Gentoo Hardened SELinux Testing

COLLABORATORS			
	TITLE : Gentoo Hardened SELinux Testing		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Sven Vermeulen	February 14, 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

As an active contributor to the Gentoo Hardened project with regard to SELinux support (both documentation, policy development and ebuilds) I find it important to have a good testing approach. Thanks to virtualization efforts such as KVM, it is possible to generate a semi-automated testing environment, allowing me to test various setups and installations before considering contributions stable enough.

Of course, why using semi-automated testing rather than fully automated? Well, I'm probably going to automate more and more to the point it is fully automated, but for the time being, it is a semi-automated approach ;-)

1.1 Testing Overview

The following table shows the tests that I execute. Most tests are based on the documentation offered by Gentoo, which also means that I occasionally have to review the scripts to see if they still follow the instructions (as documentation changes over time).

Test Name	Test Description
Stage3 hardened installation	Perform a blank hardened installation, using the latest stage3-hardened-amd64 snapshot (in the autobuilds directory). Follow the instructions of the Gentoo Handbook closely.
SELinux conversion	Convert the hardened installation to a SELinux, following the instructions in the Gentoo Hardened SELinux handbook. Enable SELinux enforcing/strict mode and perform a few standard tests.
MySQL installation and testing	Install MySQL following the MySQL guide and perform a few standard tests.

1.2 Testing Architecture

To enable semi-automated testing, I run a simple webserver which provides read access to:

- the host distfiles (URL: /gensetup/gentoo/distfiles)
- the stage3 tarball, portage snapshot, Linux kernel binary package and installation script (URL: /gensetup/gentoo)
- various binhosts for different installation types (URL: /gensetup/gentoo/binhost)

The tests all run inside KVM guests (with network access). The idea is as follows:

1. Perform a standard stage3 installation, call that "hardened-base".
2. Make a copy-on-write from "hardened-base" and use that one to further test hardened setups, call that "hardened"
3. Make a copy-on-write from "hardened-base" and use that one to test the SELinux installation instructions, call that "selinux-base"
4. Make a copy-on-write from "selinux-base" and use that one to further test various SELinux setups, call them "selinux"

2 Stage3 Hardened Installation

This test performs a hardened installation in a virtual environment. At the end, we should have a bootable virtual environment with a small but functional hardened system (with pax/grsecurity hardening active) inside.

2.1 Information

The stage3 hardened installation uses a 20Gbyte qemu image (qcow2 format):

```
~$ qemu-img create -f qcow2 test.img 20G
```

This image is booted using the system rescue CD ISO:

```
~$ qemu-system-x86_64 --enable-kvm -net nic,model=virtio,macaddr=00:11:22:33:44:a0,vlan=0 \
 -net vde,vlan=0 -drive file=test.img,if=virtio,boot=on -usb -usbdevice tablet -smp 2 \
 -k nl-be -m 1024 -boot d -cdrom systemrescuecd-x86-2.0.0.iso
```

The virtual drive (vda) is formatted with a root partition (vda1, ext4), a swap partition and a physical group (vda3) for LVM2. A volume group is then created with usr, home, opt and var as logical volumes.

```
~# df -hT
Filesystem            Type    Size  Used Avail Use% Mounted on
/dev/vda1             ext4    2.0G  424M  1.5G  23% /
udev                 tmpfs     10M  104K  9.9M   2% /dev
shm                  tmpfs    498M     0  498M   0% /dev/shm
/dev/mapper/vg-usr    ext4    9.9G  2.2G  7.3G  23% /usr
/dev/mapper/vg-home   ext4    2.0G   67M  1.9G   4% /home
/dev/mapper/vg-opt    ext4    2.5G   68M  2.3G   3% /opt
/dev/mapper/vg-var    ext4    2.5G  128M  2.3G   6% /var
tmpfs                tmpfs    498M     0  498M   0% /tmp
```

During the Gentoo installation, the following settings are used.

Setting	Description
Profile = hardened/linux/amd64	The Portage profile selected during the installation is the "hardened/linux/amd64" profile.
USE = -ldap -gtk -xorg -ipv6 -pppd	The USE flags are modified slightly to keep the number of package dependencies to a minimum.
clock/TIMEZONE = Europe/Brussels	The timezone is set to Europe/Brussels
keymaps/KEYMAP = be-latin1	The keymap is set to be-latin1
net/config_eth0 = ("dhcp")	The network configuration uses DHCP
/etc/locale.gen only en_US (ISO-8859-15 and UTF-8)	The number of locales to be generated by glibc installation is limited to the en_US ones.

Also, the following packages are installed:

- mdadm (as I hope to extend the installation later to use two virtual disks in software RAID-1 configuration)
- lvm2 (as the installation uses LVM2)
- syslog-ng (system logger), this one is also added to the default runlevel. During the installation, "**unset path**" is executed (otherwise the installation of syslog-ng fails).
- dhcpcd (DHCP client)
- grub (bootloader)
- layman (to add overlays)
- vim (personal favorite editor)
- git (for client)
- eix (portage package information caching)
- hardened-sources (kernel sources)

The kernel configuration/build itself is premade (occasionally, the Linux kernel build is done and a binary package is created which can then be reused across all virtual instances) so is not actually tested.

The method to automatically perform this installation is scripted and available from http://github.com/sjvermeu/small.coding/gensetup.

2.2 Deviations

A few deviations are in place from a standard installation. These deviations are occasionally cleared

2.3 Steps

Set up the qemu image (20Gbyte);

```
~$ qemu-img create -f qcow2 test.img 20G
```

Boot the image with the sysresccd livecd:

```
~$ qemu-system-x86_64 --enable-kvm -net nic,model=virtio,macaddr=00:11:22:33:44:a0,vlan=0 \
 -net vde,vlan=0 -drive file=test.img,if=virtio,boot=on -usb -usbdevice tablet -smp 2 \
 -k nl-be -m 1024 -boot d -cdrom systemrescuecd-x86-2.0.0.iso
```

Download the installation scripts:

```
~# wget http://192.168.100.1:8080/gensetup/gentoo/gensetup.sh
~# wget http://192.168.100.1:8080/gensetup/gentoo/simple.conf
~# chmod +x gensetup.sh
```

Execute the gensetup.sh script to automatically setup the Gentoo Hardened installation:

```
~# ./gensetup.sh
```

Finally, reboot the virtual system.

2.4 Additional Tests

No additional tests are performed right now.

3 SELinux Conversion Installation

In this test, we convert a fresh hardened installation to SELinux.

3.1 Information

To convert the system, we follow the instructions of the draft SELinux handbook closely, being:

1. Installing layman and adding the hardened-development overlay
2. Update the /tmp context to tmp_t (instead of tmpfs_t)
3. Enable a couple of ~arch packages
4. Switch the Gentoo profile to a SELinux hardened one
5. Perform the "manual" changes (automatically now ;-)
6. Upgrade the linux-headers and rebuild glibc

7. We skip the SELinux kernel installation as our kernel is already SELinux aware
8. We update fstab to include /selinux
9. After rebooting, we install the SELinux utilities in the correct order
10. We set our target policy to "strict"
11. We relabel the device files and then the entire filesystem
12. After rebooting, we set the global_ssp boolean to on

After all this, we're set.

3.2 Deviations

Currently, the only deviation I have is that the Linux kernel booted with is already SELinux aware, so this step is skipped.

3.3 Steps

Setup the BINHOST to the selinux-base binhost:

```
~# vim /etc/make.conf
PORTAGE_BINHOST="http://192.168.100.1:8080/gensetup/gentoo/binhost/selinux-base"
```

Download the update_selinux.sh and master.lib.sh files:

```
~# wget http://192.168.100.1:8080/gensetup/gentoo/update_selinux.sh
~# wget http://192.168.100.1:8080/gensetup/gentoo/master.lib.sh
~# chmod +x update_selinux.sh
```

The update script will ask to reboot the system twice and tell you to continue then with the next step:

```
~# touch EMPTY
~# ./update_selinux.sh EMPTY
After the first reboot:
~# ./update_selinux.sh EMPTY label
After the second reboot:
~# ./update_selinux.sh EMPTY booleans
```

The EMPTY file is because the script requires a configuration file to be passed, even though for this script no configuration file is needed. This won't be necessary in future versions of the script.

3.4 Additional Tests

The next few tests are needed to validate if everything works properly:

1. Reboot the virtual environment, add "enforcing=1" as kernel parameter to boot in SELinux mode.

   This should work flawlessly.

2. Edit the /etc/selinux/config file to use enforcing/strict mode. Then, update portage (emerge --sync; layman -S) and update world.

   This too should work flawlessly.

4 Appendix: Facilitating Using BINHOST

Portage has the ability to use BINHOSTs for fast deployments.

Within the test setup, BINHOST can be used to improve build speeds. However, it is important to occasionally clear the BINHOST so that the build process is tested too.

4.1 Feeding the BINHOST

A binhost should be based upon a Gentoo Portage profile + make.conf setting (including USE flags). The moment this changes, a new BINHOST needs to be set up.

To set up a BINHOST:

1. Perform an entire installation up to the point that it is either finished, or that the profile or make.conf changes.
2. Run quickpkg --include-config=y --include-unmodified-config=y for all packages (hint: use /var/db/pkg or, even better, emerge.log output)

```
quickpkg --include-config=y --include-unmodified-config=y $(grep "completed emerge.* to ↩
    /" /var/log/emerge.log | awk '{print "="$8}')
```

3. Upload the entire /usr/portage/packages location to the BINHOST

In the future, you can refer to this BINHOST using PORTAGE_BINHOST.