SANDBOXING

BY KURT SEIFRIED

Unknown and Untrusted

If you're like me, you love to test new software, and therein lies one of the huge advantages of the open source world. Almost everything is just a short *wget, ./configure; make; make install* away, and there's no need to pay, register, provide personal information, wait a week for the CD to arrive, and so forth. But how can you be certain that the software won't interfere with your system, overwrite something, or otherwise behave badly?

Or what if you want to run a web service that you know has a history of problems allowing for remote code execution on the web server?

Sandboxing

A common programming and system administration technique is to use sandboxes, which essentially are restricted areas for the software (or in some cases, an entire operating system or group of systems) to run where it can't interfere with production systems. By setting up a walled-off testing area, you know that if anything does go wrong, it is less likely to cause severe problems, such as affecting your real file server or web server. Additionally, it is easier to observe and verify the behavior of the software because there is less going on within the sandbox.

This leads to the two main requirements of a sandbox: You need to be able to isolate the software, and you need to be able to monitor what the software is doing and control it.

Fortunately, over the past few years, a number of advancements in computing have made the first requirement much easier to meet. Faster CPUs, larger hard disks, and cheap memory, combined with widespread virtualization software, now mean that almost anyone with a recent computer – at least 1-2GHz and 512MB of RAM – can easily run at least one entire operating system on top of their existing operating system.

Unfortunately, many of these products do not address the second requirement very well, with many either requiring the virtualized operating system (also known as the guest) to be modified significantly or to use virtual files to hold the hard-drive contents for the guest.

Sandboxing an OS with VMware Server

The good news is that VMware Server is free to download and use. The bad news is that it is a closed source product. Please note that I haven't covered all the possible options, such as Bochs [1], Xen [2], User-Mode Linux [3], VirtualBox [4], KVM [5], OpenVZ [6], QEMU [7], etc.) because there are simply too many to fit within the pages of this article.

Additionally, I like VMware Server [8] because it only requires a few kernel modules (*vmnet*, *vmmon*) and can run almost any operating system as a guest without any modifications to the guest operating system.

Installation is relatively straightforward: You simply download and unpack the file and run the *vmware-config.pl* script. After you answer a few quick questions, you are ready to run. The major downside to VMware Server is that it uses disk-based image files for the guest operating system, so to examine the "hard drive" for the guest operating system, you will either need to stop or suspend it and then mount the disk image (Listing 1).

The advantage is that you can literally stop an operating system in its tracks, examine a frozen snapshot of it at your leisure, then resume it when you're done.

Sandboxing an Application with chroot

Sometimes, however, sandboxing an entire operating system is overkill. What if you just want to compile some software and install it without affecting your current system or give yourself the option of easily removing the software? Oddly enough, this is the exact same challenge that Bill Joy ran into while working on BSD back in the 1980s. His solution was to create the chroot system call and utility program.

With chroot, you must remember one critically important thing: chroot was **not** meant to be a security mechanism. Instead, it was designed to make software testing and installation easier and safer. A process or a user with root privileges can easily break out of a chroot environment and cause damage to the underlying operating system. However, this

can largely be mitigated by running all software within the chroot as a non-root user and removing any potentially unsafe *setuid* binaries that run as root or with otherwise elevated privileges.

Building a chroot Environment

On RPM- and Debian DPKG-based systems, building a chroot environment is relatively easy. Some people will accuse me of being RPM-centric, and they'd be correct – I started with Slackware 1.0, but I switched after seeing Red Hat 3.0.3 and have been using Red Hat and CentOS ever since.

Also, Debian has documented the process of building a chroot environment properly, so I do not need to repeat it here [9].

To build a complete chroot environment, you need several basic items:

- a file system with some basics, such as */dev/* and */proc/* (so that things like *ps* will work);
- any programs and libraries needed to run the software you want to test; and,
- optionally, an easy way to install or update software within the chroot, which is especially important if you want to use the chroot as a production environment to compartmentalize software).

Listing 1: Mount the Disk Image

```
01 # vmware-mount.pl -p Centos.
  vmdk
02
03 Nr      Start       Size Type
  Id System
04 -- ---------- ---------- ----
  -- ------------------------
05  1        63     208782 BIOS
  83 Linux
06  2    208845     530145 BIOS
  82 Linux swap
07  3    738990   20225835 BIOS
  83 Linux
08
09 # vmware-mount.pl Centos.vmdk
  3 /mnt/vmware/
10
11 # df
12 Filesystem          1K-blocks
  Used Available Use% Mounted on
13 /dev/nb0              9796164
  2548372   6742148  28% /mnt/
  vmware
```

Step 1: Basic File System

Here, I use */chroot* as the chroot base directory. As root, execute:

```
# mkdir /chroot
# mkdir /chroot/proc
# mkdir /chroot/dev
# mount -t proc ➥
proc /chroot/proc
# /sbin/MAKEDEV generic -D ➥
/chroot/dev -d /chroot/dev
```

Step 2: Prep chroot for yum Usage

Installing the release package (e.g., *centos-release*, *redhat-release*) into the chroot will let yum work:

```
# rpm -Uvh --nodeps ➥
--root=/chroot/ ➥
centos-release-5-1.0.el5.➥
centos.1.x86_64.rpm
```

Step 3: Install into the chroot

The RPM also installs the software into the chroot, but yum will handle dependencies and make things much simpler:

```
# yum --installroot=/chroot/➥
install bash yum vim-minimal
```

At a minimum, I recommend a shell (bash), yum to install software, and the vim editor to modify files in the chroot.

Step 4: Network Configuration Files

If you want to access the network from within the chroot, you need a *resolv.conf* file (lets applications know where your DNS servers are to be found) and:

```
# mkdir /chroot/etc/
# mkdir /chroot/etc/sysconfig
# cp /etc/resolv.conf ➥
/chroot/etc/
# cp /etc/sysconfig/network ➥
/chroot/etc/sysconfig/
```

"Logging" In to a chroot

At this point, you'll be able to access the chroot with a command such as *$ chroot /chroot/ bash*, which will chroot you into the */chroot/* directory and execute bash from within it.

As I mentioned, chroot is not an inherently secure method for isolating applications. By not logging into the chroot as a privileged user such as root, and by removing any *setuid* and *setgid* binaries that run with elevated privileges, you can ensure that nothing runs as root within the chroot environment:

```
# find / -type f -perm +6000
```

Conclusion

Sandboxing is now easier than ever and its benefits have never been more important. Isolating badly written web applications from the underlying operating system or letting an administrator install a program without affecting the system can save both time and money. Like anything, prevention and foresight can significantly reduce the amount of work needed to maintain and fix a system long term, and sandboxing offers a practical tool to accomplish this. ■

INFO

[1] Bochs: http://bochs.sourceforge.net/

[5] KVM: http://kvm.qumranet.com/kvmwiki

[6] OpenVZ: http://openvz.org/

[7] QEMU: http://fabrice.bellard.free.fr/qemu/

[3] User-Mode Linux: http://user-mode-linux.sourceforge.net/

[8] VMware Server: http://www.vmware.com/products/server/

[4] VirtualBox: http://www.virtualbox.org/

[2] XEN: http://www.xensource.com/

[9] Debian chroot instructions: http://www.debian.org/doc/manuals/reference/ch-tips.en.html#s-chroot

[10] FreeVPS: http://www.freevps.com/

[11] Linux-VServer: http://linux-vserver.org/

[12] AppArmor: http://www.novell.com/linux/security/apparmor/

[13] SELinux: http://www.nsa.gov/selinux/

THE AUTHOR

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He is married and has four cats but no fish (because the cats are more hungry than afraid of water). He often wonders how it is that technology works on a large scale but often fails on a small scale.