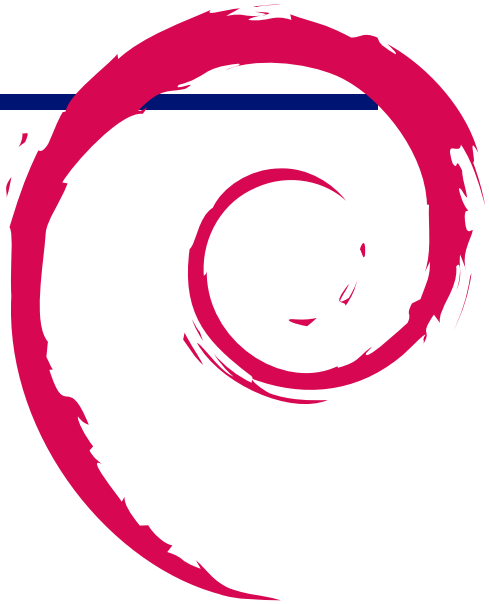
.Deb

銀河系唯一のDebian専門誌

2013 年 4 月 20 日

特集 1： samba で統合認証

特集 2： 開発環境管理



1 Introduction

上川 純一

今月の Debian 勉強会へようこそ。これから Debian の世界にあしを踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか?

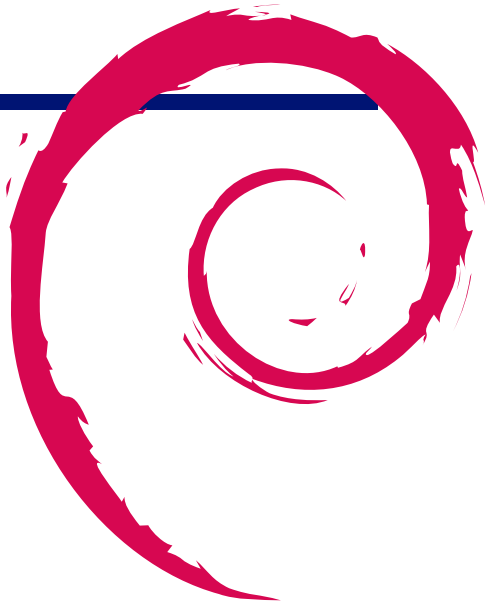
Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場の提供。
  - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
  - Debian のためになることを語る場を提供する。
  - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。

デビアン勉強会

目次



2 事前課題

上川 純一

今回の事前課題は以下です:

1. 最近 Debian のアカウント・ システム管理で考えていること
2. Debian での開発環境をどう管理しているか

この課題に対して提出いただいた内容は以下です。

2.1 べつやく

2.1.1 最近 Debian のアカウント・ システム管理で考えていること

ユーザアカウントの管理について特に意識したことがありません。ログイン時のパスワードを共通化できる仕組みがあるらしい・・・ぐらいの認識です。

2.2 koedoyoshida

2.2.1 最近 Debian のアカウント・ システム管理で考えていること

会社のサーバは Linux 系で統一されているので、 uid,gid, アカウント名をそれに合わせる程度です。 ldap 連携も可能ですが、自分の管理しているマシンでは (社内向けのサービスを提供していないので) 行っていません。

2.2.2 Debian での開発環境をどう管理しているか

stable 環境に chroot で sid 環境を作る orVMware 環境上へ構築しています。

2.2.3 Samba を Windows のドメインに参加させたことはありますか。

「 Samba を Windows のドメインに参加させたこと」、「 Linux のユーザ認証を Windows に統合したこと」はありません。逆に samba を DC として windows クライアントを参加させたことはあります。

2.3 ribbon@ns.ribbon.or.jp

2.3.1 Samba を Windows のドメインに参加させたことはありますか。

あります。

2.3.2 参加させたことがある方、ハマったところを教えてください

バージョンによって挙動が違うところです。

2.4 dictoss(杉本　典充)

2.4.1 最近 Debian のアカウント・ システム管理で考えていること

サーバで使っていても自分しか使用しないので、個人アカウントを作って完結してしまっている。

2.4.2 Debian での開発環境をどう管理しているか

amd64 マシンを使っているので、 i386 環境は chroot で入れるように設定している。 tarball を持ってきたアプリがビルド時に uname するものが一部あり、うまくビルドが通らないな場合を考えて KVM 環境も併用している。ただ、ネットワーク関係のテストをするときは iptables を使う場合が多いので最初から KVM 環境で作って試している。

2.5 henrich

2.5.1 最近 Debian のアカウント・システム管理で考えていること

Fedora のようにユーザーがアカウントを取得して、ウェブからアクセスできる様々な活動ができるといいな、と思っています。例: https://admin.fedoraproject.org/updates

2.5.2 Debian での開発環境をどう管理しているか

どう管理、というのがちょっと意味が掴み取れません。普通に cowbuilder と piuparts と lintian 使ってるぐらいですが、これをどのように変更していけば効率の良い環境になるのかは知りたい所です。

2.6 野首

2.6.1 最近 Debian のアカウント・システム管理で考えていること

Debian のアカウント管理は、自分の周りのマシンではいまだ/etc/passwd ベースばかりです。

2.6.2 Debian での開発環境をどう管理しているか

開発環境の管理としては、パッケージに svn-buildpackage と git-buildpackage を併用しています。特別使い分けたわけではなく、後になって git を覚えてから git-buildpackage を使ってみただけなので、これに特別な利点があったりはしません。むしろ設計思想が全然違うので一本化できず困っています。あとは sid 環境に lxc を使ってみたり、 arm や sparc のテストのために qemu を使い始めたりしている程度です。

2.7 石井一夫

2.7.1 最近 Debian のアカウント・システム管理で考えていること

セキュリティは結構深刻で、内部の者も悪さをする恐れがあり、かなり、気を使います。サーバでは、一般ユーザに sudo 権限を取得できないようにし、管理者用アカウントを作って wheel のグループに所属させ、それだけが、 sudo 権限を取得できるようにしています。パスワードは、 3 ヶ月ごとに変更。 Kerberos は気になりますが、使いこなせていません。

2.7.2 Debian での開発環境をどう管理しているか

GCC とか、 JDK とかインストールできるものは、可能な限りインストールしていますが、環境管理とまでは行っていません。これからの課題です。

2.8 上川純一

2.8.1 最近 Debian のアカウント・システム管理で考えていること

個人の環境は自分と目的用途別に作っているアカウントで、自分のアカウント以外はインタラクティブに利用する用途ではないので基本的にはパスワードを設定せず利用しています。

昔はファイル共有で NFS を使っていたのでシステム間で UID を一致させないといけないなどの面倒がありましたが最近は sshfs とか rsync/ssh でファイルを共有するので片方向の鍵認証でファイルのアクセス権が決まるモデルになっています。こっちのほうが管理者としてはやりやすいですね。

2.8.2 Debian での開発環境をどう管理しているか

cowbuilder で amd64 sid の環境のみを維持管理しています。昔はいろいろな CPU アーキテクチャーとか OS をとかをためしていたので大量のイメージがありました。 cowbuilder だと常に最新版のバイナリを使うことになります。また、必要なソフトウェアを必要なときに Just In Time でインストールしてくれることになります。 OS イメージの構成・履歴管理などを省略してくれるのでいいです。

開発環境の仮想マシンを管理するとどういうメモリを割り当てるかとかネットワークの設定をどうするかだとかというのも心配することになるんですが、 cowbuilder ではホスト OS の環境をそのままつかっているので楽です。開発環境のメンテナンスが目的ではなく開発環境を使うことが目的なので開発環境はメンテナンスフリーなのが理想だと思います。

各種クラウドサービスを使うと OS イメージのインストールからさせられることが多いのですがそこではじめていかに世間の人が環境の維持管理を重要なタスクだと思っていて、 cowbuilder が楽なのかを認識した気がします。

2.9 まえだこうへい

2.9.1 最近 Debian のアカウント・システム管理で考えていること

Debian の、というか LDAP に代わるもっとシンプルなアカウント管理の仕組みを作りたいなと思っているだけで何もやってません。

2.9.2 Debian での開発環境をどう管理しているか

Python でのツールの開発はの Sid を使っていますが、 Sid 自体は Virtual Box 上にあるので、 Shutdown 時に都度 snapshot を取っています。

2.9.3 Samba を Windows のドメインに参加させたことはありますか。

某所で Samba 使っていましたが、某所の AD には参加できない (その某所とは別の法人組織だったので) が、アクセスは某所の PC でしかできないので、ユーザアカウントを同一にするため、某所のシステムからユーザアカウント引っこ抜いて LDIF に変換

して LDAP でアカウント管理してました。(Samba とは直接連携してない)

2.10 吉野 (yy_y_ja_jp)

2.10.1 最近 Debian のアカウント・システム管理で考えていること

スタンドアロンで使うとき，VM で使うときがありますがどちらも考える必要を感じてません.

2.10.2 Debian での開発環境をどう管理しているか

どちらの場合も特に何もしてないです.

2.10.3 Samba を Windows のドメインに参加させたことはありますか。

ありません.

2.10.4 Linux のユーザ認証を Windows に統合したことはありますか。

ありません.

2.11 岩松 信洋

2.11.1 最近 Debian のアカウント・システム管理で考えていること

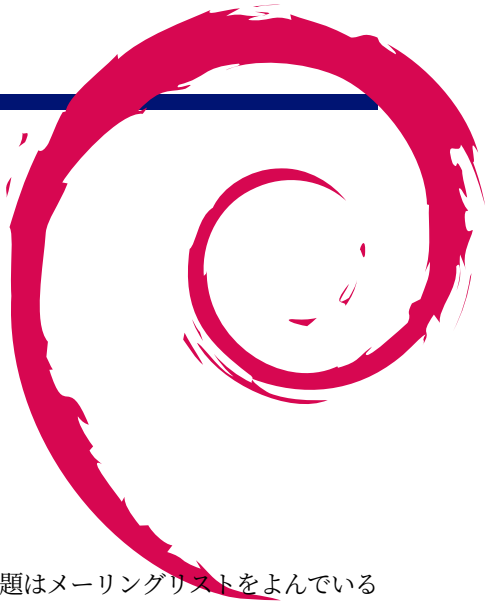
特にアカウント管理に関して考えてないです。昔ながらのユーザとグループで管理しています。

2.11.2 Debian での開発環境をどう管理しているか

chroot と alias, シェルスクリプトによる切り替えを使っています。

2.11.3 「Samba を Windows のドメインに参加させたことはありますか。参加させたことがない方、もしお時間があれば実際にやってみて、ハマったところを教えてください。参加させたことがある方、ハマったところを教えてください。」、「Linux のユーザ認証を Windows に統合したことはありますか。ある場合はその際の方法やハマったところについて教えてください。ない場合は、もしお時間があれば実際にやってみてハマったところを教えてください」にお答えください。

誰でもアクセスできるような samba サーバしか構築したことないので、特にハマったということはないです。



3 Debian Trivia Quiz

上川純一

ところで、みなさん Debian 関連の話題においついていますか? Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけではりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

今回の出題範囲は `debian-devel-announce@lists.debian.org` や `debian-devel@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 1. DSA のサーバを新しくホスティングしてくれるのはどれか

☐ A bytemark

☐ B grnet

☐ C manda

問題 2. wheezy の backports の apt-line で適切なものはどれか

☐ A deb sstp://backports.debian.org/debian/ wheezy-backports backports

☐ B deb http://backports.debian.org/debian/ wheezy-backports main

☐ C deb http://ftp.debian.org/debian/ wheezy-backports main

問題 3. tech ctte 699808 の結果アップロードされたのは

☐ A syslinux 4

☐ B syslinux 5

☐ C grub

問題 4. あたらしく DPL になったのはだれか

☐ A Stephano Zacchiroli

☐ B Lucas Nussbaum

☐ C Gergely Nagy

4 Debian 勉強会予約システム変更履歴

上川純一

Debian 勉強会の予約システムの機能をいくつか変更したので報告します。

4.1 アンケートのリマインダ

アンケートを送付していますが、現状メールで通知しているのみで、メールからしかアンケート回答できません。勉強会に登録している場合にアンケートを依頼してもメールがとどいていないとかメールに気づいていないというフィードバックをもらいました。

しかしながらメール通知以外で勉強会のあとにウェブページに来てもらうというのは難しいと思うのですが、ものは試しという事でトップページにリンクを表示するようにして見ました。参加しているイベントでアンケートの質問が作成されていてアンケートの回答がまだなされていない場合にアンケートのリンクを表示します。

4.2 勉強会に登録した時刻

SVG でなにができるのか試してみる勉強のついでに勉強会幹事用のページにグラフを出すようにして見ました。時刻を１０に分割してそれぞれの時刻においての登録の頻度の推移をみれるようにしています。このグラフを見ることでどのイベントでいつごろ何人登録したのかがわかります。試しに最近の二回を見てみたのですがけっこう違いますね。

HTML5 において SVG の画像をうめこむのは結構簡単で、そのまま SVG タグを埋め込めばいいだけです。 inkscape の吐く SVG をみて手書きは無理かなと思っていたのですが基本的な記法だけであれば手書きするのも悪くはないと思える書式でした。

```
<svg width="800" height="120">
         <text x="0" y="120">2012-11-03</text>
         <text x="400" y="120">2012-11-17</text>
         <text x="0" y="100">0</text>
         <text x="0" y="16">6</text>
         <polyline points="
0,83.3333333333
50,66.6666666667
100,83.3333333333
150,100.0
200,100.0
250,100.0
300,100.0
350,100.0
400,0.0
450,33.3333333333 "
         fill="none" stroke="#333">
         </polyline>
   </svg>
```

図 1　2012 年 11 月の勉強会の登録時刻

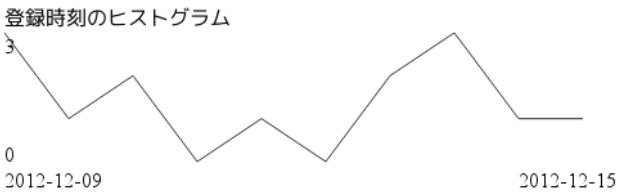


図 2　2012 年 12 月の勉強会の登録時刻

4.3　コミット数

気になったのでコミット数のグラフを作成してみました。四半期ごとにプロットしています。年末に集中してコミットしているのが見て取れます。しばらくいじってない気がしていましたが年に一回くらいは頑張っていじってる時期があるみたいですね。一年を振り返るついでにいろいろ弄りたくなるのかもしれません。

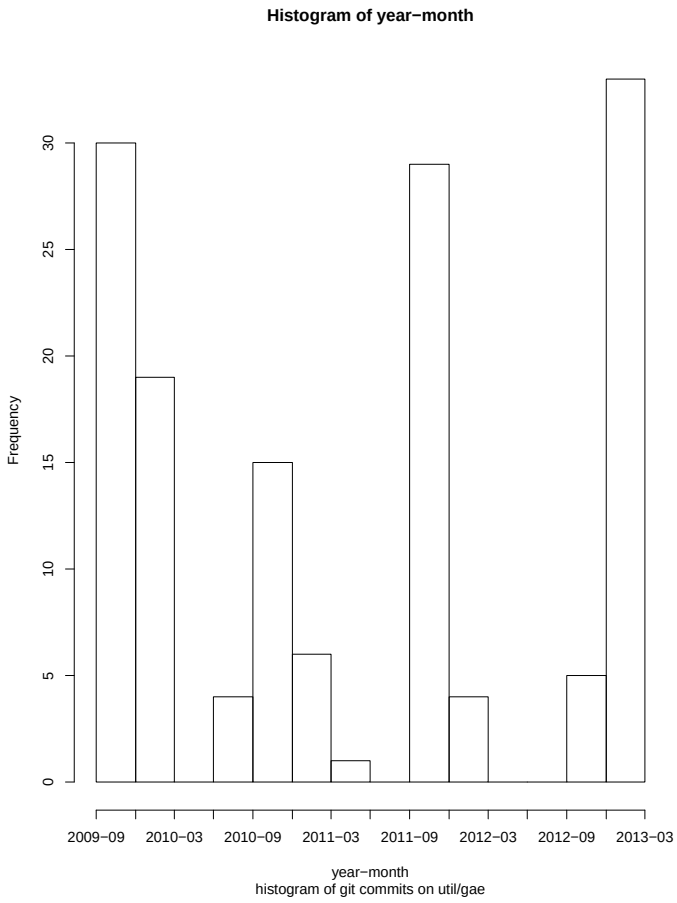
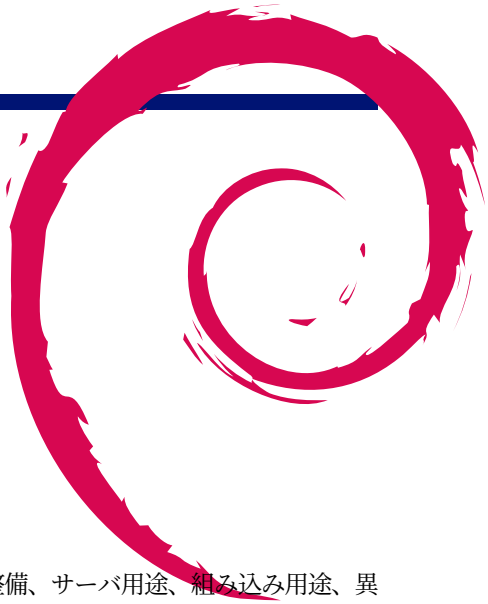


図 3　各四半期毎における util/gae ディレクトリの git コミット数



5 debootstrap を有効活用してみよう

杉本典充

debian のインストール処理に使う debootstrap コマンドの応用例として開発環境整備、サーバ用途、組み込み用途、異種 OS 利用とまとめてみました。

5.1 仮想化技術について

最近 KVM を事例とする「 仮想化」という言葉をよく聞くようになりました。仮想化にもいくつかの種類があると思い調べてみたところ、以下の例が見つかりました。今回は、コンテナ型仮想化に着目していきます。

仮想化技術	実装例	仮想化環境の特徴
完全仮想化 (エミュレーション型)	QEMU、 VirtualBox	既存の OS を無修正のままゲスト環境として動作させる。ホスト OS で動作する仮想化アプリケーションがエミューションする。
完全仮想化 (ハイパーバイザ型)	KVM	既存の OS を無修正のままゲスト環境として動作させる。 CPU 等の仮想化機能を使うことでホスト環境におけるオーバーヘッドを極力減らしている。
準仮想化	Xen	ホスト環境とやりとりする API を利用できるように修正が入った OS をゲスト OS として動作させる方式(=既存の OS そのままでは動かない)
コンテナ型仮想化	OpenVZ、 LXC、 FreeBSD jail	ホスト環境とゲスト環境は同一カーネルで動作しつつ、ホスト環境から分離したゲスト環境を提供する。

5.2 debian におけるコンテナ環境の作り方

5.2.1 debootstrap

debian においてコンテナ型仮想化の環境を構築する機能として debootstrap があります。 debootstrap を実行すると、 debian をインストールしたときのルートディレクトリ構造を任意のパスに作成することができます。

debootstrap のインストールは以下の apt-get コマンドで実行できます。 deboostrap には debootstrap (sh スクリプトで実装) と cdebootsrap (C 言語で実装) の 2 種類のコマンドがあります。

```
$ sudo apt-get install debootstrap cdebootstrap
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --arch=amd64 sid ./mysid http://ftp.jp.debian.org/debian
$ ls mysid
bin   dev  home  lib64  mnt  proc  run   selinux  sys  usr
boot  etc  lib   media  opt  root  sbin  srv      tmp  var
```

5.2.2 chroot コマンド

debootstrap で作成したコンテナ環境でプログラムを実行したいのですが、環境変数とディレクトリの配置が合わないためうまくコマンドを実行することができません。そこで chroot コマンドを使います。[*1] [*2]

chroot コマンドを実行するとルートディレクトリでない場所をあたかもルートディレクトリであるように見せることができます。これにより普通に debian をインストールした状態のプログラムの配置と PATH 環境変数がうまく合い、コンテナ環境のプログラムを実行できるようになります。

```
$ sudo chroot /srv/chroot/mysid
# pwd
/
# ls
bin   dev  home  lib64  mnt  proc  run   selinux  sys  usr
boot  etc  lib   media  opt  root  sbin  srv      tmp  var
```

chroot コマンドにも現在ではいくつかの派生版があります。

- chroot
  - 元祖 chroot。実行するには root 権限が必要。
- dchroot
  - chroot を一般ユーザで実行できるように改良した版。(とはいえ debootstrap と設定ファイル作成は root 権限が必要)
- schroot
  - dchroot のセキュリティレベルを向上させた改良版。

今度は schroot コマンドでコンテナ環境に入ってみます。

```
$ sudo apt-get install schroot
$ cd /etc/schroot
$ sudo vi schroot.conf

[mysid]
description=my sid for devel
type=directory
directory=/srv/chroot/mysid
users=norimitu
root-groups=root
personality=linux
preserve-environment=true

$ schroot -c mysid
W: Failed to change to directory‘ /etc/schroot’: No such file or directory
I: The directory does not exist inside the chroot.  Use the --directory option to run the command in a different directory.
W: Falling back to directory‘ /home/norimitu
$ ls -la /srv
total 8
drwxr-xr-x  2 root root 4096 Apr 14 02:59 .
drwxr-xr-x 22 root root 4096 Apr 14 03:04 ..

(何もないです。chroot 環境に入っています)
$ cd /home/norimitu
$ ls
(chroot 元のホスト環境のファイルが出てくる。)
```

直感的には chroot コマンドの方がわかりやすいです。

schroot コマンドの場合は、schroot した環境内のホームディレクトリはホスト環境のホームディレクトリと bind されます。そのため、ホスト環境にあるファイルを schroot した環境からそのまま読み書きできます。(chroot の場合はそう

[*1] chroot システムコールは 1982 年にビル・ジョイが作成したものが起源とされています。ビル・ジョイはプログラムをクリーンビルドできる環境がほしかったため通常利用の環境と分離する手段として開発したと言われています。

[*2] クリーンビルド目的で chroot コマンドを使う手法では debian パッケージのビルドにも使われています。

いう仕組みがないため、コピーする必要がある。）

5.3 debootstrap の使いどころ

debootstrap を使うと何が便利なのか使いどころの例を上げてみます。

5.3.1 生活環境と開発 (テスト) 環境を分離する

普段使用する環境は stable を利用するが、バージョン等の都合で部分的に sid で提供されているパッケージを利用したい場合があります。その場合、前述した方法でコンテナ環境を作成し、 chroot すれば利用できます。

5.3.2 コンテナ内で異なる CPU アーキテクチャの環境を動かす

debootstrap したコンテナ環境は通常ホスト環境で動作する CPU アーキテクチャで構築します。 chroot コマンドでコンテナ環境に入っている場合でもカーネルはホスト OS と同じため、異なる CPU アーキテクチャのバイナリをネイティブに実行できないためです。

amd64 カーネルを実行しているマシンの場合、 i386 バイナリは実行できますので i386 環境のコンテナ環境を構築すれば実行できます。

私は諸般の都合で i386 の postgresql-9.1 を動かして VPS へレプリケーションしていますが、 amd64 と i386 間といった CPU アーキテクチャが異なる場合にレプリケーションができない仕様となっているため、 VPS のホスト OS は amd64、コンテナ内で i386 環境を動かしてレプリケーションしています。

```
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --arch=i386 sid ./mysid-i386 http://ftp.jp.debian.org/debian
$ sudo chroot ./mysid-i386
# uname -a
Linux www6368uj 3.2.0-4-amd64 #1 SMP Debian 3.2.41-2 x86_64 GNU/Linux
# apt-get install file
# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=0x498625fc854816515ec12819544ebedff51d5c32, stripped
```

また、 chroot と QEMU を組み合わせることにより、ホスト OS のカーネルでは動作しない CPU アーキテクチャのコンテナ環境を QEMU を使ったエミュレートで動作させることができます。最近流行りの安価な ARM ボードで debian を動作させるためのディレクトリツリーを作成するのに便利です。[*3]

ホスト環境と異なる CPU アーキテクチャのコンテナ環境を chroot する場合、以下の手順で実行する必要があります。

- debootstrap は「 –foreign」引数を渡して実行すること
- コンテナ環境内に「 qemu-*-static」ファイルをコピーしておくこと（ 例: qemu-arm-static）
- chroot で入った後で debootstrap の second stage を実行すること

[*3] 実際に ARM ボード上で debian を起動させるためには SD カード等へのファイルシステム作成、カーネルやドライバの作成、ブートローダーの書き込み等、色々やらないと動きません。

```
$ sudo apt-get install binfmt-support qemu qemu-user-static debootstrap
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --foreign --arch=armel wheezy ./armdev1 http://ftp.jp.debian.org/debian
$ sudo chroot ./armdev1
chroot: コマンド ‘/bin/bash’ の実行に失敗しました: No such file or directory

$ sudo cp /usr/bin/qemu-arm-static /srv/chroot/armdev1/usr/bin/
$ sudo chroot armdev1
I have no name!@hostname:/#
Linux hostname 3.2.0-4-amd64 #1 SMP Debian 3.2.41-2 armv7l GNU/Linux

I have no name!@hostname:/# apt-get update
bash: apt-get: command not found
I have no name!@hostname:/# ls /debootstrap
arch  debootstrap      debpaths        functions  suite
base  debootstrap.log  devices.tar.gz  required   suite-script
I have no name!@hostname:/# /debootstrap/debootstrap --second-stage
I: Installing core packages...
I have no name!@hostname:/# apt-get update
Reading package lists...

I have no name!@hostname:/# vi /etc/apt/sources.list

deb http://ftp.jp.debian.org/debian wheezy main
deb-src http://ftp.jp.debian.org/debian wheezy main

deb http://security.debian.org/ wheezy/updates main
deb-src http://security.debian.org/ wheezy/updates main

I have no name!@hostname:/# apt-get update
I have no name!@hostname:/# apt-get install file
I have no name!@hostname:/# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=0x5bc97dbca9ac168932d898a5e2eaf68e8fde5e16, stripped
```

5.3.3 サーバでたくさんのコンテナを常駐させて動かしたい

今までのコンテナ環境は手でコマンドを実行することでコンテナ環境に入っていました。コンテナ環境をサーバのように動かしたいというニーズもあります。今回は Debian GNU/Linux wheezy amd64 上で LXC（ Linux Containers）を用いてコンテナ環境を常駐させてみます。

まずホスト環境の設定と環境を確認します。

```
$ sudo apt-get install lxc
$ sudo vi /etc/fstab

(追記します)
cgroup  /sys/fs/cgroup  cgroup  defaults  0   0

$ sudo mount -a
$ lxc-checkconfig
Kernel config /proc/config.gz not found, looking in other places...
Found kernel config file /boot/config-3.2.0-4-amd64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

LXC で動作するコンテナ環境のネットワークはホスト環境とブリッジ接続する必要があります。

ここでは eth0 はそのままでブリッジデバイスを１つ作成し、 LXC のコンテナ環境はブリッジデバイスの NAT 環境下

で動作することとします。[*4]

```
$ sudo vi /etc/sysctl.conf

( 変更)
net.ipv4.ip_forward=1

$ sudo sysctl -p
net.ipv4.ip_forward = 1

$ vi br-lxc.sh
# script to setup a natted network for lxc guests
CMD_BRCTL=/sbin/brctl
CMD_IFCONFIG=/sbin/ifconfig
CMD_IPTABLES=/sbin/iptables
CMD_ROUTE=/sbin/route
NETWORK_BRIDGE_DEVICE_NAT=lxc-bridge-nat
HOST_NETDEVICE=eth0
PRIVATE_GW_NAT=192.168.20.1
PRIVATE_NETMASK=255.255.255.0
PRIVATE_NETWORK=192.168.20.0/24

${CMD_BRCTL} addbr ${NETWORK_BRIDGE_DEVICE_NAT}
${CMD_BRCTL} setfd ${NETWORK_BRIDGE_DEVICE_NAT} 0

${CMD_IFCONFIG} ${NETWORK_BRIDGE_DEVICE_NAT} ${PRIVATE_GW_NAT} netmask ${PRIVATE_NETMASK} promisc up

${CMD_IPTABLES} -t nat -A POSTROUTING -o ${HOST_NETDEVICE}  -j MASQUERADE
${CMD_IPTABLES} -A FORWARD -i ${NETWORK_BRIDGE_DEVICE_NAT} -o ${HOST_NETDEVICE} -s ${PRIVATE_NETWORK} -j ACCEPT

$ sudo ./br-lxc.sh
$ sudo ifconfig lxc-bridge-nat
lxc-bridge-nat Link encap:イーサネット  ハードウェアアドレス fe:24:c3:eb:6d:c3
          inet アドレス:192.168.20.1 ブロードキャスト:192.168.20.255  マスク:255.255.255.0
          inet6 アドレス: fe80::409a:3cff:fee6:33b0/64 範囲:リンク
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  メトリック:1
          RX パケット:6556 エラー:0 損失:0 オーバラン:0 フレーム:0
          TX パケット:1121 エラー:0 損失:0 オーバラン:0 キャリア:0
      衝突 (Collisions):0 TX キュー長:0
          RX バイト:543131 (530.4 KiB)  TX バイト:95778 (93.5 KiB)
```

コンテナを作成します。 LXC におけるコンテナは “/var/lib/lxc” ディレクトリの下にできます。

```
$ sudo lxc-create -n lxc-deb1 -t debian
  ( ダイアログ形式でインストールするバージョンやダウンロード先ミラーの URL、
    root パスワードが聞かれるので答える)

$ cd /var/lib/lxc/lxc-deb1
$ ls
config    rootfs
$ sudo vi config

( 追記します)
## Network
lxc.utsname = lxc-deb1
lxc.network.type = veth
lxc.network.flags = up

# that's the interface defined above in host's interfaces file
lxc.network.link = lxc-bridge-nat

# name of network device inside the container,
# defaults to eth0, you could choose a name freely
# lxc.network.name = lxcnet0

lxc.network.hwaddr = 00:FF:AA:00:00:01

# the ip may be set to 0.0.0.0/24 or skip this line
# if you like to use a dhcp client inside the container
lxc.network.ipv4 = 192.168.20.101/24
```

LXC コンテナ内で sshd が起動するよう設定しておきます。

```
$ sudo cp /etc/resolv.conf /var/lib/lxc/lxc-deb1/rootfs/etc/
$ sudo vi /var/lib/lxc/lxc-deb1/rootfs/etc/ssh/sshd_config

( 変更前)   #ListenAddress 0.0.0.0
( 変更後)   ListenAddress 192.168.20.101
```

LXC コンテナを起動します。 “-d” オプションはバックグラウンドで起動させるオプションです。これでコンテナ環境が常駐して動作するようになります。

[*4] http://wiki.debian.org/LXC/SimpleBridge

```
$ sudo lxc-start -n lxc-deb1
Using makefile-style concurrent boot in runlevel 2.
Starting OpenBSD Secure Shell server: sshdCould not load host key: /etc/ssh/ssh_host_rsa_key
Could not load host key: /etc/ssh/ssh_host_dsa_key

なぜか ssh 鍵を作る処理が自動実行されず止まってしまうため、手動で作ってみる。
$ sudo chroot /var/lib/lxc/lxc-deb1/rootfs ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
$ sudo chroot /var/lib/lxc/lxc-deb1/rootfs ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key

$ sudo lxc-start -n lxc-deb1 -d
$ ssh root@192.168.20.101
root@lxc-deb1:~#

  ログインできました。

$ sudo lxc-stop -n lxc-deb1
```

5.3.4 異なる OS 上で debian を構築する

debootstrap は異なる OS 上で debian を構築することができます。ここでは、 FreeBSD-8.3-RELEASE amd64 の jail 機能を用いて kfreebsd-amd64 を prisoner として動かしてみます。[*5]

まずは jail 環境を作成するためのツールをインストールします。

OS が異なると debootstrap コマンドがないため、 tarball（ 中身は sh スクリプト版）をダウンロードして実行します。[*6]

```
> cd
> wget http://ftp.jp.debian.org/debian/pool/main/d/debootstrap/debootstrap_1.0.48.tar.gz
> tar xf debootstrap_1.0.48.tar.gz
> cd debootstrap-1.0.48
# su
# setenv DEBOOTSTRAP_DIR ‘pwd‘
# ./debootstrap --arch=kfreebsd-amd64 wheezy /usr/jails/jailkfdeb http://ftp.jp.debian.org/debian
# kldload fdescfs linprocfs linsysfs tmpfs
# umount /usr/jails/jailkfdeb/dev/fd
# umount /usr/jails/jailkfdeb/dev
# mount -t linprocfs linprocfs /usr/jails/jailkfdeb/proc
# mount -t linsysfs linsysfs /usr/jails/jailkfdeb/sys
# mkdir -p /usr/jails/jailkfdeb/lib/init/rw
# mount -t tmpfs tmpfs /usr/jails/jailkfdeb/lib/init/rw
# cp /etc/resolv.conf /usr/jails/jailkfdeb/etc/resolv.conf
```

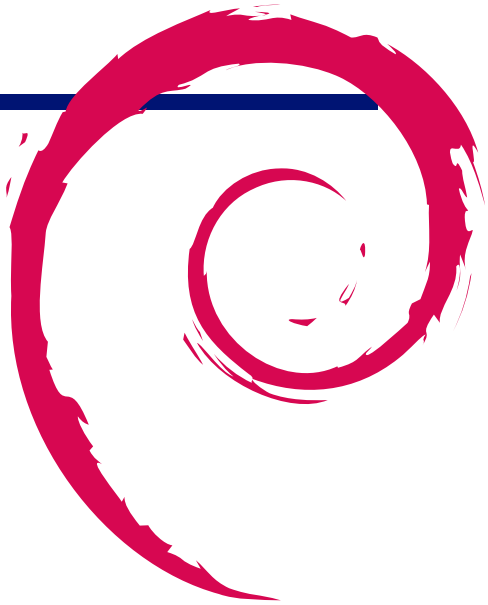
[*5] http://blog.vx.sk/archives/22-Updated-Tutorial-Debian-GNUkFreeBSD-in-a-FreeBSD-jail.html
[*6] ports で/usr/ports/sysutils/debootstrap もありますのでそちらを使ってもいいです

```
> sudo portsnap fetch
> sudo portsnap update
> cd /usr/ports/sysutils/ezjail
> sudo make
> sudo make install

> sudo touch /etc/fstab.jailkfdeb
  （ このファイルがないとエラーになるためとりあえず作成します）
> vi /etc/rc.conf
（ 追記します）
ifconfig_em0_alias0="inet 192.168.1.63/32"

> ifconfig em0 192.168.1.63 netmask 255.255.255.255 alias
> vi /usr/local/etc/ezjail/jailkfdeb

# To specify the start up order of your ezjails, use these lines to
# create a Jail dependency tree. See rcorder(8) for more details.
#
# PROVIDE: standard_ezjail
# REQUIRE:
# BEFORE:
#

export jail_jailkfdeb_hostname="jailkfdeb"
export jail_jailkfdeb_ip="192.168.1.63"
export jail_jailkfdeb_rootdir="/usr/jails/jailkfdeb"
export jail_jailkfdeb_exec_start="/etc/init.d/rc 3"
export jail_jailkfdeb_exec_stop=""
export jail_jailkfdeb_flags="-l -u root"
export jail_jailkfdeb_mount_enable="YES"
export jail_jailkfdeb_devfs_enable="YES"
export jail_jailkfdeb_devfs_ruleset="devfsrules_jail"
export jail_jailkfdeb_procfs_enable="YES"
export jail_jailkfdeb_fdescfs_enable="YES"


> sudo /usr/local/etc/rc.d/ezjail start jailkfdeb
Configuring jails:.
Starting jails: jailkfdeb.
> jls
   JID  IP Address      Hostname                      Path
    11  192.168.1.63    jailkfdeb                     /usr/jails/jailkfdeb
> jexec 11 /bin/sh

（ ここからコンテナ環境の中です）

# uname -irps
GNU/kFreeBSD 8.3-RELEASE-p6 amd64 Intel(R) Core(TM) i5-2500S CPU @ 2.70GHz
# vi /etc/apt/sources.list

deb http://ftp.jp.debian.org/debian wheezy main contrib non-free
deb-src http://ftp.jp.debian.org/debian wheezy main contrib non-free

# apt-get update
# apt-get install openssh-server
# vi /etc/ssh/sshd_config

修正前）  #ListenAddress 0.0.0.0
修正後）  ListenAddress 192.168.1.63

# /etc/init.d/ssh restart
# passwd
# exit

（ コンテナ環境からホスト環境に戻ります）

> ssh root@192.168.1.63
```

ssh でコンテナ環境にログインできました。

5.4 参考情報

- Debian Wiki Schroot http://wiki.debian.org/Schroot
- Debian Wiki LXC http://wiki.debian.org/LXC
- Debian Wiki EmDebianCrossDebootstrap http://wiki.debian.org/EmDebian/CrossDebootstrap
- 関西エリア Debian 勉強会 2009 年 04 月号「 Debian を新規に install してから常用環境にするまで」 佐々木洋平
- Updated Tutorial: Debian GNU/kFreeBSD in a FreeBSD jail http://blog.vx.sk/archives/22-Updated-Tutorial-Debian-GNUkFreeBSD-in-a-FreeBSD-jail.html



6 Samba で Linux の認証を Windows に統合してみたり

たかはしもとのぶ

6.1 無慈悲な一言、言われませんか w

Windows の大群の中で Linux サーバを運用しているとよく言われるのが、「 その (Linux) サーバの認証、 Active Directory でできないの?」という大変無慈悲な (笑) 一言ではないでしょうか。ということで、今回は Samba を使って Linux サーバの認証を Windows に統合する方法について、いろいろご紹介していきたいと思います。

6.2 Samba って · · · · · ·

一応説明しておきますと、 Debian を含む UNIX 系 OS 上でファイル共有をはじめとする Windows サーバの各種機能を実現するオープンソースのソフトウェアです。元々はファイル共有、プリンタ共有の機能から出発していますが、最近では Windows の中核機能である Active Directory のドメインコントローラ機能や、 Windows ドメインのクライアント機能など、多くの機能を実装しています。

ただ、そのあたりの機能の紹介をしていると、 Samba というより Windows の機能の紹介になってしまうので、今回は Linux メインな方でも需要がありそうな機能ということで、認証統合を取り上げてみました。

6.3 Samba でファイルサーバを構築してみる

まずは、 Samba の機能紹介も兼ねて Samba でファイルサーバを構築してみましょう。これはパッケージを適切にインストールすれば簡単です。

```
# apt-get install samba
```

インストールの途中で聞かれる質問は、 OK を押して進めておきましょう (後で修正します)。

インストールが終わったら、/etc/samba/smb.conf を編集します。デフォルトでは各ユーザのホームディレクトリを読み取り専用で共有する設定になっていますので、ここでは読み書き可能にしてみます。

```
[homes]
   comment = Home Directories
   browseable = no

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
   read only = no ← yes から変更
```

もう一つ、 Samba でファイル共有を行う例として/tmp を読み書き可能で共有してみましょう。次のような記述を smb.conf の末尾に追加します。

```
[tmp-share]
  path = /tmp
  read only = no
```

1 行目が共有名の指定で、ここでは tmp-share という名前を指定しています。 2 行目は共有するパスの指定で、ここでは/tmp を指定しました。 3 行目は読み書き可能とする設定です。デフォルトでは読み取り専用で共有されます。

次に、このサーバへのアクセスを行うユーザを作成します。/etc/passwd のユーザに加えて、 Samba では Windows 独自の形式でハッシュ化された認証情報を扱う必要があるため、 smbpasswd コマンドなどを用い、 Samba ユーザという独自のユーザを作成してパスワードを設定する必要があります。

```
# useradd -m local1
# smbpasswd -a local1
New SMB password:
Retype new SMB password:
Added user local1.
```

ここで次のようにして Samba サーバを再起動して smb.conf の設定変更を反映させます。

```
# /etc/init.d/samba restart
Stopping Samba daemons: nmbd smbd.
Starting Samba daemons: nmbd smbd.
```

再起動後、 ¥¥Samba サーバの IP アドレス として Windows からアクセスしてみましょう。ファイアウォールなどの設定を行っていなければ、図 4 のようにユーザ名とパスワードを聞かれるダイアログが表示されます。

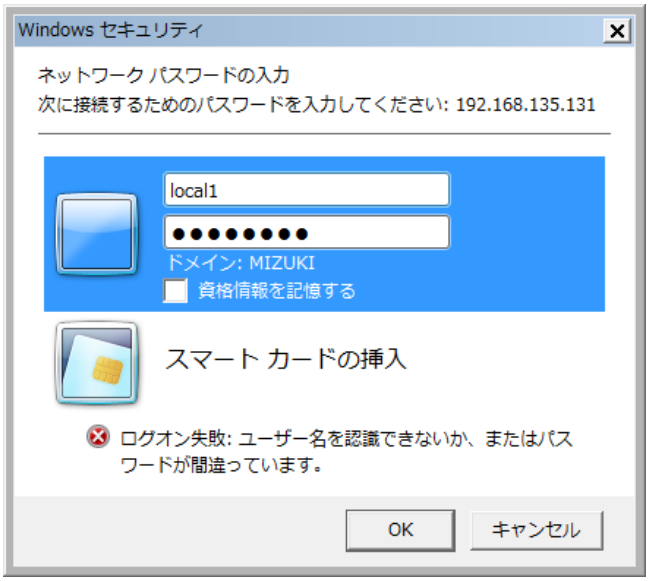


図 4　ユーザー名とパスワードの入力ダイアログ

先ほど設定した user1 というユーザ名とパスワードを入力すれば、 user1 というホームディレクトリの共有と tmp-share という先ほど作成した共有が見えているはずです (図 5)。

6.4　Samba を Active Directory(Windows ドメイン) に参加させてみる

次に、 Samba を Active Directory に参加させて、先ほど構築したファイルサーバへのアクセスの際の認証を Windows ドメインに統合してみましょう。これもパッケージを適切にインストールしていれば簡単です。

まず次のようにして/etc/resolv.conf を編集して、 DNS ドメイン、 DNS サーバとして Active Directory の DNS サーバを指定します (まだ指定していない場合) 。ここでは W2K8R2AD3.LOCAL というドメイン名で DNS サーバの IP アドレスが 192.168.135.100 である場合の例を示します (DHCP から IP アドレスを取得する設定の場合、このファイルは上書きされてしまうので、固定 IP の設定にしておく必要があります):

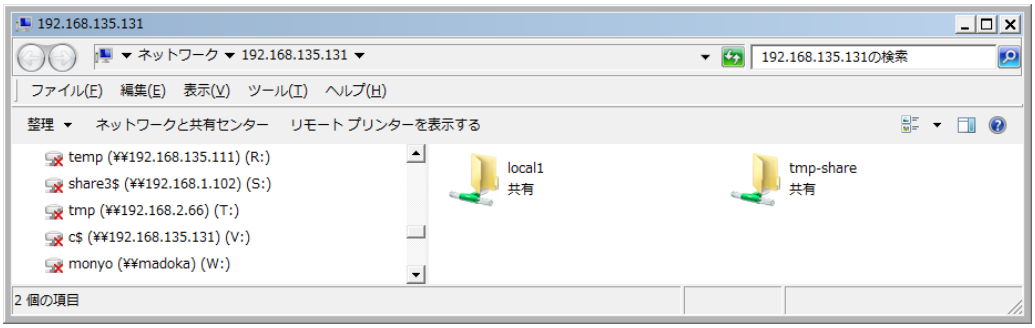


図 5　エクスプローラーから Samba サーバの共有を参照

```
domain w2k8r2ad3.local ← Active Directory のドメイン名を指定
search w2k8r2ad3.local
nameserver 192.168.1.100 ← Active Directory の DNS サーバ（通常はドメインコントローラ）の IP アドレスを指定
```

ついで、 smb.conf ファイルを修正します。ここでは W2K8R2AD3.LOCAL という Active Directory ドメインに参加させる場合を例に取って説明します。

```
[global]
  ...

  workgroup = W2K8R2AD3

  ...

  security = ads
  realm = W2K8R2AD3.LOCAL
```

最後に、 net ads join コマンドを使って Samba をドメインに参加させます

```
# net ads join -U administrator
Enter administrator's password:
Using short domain name -- W2K8R2AD3
Joined 'SQUEEZE32-5' to realm 'W2K8R2AD3.local'
No DNS domain configured for squeeze32-5. Unable to perform DNS Update.
DNS update failed!
```

-U オプションに続き、ドメイン参加に使用するユーザを指定します。これは必ずしも Administrator である必要はありませんが、 Active Directory 側の設定に依存しますので、事前に確認していてください。参加の際、 DNS update failed! というエラーが出ますが、これは無視して構いません。

次に、このサーバへのアクセスを行うユーザを作成します。認証は Active Directory で行うので、 Samba ユーザは不要です。/etc/passwd にユーザを追加します。このユーザのユーザ名は Active Directory 側と合わせる必要があります。ここでは Active Directory 側に aduser01 というユーザが既に作成済である前提で、 aduser01 を追加して見ます:

```
# useradd -m aduser01
```

ここで次のようにして Samba サーバを再起動して smb.conf の設定変更を反映させます。

```
# /etc/init.d/samba restart
Stopping Samba daemons: nmbd smbd.
Starting Samba daemons: nmbd smbd.
```

再起動後、 Windows 側に aduser01 としてログオンの上、 ~~¥¥~~Samba サーバの IP アドレスとして Windows からアクセスしてみましょう。認証を聞かれることなく、 Samba サーバにアクセスできるはずです。何かファイルを作成の上 Debian 上で確認すると、次のようにファイルの所有者が aduser01 になっていることが確認できると思います。

```
# ls -l
total 4
drwx------ 2 root     root     4096 Apr 14 11:24 ssh-XHMOEq1038
-rwxr--r-- 1 aduser01 aduser01    0 Apr 14 11:22 test01.txt
```

Active Directory 参加前との簡単な比較を図 6 に示します:

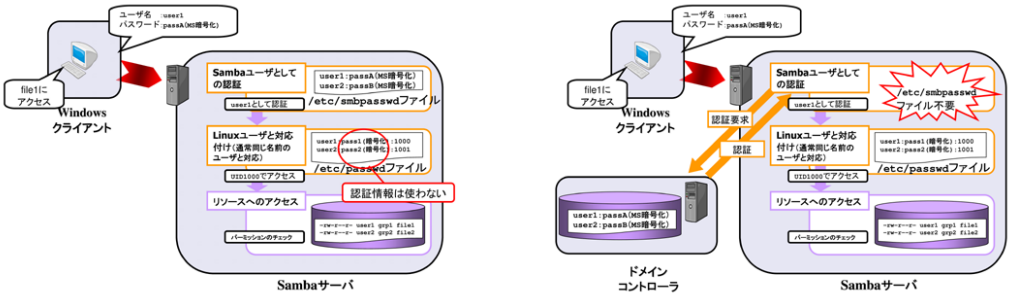


図 6　Active Directory 参加前後の認証フローの比較

6.5　ユーザの作成を自動化する

ここまでの設定で、認証の統合には成功しました。しかしこの状態だと認証を行いたいユーザ毎に Debian 上の /etc/passwd にユーザを作成する必要があります。多数のユーザがアクセスする必要がある環境では、ユーザのメンテナンスがかなりの負荷になってしまうと思います。

Samba では **Winbind** という機能を使うことで、ユーザの自動生成が可能になります。パッケージを使っていれば、この設定も簡単に行えます。

まず、次のようにして winbind パッケージをインストールします:

```
# apt-get install winbind
```

wbinfo -t コマンドを用いて、次のように RPC が成功することを確認しておきましょう:

```
# wbinfo -t
checking the trust secret for domain W2K8R2AD1 via RPC calls succeeded
```

引き続き、/etc/nsswitch.conf の passwd:および group:行に、次のように winbind というキーワードを追加します:

```
passwd:         compat winbind
group:          compat winbind
```

最後に、 smb.conf 内の次の行のコメントを外し、設定を追加して、 winbind を再起動します:

```
# Some defaults for winbind (make sure you're not using the ranges
# for something else.)
   idmap uid = 10000-20000
   idmap gid = 10000-20000
   template shell = /bin/bash
   template homedir = /home/%U
```

ここで Active Directory に例えば aduser02 というユーザを追加して、その情報を取得すると……

Samba サーバの/etc/passwd では特にユーザの追加を行っていないにも関わらず、次のように Winbind 機構によって自動生成されたユーザ情報が返却されるはずです:

```
$ id 'W2K8R2AD3\aduser02'
uid=10001(W2K8R2AD3\aduser02) gid=10000(W2K8R2AD3\domain users) groups=10000(W2K8R2AD3\domain users)
$ getent passwd  'W2K8R2AD3\aduser02'
W2K8R2AD3\aduser02:*:10001:10000:aduser 02:/home/aduser02:/bin/bash
```

Windows にユーザでログオンして Samba サーバの共有に何かファイルを作成すると、次のように、ユーザ名、グループ名に自動生成されたものが表示されていることが確認できます:

```
$ ls -l /tmp
total 4
drwx------ 2 root                root                    4096 Apr 14 12:40 ssh-wQPxWv1345
-rwxr--r-- 1                1001 aduser01                   0 Apr 14 11:22 test01.txt
-rwxr--r-- 1 W2K8R2AD3\aduser02 W2K8R2AD3\domain users    0 Apr 14 12:39 test02 - コピー.txt
```

Winbind 機構のない状態と Winbind 機構を有効にした状態での比較を次の図 7 に示します:

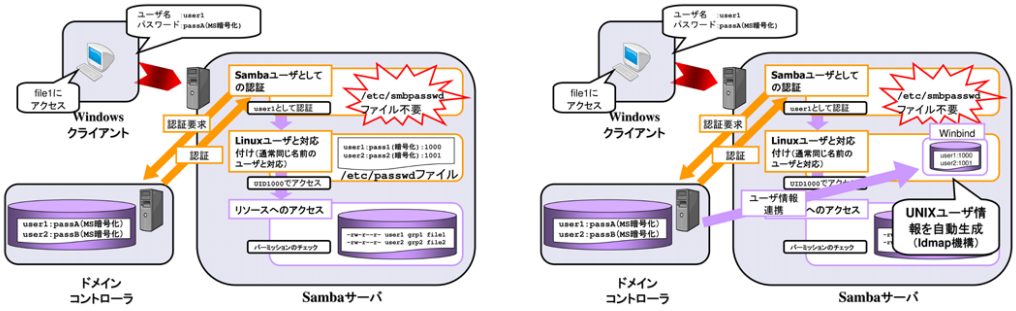


図 7　Winbind 機構有無による認証フローの比較

ここで自動生成されるユーザ、グループの UID,GID やシェルなどの情報は、さまざまな設定でカスタマイズすることが可能です。今回は時間がないので省略しますが、 1 点だけ、生成されるユーザ名、グループ名に付加されるドメイン名部分が煩雑だと感じる場合は、 `smb.conf` に次のパラメータを設定してください。

```
  winbind use default domain = yes
```

次のようにドメイン名がない単純な名前で表示されるようになります:

```
$ id aduser02
uid=10001(aduser02) gid=10000(domain users) groups=10000(domain users)
```

ただし、当然ですが Samba が所属するドメインが他のドメインと信頼関係を結んでいるような複数ドメイン環境では名前が重複することがあります。

6.6　ssh の認証を Windows に統合してみる

ここまでの設定で、 Samba を用いる際のユーザ認証と、ユーザの自動作成について説明しました。最後に Samba 以外のプロダクトからこのユーザ認証を活用する方法について説明します。ここでは ssh を例に取って説明しますが、 PAM で認証を行うプロダクトであれば、同様の適用が可能です。

実は、ここまでの設定を行っていれば、 Active Directory のユーザを用いて Linux にログインすることが (他の方法でログインを抑止していない限り) 可能です。次に例を示します:

```
$ ssh 192.168.135.35 -l W2K8R2AD3\\aduser02
W2K8R2AD3\aduser02@192.168.135.35's password:
Linux squeeze32-5 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Could not chdir to home directory /home/aduser02: No such file or directory
W2K8R2AD1\aduser02@squeeze32-5:/$
```

ホームディレクトリを作成していないため、エラーが発生していますが、ログイン自体は無事に成功していることが分かります。もちろんこの aduser02 というユーザは Winbind 機構によって自動作成されたものですので、`/etc/passwd` 上には存在していません。

ホームディレクトリがないと何かと不便ですが、手作業で作成するのもゲイがないので、最後にホームディレクトリを自

動で作成する方法を紹介します。まず次のようにして libpam-modules をインストールしてください:

```
# apt-get install libpam-modules
```

その後、/etc/pam.d/common-session ファイルに次のように 1 行追加します:

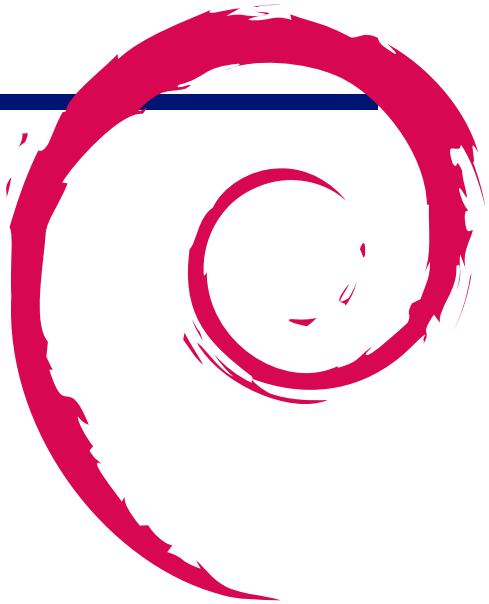
```
session optional                        pam_winbind.so
# end of pam-auth-update config
session    required    pam_mkhomedir.so skel=/etc/skel/ umask=0022 ←この行を追加
```

オプションの詳細は pam_mkhomedir(8) を参照してください。

Debian では pam-auth-update というコマンドで自動的に PAM の設定を行う方法もありますが、残念ながら上記の設定はサポートしていないため、手作業でファイルを修正する必要があります。

設定後、先ほどと同様に別のマシンから aduser02 でログインすると、次のように Creating directory ... というメッセージが表示されて、ホームディレクトリが自動作成されていることが確認できます:

```
$ ssh 192.168.135.35 -l W2K8R2AD3\\aduser02
W2K8R2AD3\aduser02@192.168.135.35's password:
Creating directory '/var/home/aduser02'.
Linux squeeze32-5 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 14 11:40:23 2013 from 192.168.135.16
W2K8R2AD3\aduser02@squeeze32-5:~$
```



7 索引

Debian 勉強会資料

2013 年 4 月 20 日　　初版第 1 刷発行

東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）