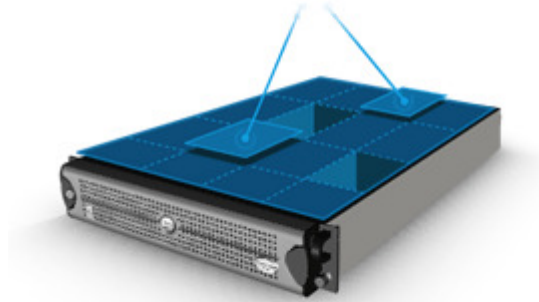


Virtual Private Asterisk

By Champ Clark (Da Beave) and John Lewis

Asterisk PBX Instances



Introduction: If you're not already familiar with Asterisk, it's primarily a Linux based PBX (Private Branch eXchange) that supports multiple telephone company related hardware cards, such as the Digium TE110P and X100P, plus Voice over IP (VoIP). (For more information, check out <http://www.asterisk.org>.)

Linux VServers (<http://linux-vservers.org>), sometimes known as Virtual Private Servers (VPS (VPSes)), are a way for a single Linux machine to run multiple instances of Linux. The basic idea is to "partition" off the system's resources. Each VPS acts like it's own independent machine. In fact, you can load different Linux distributions in each VPS. For example, your "host" system might be running Gentoo, while VPSes under the host system may also be running Gentoo or perhaps Slackware, RedHat/Fedora, Debian, etc.

Notes: (1.) We're using Gentoo with the 2.6 kernel. (2.) Obviously, sharing 'ztdummy' isn't the same as sharing physical hardware.

This article touches on two things...:

- 1.** Installing Asterisk in a nice, secure fashion. These steps can also be helpful in securing other applications as well, for example, a DNS server in a secure virtual environment. We're using Asterisk because it's primarily what we are interested in and it's a little tricky.
- 2.** Running multiple instances of Asterisk on a single machine. While doing research for this paper we have often encountered the daunting question, not once, but on several occasions...: "Why on Earth would you want to do something like that?" And, to answer this, we need to tell you about our little organization.

We are part of the Telephreak crew (<http://www.telephreak.org>). We're a small set of individuals that truly enjoy "hacking" away with Asterisk. We have access to several nice systems with ample bandwidth, primarily dedicated to our projects. Using the following steps in this document, and with the Asterisk ztdummy driver rather than physical hardware, we can give our members access to a virtual system. Each virtual system has 'root' access, granting its owners the ability to load and run Asterisk or anything else one may need.

One of our current systems we experiment with is a Dual 2.8 GHz Xeon system with 2 GB of RAM. The system typically runs about 15 instances of Asterisk. All things considered, the system is used for a lot more things, such as web, mail, SQL and DNS serving and it handles everything rather well. To date, the system currently has about 1330 processes, of which, 215 are Asterisk. Basically, we are running a fairly nice telephone company switch in a small box!

Whether you're an ISP that wants to run multiple instances of Asterisk as a value added benefit for your virtual private server users, or whether you just want to create a very secure Asterisk install, the directions to accomplish both are roughly the same. The primary "host" system runs a modified/patched kernel, and all other VPSes "piggy back" off that kernel as well. There are several advantages to this. For one, each VPS can

run at maximum performance the system will allow. Since it's one kernel for all VPSes and no hardware emulation, the slow-down of a VPS is effectively zero. On the security side of things, each VPS is contained within its own secure context. That is, a VPS does not have direct access to the host system kernel, unless you mistakenly give them access.

It also adds a layer of security. For example, VPS owners/users are unable to load their own modules or directly alter the kernel through `/dev/kmem`, or by any other means, even if they're "root". For the majority of applications people want to use in a VPS environment (Apache, Postfix, etc.), direct hardware access is rarely necessary, and this keeps the host system safe.

You can think of a VPS system as a 'chroot' type environment on steroids.

Using a VServer for a Secure Asterisk Installation:

Step 1 -- Getting Started

The first steps are setting up your system with the proper VPS patches, but that's outside of the scope of this document. There are plenty of HOWTO documents available online at the [Linux VServer site](#). It should be noted that creation of your VPS images should be carefully done. For example, you don't want to create any and all `/dev` devices. You'll carefully want to use only the skeleton devices supplied by the 'vserver' command. If possible, use the [PaX](#) and [grsecurity](#) patches. Under Gentoo, you can 'emerge vserver-sources' for the the default VServer patches with Pax/GRSec. (*Note: As of this writing, ebuids that use the 2.6.11 kernel have GRSecurity/PaX included while ebuids with the 2.6.12 kernel do not. This will probably change soon*). If you do decide to use GRSecurity, you'll want to disabled "Additional Restrictions" in the "Filesystem" section of GRSecurity. If you leave this enabled and attempt to start a VPS, the host won't be able to grant access to sections of the `/proc` file system for the VPS. I normally set the GRsecurity "Security Level" to "Custom" and mimic the "High" setting minus the "Additional Restrictions".

After our patched VServer + Pax + GRSec kernel is up and operational, we can go ahead and prepare the host system to use our Asterisk/Zaptel hardware. Yes, this includes the 'ztdummy' driver. The idea is that the host system will control the hardware access via a loadable module. We'll let our secure Asterisk install access the hardware from within the VPS. So, on the host system, we'll need to pull the Asterisk and Zaptel source code (and even LibPRI if you're using a card like the TE110P). I typically do this via CVS, which I suggest, but you can also use the 'tarball' files as well.

CVS Example:

```
# cd /usr/src
# mkdir asterisk
# cd asterisk
# export CVSR00T=:pserver:anoncv@s cvs.digium.com:/usr/cvsroot
# cvs checkout -r v1-0 zaptel libpri
```

This will grab the Zaptel and LibPRI stable source tree. Remove the "-r 1-0" if you're in the mood to play with the latest development tree.

And now, we start building...:

```
# cd zaptel
# make clean; make install
```

And, you can build the LibPRI if necessary...:

```
# cd libpri
# make clean; make install
```

After the Zaptel modules are built, simply load them up like you would do a normal Asterisk install, for example, "modprobe wct1xxp" (TE110P card) or whatever you require. Verify that the hardware is seen via "dmesg", and that there are no errors. We'll now need to setup our VPS image to support Asterisk.

Step 2 -- Modifying Your VPS Image

If you've followed the directions at the Linux VServer site on how to create your VPS image, you'll need to do some slight modifications. First off, you'll have to create the necessary Asterisk devices *within* the VPS environment. This can *only* be done *outside* of the VPS, not within it! If you attempt this within step *within* a VPS (not on the host system), the host kernel will not let you for security reasons. So, from outside that environment, you'll want to run the following script, called "vcreate"....:

```
#!/bin/bash

# This is a simple shell script to create normal/security device
# structures for normal VPSes, and extra support for Asterisk.

if [ "$1" == "" ]
then
echo
echo "You must supply the path of the VPS you want to install the"
echo "devices in! Also, make sure to remove ALL old devices from"
echo "the VPS. That is, 'cd /vservers/yourvpsname/dev && rm -rf *'"
echo
echo "To use, type:"
echo "$0 [VPS path] [--asterisk]"
echo
echo "Example: To install WITHOUT Asterisk devices, use:"
echo "$0 /vservers/yourvpsname"
echo
echo "Example : To install WITH Asterisk devices, use:"
echo "$0 /vservers/yourvpsname --asterisk"
exit
fi

mknod $1/dev/full c 1 7
mknod $1/dev/null c 1 3
mknod $1/dev/ptmx c 5 2
mknod $1/dev/random c 1 8
mknod $1/dev/urandom c 1 9
mknod $1/dev/tty c 5 0
mknod $1/dev/zero c 1 5
mkdir $1/dev/pts
chmod a+rw $1/dev/full $1/dev/null $1/dev/ptmx $1/dev/tty $1/dev/zero

mkdir -p $1/dev/vc

N=0; \
while [ $N -lt 65 ]; do \
mknod $1/dev/vc/$N c 4 $N; \
N=$((N+1)); \
done
chmod go-rwx $1/dev/vc/*

if [ "$2" == "" ]
then
echo "Okay, all done! (Note: Asterisk devices were NOT created!)"
```

```

        exit
    fi

# The next section is for Asterisk only!
mkdir -p $1/dev/zap
mknod $1/dev/zap/ctl c 196 0
mknod $1/dev/zap/timer c 196 253
mknod $1/dev/zap/channel c 196 254
mknod $1/dev/zap/pseudo c 196 255

N=1; \

while [ $N -lt 250 ]; do \
    mknod $1/dev/zap/$N c 196 $N; \
    N=$((N+1)); \
done

echo "Okay, all done! (Note: Asterisk devices were successfully installed!)"

```

Execution of "vcreate" is fairly straight forward. Typically, this is what I'll do. For the example, we'll say that /vserver/secureasterisk is the path to our pre-exploded VPS image.

```

# chmod a+rx vcreate
# rm -rf /vservers/secureasterisk/dev/*
#   (Yes, we're removing all devices.)
# ./vcreate /vservers/secureasterisk --asterisk

```

We now have proper Asterisk device support within our VPS to access our host system's Asterisk/Zaptel modules.

Step 3 -- Installing Asterisk Goodies

We now have a little more work to do to get Asterisk fully functional in our VPS environment. Technically as it stands, you could grab the Asterisk source code and get the basic functions up and running. However, functions like music on hold and conference calling will not yet work. In order to add this functionality, we'll go ahead and bring our VPS online (by using 'vserver secureasterisk start') and then enter the VPS environment (by using 'vserver secureasterisk enter'). Remember, we are now *within* the VPS environment!

```

# cd /usr/src
# mkdir asterisk
# cd asterisk
# export CVSROOT=:pserver:anoncvs@cvs.digium.com:/usr/cvsroot
# cvs checkout -r v1.0 asterisk zaptel

```

This will grab the v1.0 Asterisk source. You can also use Asterisk 'tarball' files as well. We'll now use the Zaptel sources to 'prep' the system.

```

# make libtonezone.so
# cp libtonezone.so /usr/lib/libtonezone.so.1.0 (maybe libtonezone.so.1.0)
# cp zaptel.h /usr/include/linux # Used for "meetme" detection.
# cp tonezone.h /usr/include
# cd /usr/lib
# ln -s libtonezone.so.1.0 libtonezone.so.1
# ln -s libtonezone.so.1.0 libtonezone.so
# ldconfig

```

You'll only need to do the above steps within the zaptel directory. You will *not* want to do a normal "make install". There's no point to it since you cannot load modules from within the VPS environment. The system will

generate an error if you even attempt a "make install". (It will 'blow up' when it attempts to create the Zap devices!)

Step 4 -- Installing Asterisk

At this point you can continue your normal Asterisk build and configuration. It would also be a fine idea to run Asterisk as another user other than 'root', just for extra security.

Step 5 -- Archiving

Once you're happy with your new VPS image, you might want to 'tar' it up for future VPS roll-outs, by utilizing 'tar -jcvpf'.

Results:

Using the above steps, you can accomplish two different types of tasks. Creating a very secure Asterisk install, or running multiple instances of Asterisk on one machine.

If you're using the steps to create a secure Asterisk installation, you are now keeping the kernel/host system secure by creating a new context (VPS) for Asterisk to run inside. Once you have a working 'secure' VPS image, you can further reduce security risks by stripping the VPS image to a bare minimum set of libraries and binaries needed. At this point, if an attacker finds a vulnerability in Asterisk or the VPS itself, and uses it to penetrate the system, they'll be within a 'jailed' environment. If Asterisk was exploited, they'd find themselves in a VPS as the user running Asterisk (hopefully the user 'asterisk', and not 'root'). If the attacker gains 'root' access within the VPS, they won't have the abilities to create devices, load modules, or have direct access to the host kernel. Using utilities at the host level like 'aide' and 'chkrootkit', detecting 'backdoors', altered binaries and libraries is trivial. Rebuilding a compromised VPS is simple; repair the problems and re-roll out the image.

If you're using physical hardware rather than emulated hardware (ztdummy), then the only limitation is the device you're using. Obviously, you can't share a single channel FXO card (X100P, for example) between multiple VPS's. Once that channel is used by a VPS, it is tied to it. If two or more VPS's try to allocate the same channel on a card, one VPS will get it and the others will get a "device busy" error.

Cards with multiple channels (for example, Digium's TE110P, TDM400P, etc) can be shared, as long as you don't attempt to share the same channels. Let's say you have a full DS1 (T1) with a TE110P card, this give you 24 channels and you can configure the host system to use all those channels. The configuration for this is in the /etc/zaptel.conf. Once the card is up on the host system with all channels, you can then allocate channels on a per VPS basis. You might have VPS #1 use channels 1-12 while VPS #2 might use 13-24. Within the VPS's, you tell Asterisk the channels to use by editing the /etc/asterisk/zapata.conf. For VPS #1, you'd have a entry "channels => 1-12". In VPS #2, you'd have "channels => 13-24".

With a TE110P with 24 channels, you can have a maximum of 24 VPSs accessing a single channel each. The same can be done with any multi-channel card.

For an ISP wanting to sell VPSs running Asterisk with DS1 channel access, there is one major down fall. I haven't found a reasonable way to "secure" channels. For example, if VPS #1 goes down and isn't using channels 1-12, there is nothing to stop the operator of VPS #2 from grabbing those unused channels.

Fortunately, the ztdummy driver handles hardware emulation in different manner. The thing to remember is the ztdummy driver only supplies timing for applications like MoH (music on hold) and meetme (conferencing). The ztdummy driver never actually handles or controls inbound or outbound channels. If you're using the ztdummy driver and not addressing physically hardware, the traffic will be VoIP. This traffic is handled by the Asterisk within each VPS. This means that if channel #1 is being used on VPS #1, VPS #2 will not have direct access to it. This keeps things nice, separate and secure between VPS using the ztdummy driver.

I should note that under the Linux 2.6 kernel, we did have some 'lag' issues using the ztdummy driver and IAX2 (VoIP) channels. To correct this, you can apply the [ztdummy RTC patches](#). These only apply to the "stable" Asterisk zaptel source.

If the traffic is VoIP with in the VPS (ztdummy or not), it is isolated within that VPS. If you're using hardware, isolation is a bit more difficult.

Interesting Links: Softwink, Inc. "Install & Securing VoIP With Linux"
http://www.softwink.com/papers/Installation_Securing_VoIP_With_Linux/

Credits: This document was authored by Champ Clark ('Da Beave'). Champ works at Vistech Communications, Inc. providing network support and application development. Champ is also employed with Softwink, Inc. as a network security analyst and researcher. (You can visit Vistech Communications, Inc. on the web at <http://www.vistech.net> and Softwink, Inc. at <http://www.softwink.com>.) Of course, Vistech offers Asterisk enabled VPSes

This document was edited by John D. Lewis. John is self employed as a consultant and a technical writer. John's work includes central office engineering, outside plant supervision, network design and maintenance, and now VoIP technology. John is also working on several technical titles which he hopes to have printed. (John's site can be found on the web at <http://www.jdlewis.org>.)

Both Champ, John, and other talented individuals who have helped with this project can be found on Telephreak's conference bridge (see <http://www.telephreak.org> for details) and on Telephreak's IRC network at <irc.telephreak.org>, channel #telephreak.

Special thanks to Bruce 'Corky' Wink for extra review.