(192.168.1.1)上で、ポート8080を待ち受ける。
2. SSHのセッションを経由して、その通信がSSHクライアントへ送られる。
3. SSHクライアントがlocalhost(SSHクライアント自身）のポート80へ転送する。

# Remote Forwarding

*——リモートポート転送の例*

SSH Client

SSH Server

1. SSHセッションの開始

sshコマンド

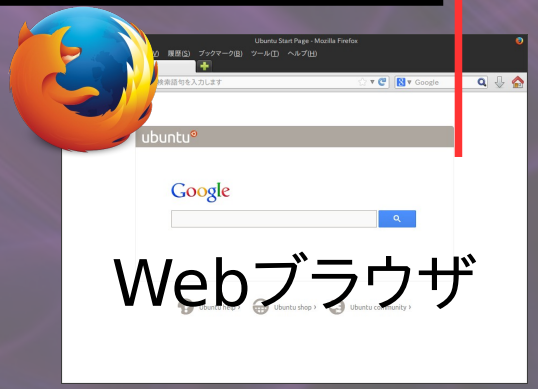
SSH Server

(192.168.1.1)

3. SSHクライアントへ転送

127.0.0.1:8080
をLISTEN

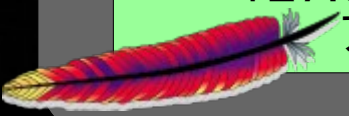
2. Localhost:8080へ接続




4. Localhost:80へ転送

Apache Web Server

127.0.0.1:80からのみ
アクセス可能



*$ ssh -R 8080:localhost:80 192.168.1.1*

# Remote Forwarding
*——リモートポート転送の例*

- 何の役に立つの？
  - 外(SSHサーバー)から内(SSHクライアント)へ通信を開始できる。
  - SSHのセッション自体は、SSHクライアントからはじまるところがポイント。
  - つまりファイアウォールを無視して、外部から内部への通信が可能。
  - 一時的にLAN内のサーバーを表に出したいような時に

# Dynamic Forwarding

*——あるいはSocks Proxy*

**SSHサーバーをSocks Proxyとして利用できる**

*$ ssh -D port hostname*

sshコマンドが、クライアントホスト上でportに指定したポートを待ち受ける。
そのポートを、Socks Proxyとして使用できる。
このポートへの接続は、すべてSSHサーバーを経由して送信される。

**例:**

*$ ssh -D 8080 192.168.1.1*

1. SSHクライアント上で、ポート8080を待ち受ける。
2. アプリケーションで、Socks Proxyにlocalhost:8080を指定する。
3. そのアプリケーションからの通信は、SSHサーバーを経由して送信される。

# Dynamic Forwarding

*——あるいはSocks Proxy*

SSH Client




2. Socks ProxyをLocalhost:8080に設定

3. www.example.comへ接続

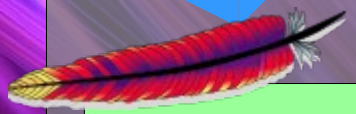
127.0.0.1:8080
をLISTEN

sshコマンド

4. SSHサーバーへ転送

1. SSHセッションの開始

Web Server



www.example.com

5. www.example.comへ
192.168.1.1から接続

SSH Server

SSH Server

(192.168.1.1)

*$ ssh -D 8080 192.168.1.1*

# Dynamic Forwarding

*——あるいはSocks Proxy*

- 何の役に立つの？
  - ポート転送と違い、アプリケーション単位で転送できる。
  - 直接アクセスできないサーバーにアクセスする際の踏み台に。
  - 検証などで、外部からのアクセスを行う必要がある時とか。
  - 日本からアクセスできないWebサイトを見る時に、AWSにSSHサーバーを立てたり。

# Forwarding X Protocol

*——VNCが許されるのは小学生までよね*

## SSHセッションを経由してXプロトコルを転送する

*$ ssh -X hostname*

X Window Systemは、クライアント・サーバーモデル。
Xクライアント(GUIアプリ)を、Xサーバーによって描画している。
SSHサーバー上のXクライアントを、SSHクライアント上のXサーバーで描画できる。
SSHだけで(アプリ単位の)リモートデスクトップが実現できる。
しかも軽いし、Linuxなら特別なアプリは不要。

## 例:

*$ ssh -X 192.168.1.1*
*$ firefox*

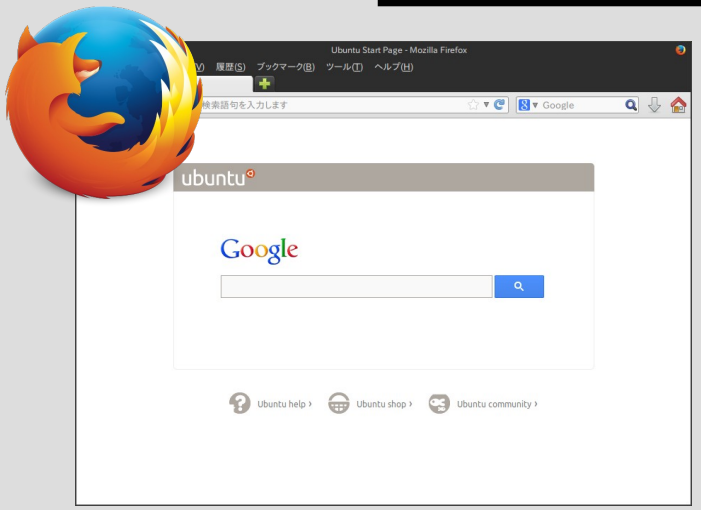
1. SSHサーバーにログインし、Firefoxを実行する。
2. SSHサーバーで起動したFirefoxが、SSHクライアントの画面に描画される。

# Forwarding X Protocol

*——VNCが許されるのは小学生までよね*

SSH Client

4. SSHサーバー上で動作しているアプリケーションが SSHクライアントのデスクトップに表示される




SSH Server

2. Firefoxを起動

3. 描画の命令を転送

Xサーバー

Firefox

sshコマンド

SSH Server

1. SSHセッションの開始

*$ ssh -X 192.168.1.1*

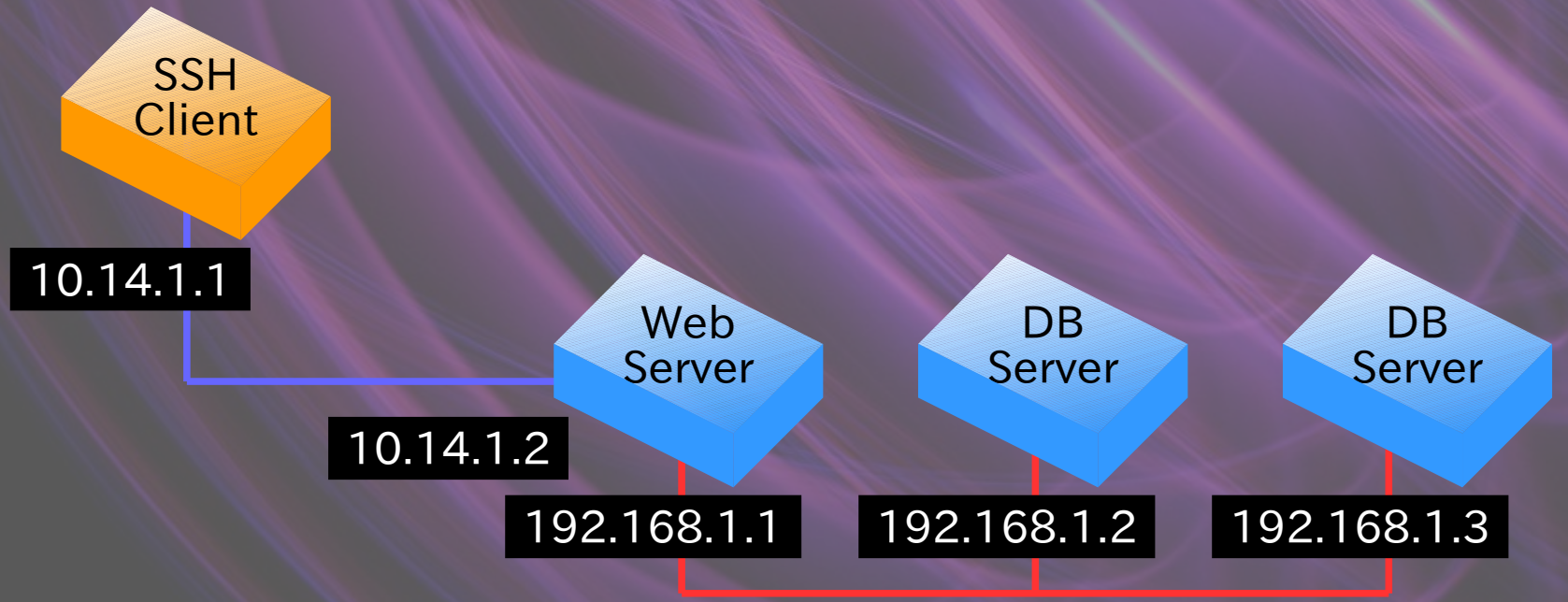
# I want to go forever

*——踏み台を意識しない接続*

## 直接アクセスできないローカルサーバーに接続したい

1. Webサーバー(10.14.1.2)にSSHログインする。
2. そこからさらにsshコマンドを実行し、DBサーバー(192.168.1.x)へ接続する。

何回もsshコマンドを打つのは面倒くさい!! scpとか死ねる!!



# I want to go forever

*——踏み台を意識しない接続*

## ProxyCommandで自動的に踏み台を経由する

ProxyCommandは「サーバに接続するために使用するコマンド」を設定する。
~/.ssh/configには、SSHクライアントの設定を書ける。
ここにホストごとの接続設定を書けばいい。

**例：**

*Host db1.example.com*
*ProxyCommand ssh web.example.com nc %h %p*

*$ ssh db1.example.com*

1. db1.example.comへ接続しようとすると……
2. ProxyCommandによってweb.example.comにSSH接続され……
3. web.e.c上でncコマンドが実行され、db1に通信を中継する。

なお最近は-Wオプションが実装されており、ncを併用しなくてもよかったりする。

# More Secure!

*——公開鍵認証のススメ*

## パスワード認証は、ブルートフォース攻撃に弱い

力技で攻撃されると、パスワードが「偶然一致してしまう」可能性も否定できない。
そこでより強力な、公開鍵認証を利用しよう。

## 鍵ペアの生成

*$ ssh-keygen -b 4096 -C comment*

## putty形式の鍵の変換

*$ ssh-keygen -i -f pkkfile*

## フィンガープリントの確認

*$ ssh-keygen -l -f id_rsa.pub*

# Copy and Import
## ――公開鍵のカンタンデリバリー

### 公開鍵をサーバーに配置しよう

id_rsa.pubの中身を、~/.ssh/authorized_keysに追記する。
エディタで開いてコピペして……めんどくさい。

ちなみに~/.ssh以下は、他人が読み書きできないパーミッションに設定しよう。

### クライアントから鍵をpushする

*$ ssh-copy-id -i id_rsa.pub hostname*

鍵配置前なので、別の鍵かパスワードでログインできる必要あり。

### Launchpadから鍵をインポートする

*$ ssh-import-id username*

コマンド一発で公開鍵を引っぱってこれるので、管理者にオススメ!

# Use the Multiple Keys

*——鍵の使い回しにご用心*

## 複数の鍵を使いわけよう

複数のサーバーで、同じ鍵を使い回すのはリスクがある。
ssh-keygenで鍵は何個でも作れる。
ssh -iで使用する鍵を選択できるけど、面倒くさい！

## 使う鍵を設定するには

デフォルトで使われる鍵は~/.ssh/id_rsa
~/.ssh/configに、使う鍵を列挙できる。

## 例:

*IdentityFile ~/.ssh/id_rsa*
*IdentityFile ~/.ssh/id_rsa.hoge*
*IdentityFile ~/.ssh/id_rsa.fuga*

# Jones, Brown and Smith

*——交渉の際には代理人を*

## パスフレーズの入力面倒くさい！

秘密鍵は漏洩に備えてパスフレーズを設定するのが基本。
使うたびにパスフレーズを入力するのは面倒！
そこで、鍵をキャッシュしてくれるssh-agent。

## 秘密鍵の配置問題

踏み台を経由して多段接続する場合、踏み台に秘密鍵を置く必要がある。
これは不用心！秘密鍵は手元にだけ置きたい！
エージェント転送で解決！

# Jones, Brown and Smith

*——交渉の際には代理人を*

**エージェントを起動する**

*$ eval $(ssh-agent) && ssh-add ~/.ssh/id_rsa*

**キャッシュされた鍵を確認する**

*$ ssh-add -l*

**エージェント転送をする**

*$ ssh -A hostname*

**エージェントを終了する**

*$ ssh-agent -k*

**なおUbuntuデスクトップでは……**

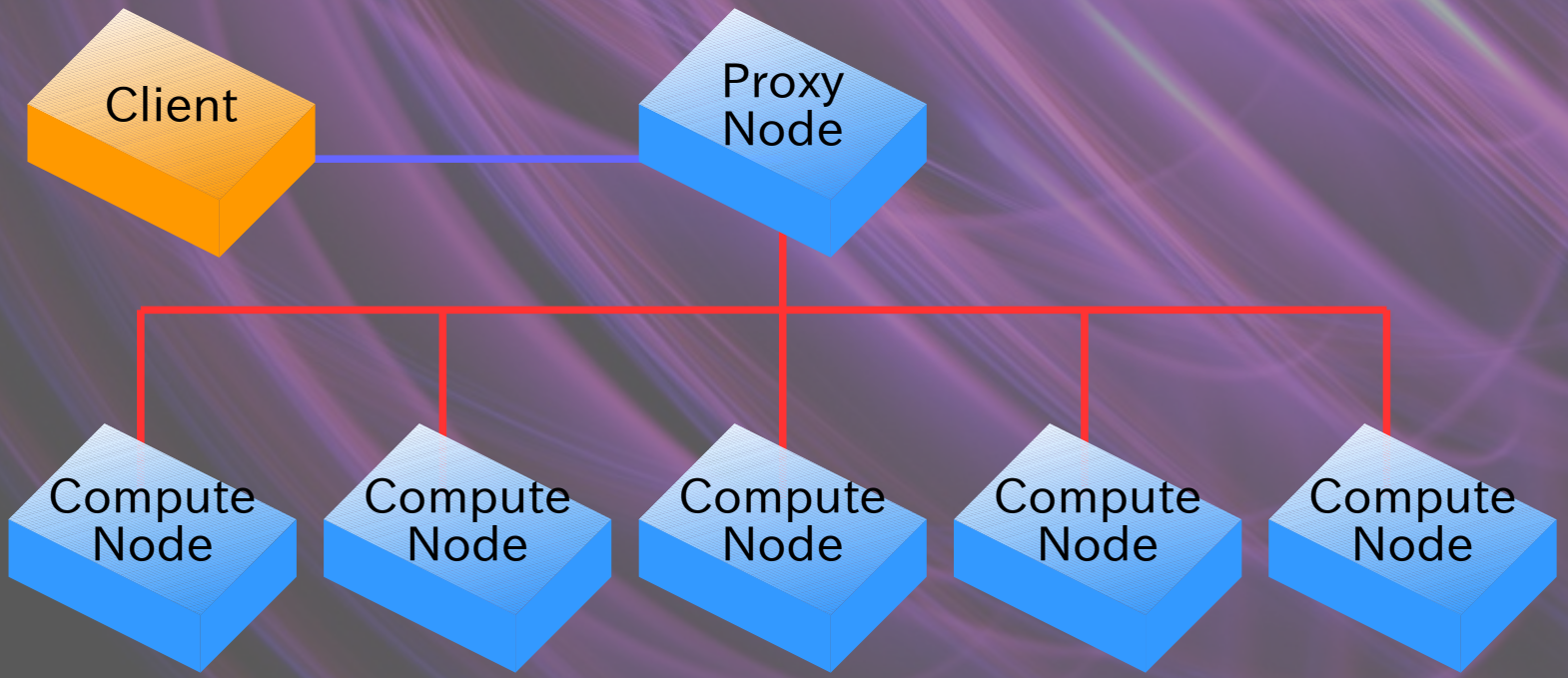
秘密鍵の初回使用時に、自動的にsshエージェントが起動する！

# Host-based Authentication

*——計算機クラスタのお共に*

**ユーザー単位ではなく、接続元ホストによって認証する**

クラスタでは、大量に増減する計算ノードごとにアカウントを作るのは非効率。
あるノードからの接続からは無条件に許可する、というのがホストベース認証。



# Configuration of SSH Server

*——SSHサーバーの設定を変更する*

## よりセキュアな運用のため サーバーの設定を変更する

**SSHサーバーの設定ファイル**

*/etc/ssh/sshd_config*

**SSHサーバーの再起動(設定反映)**

*$ sudo service ssh restart*

# Here is impassable

## ——待ち受けポートの変更

**標準の22番ポートはインターネットに晒さない**

*Port 22*
*↓*
*Port  24837*

22番ポートを晒すと、不正アクセスが実際スゴイ。
SSHと推測できないような、ランダムなポートを使うのおすすめ。
Port行は複数書ける。

**ポートを指定してSSH接続するには**

*$ ssh -p port hostname*

**ホストごとにランダムなポートなんて、覚えられない!**

*Host hostname*
*Port 24837*

~/.ssh/configに書いておけば、いちいち指定しなくてよい。

# Disabled "Open Sesame!"

*——パスワード認証を禁止する*

**パスワードが破られるのは時間の問題**

*PasswordAuthentication no*
*PermitRootLogin without-password*

公開鍵を設置したら、パスワード認証は禁止してしまうのがおすすめ。
PermitRootLoginはnoでもいい。
Ubuntuではデフォルトyesなので注意。

# To the chosen ones

*——ユーザーとグループで制限する*

## アクセス可能なユーザーとグループを指定する

*AllowUsers hoge fuga*
*DenyUsers foo bar*

デフォルトでは、すべてのユーザーがSSHログイン可能。
AllowUsersに指定されユーザーに限り、SSHログインが許可される。
AllowGroupsでは、グループ単位で管理可能。
DenyUsers/DenyGroupsは、特定のユーザーのみ拒否する。

判定はDenyUsers → AllowUsers → DenyGroups → AllowGroupsの順。

# How to use “Mobile Shell”

*——次世代のモバイルシェルを使おう*

## モバイル環境に最適のシェル

ローミング可能！通信中に接続断しても、IPアドレスが変わっても大丈夫。
UDPで通信。
SSPによって、通信の「詰まり」がない。
デーモンを起動しないので、ユーザー権限で実行可能。
ローカルエコーするので、細い回線でも快適！

## ただし……

SSHを完全に代替できるわけではない。
認証とサーバーの起動にはSSHを利用している。

## 詳しくはUbuntu Weekly Recipeで！

http://gihyo.jp/admin/serial/01/ubuntu-recipe/0220

# Any Questions?

*——質問はございませんか*

# Photos

Erik Ludwig (Spinning Optical Fiber)
Jeffrey (ubuntu-wallpaper-sunset)



Nrkbeta (Network cable up close)
CollegeDegrees360 (Schoolgirl with books on head)