How to Setup Additional Entropy for Cloud Servers Using Haveged

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=273

Brief Introduction to Entropy and Randomness

The Linux pseudo random number generator (PRNG) is a special device that generates randomness from hardware interrupts (keyboard, mouse, disk/network I/O) and other operating system sources. This randomness is used mostly for encryption like SSL/TLS, but also has many other uses. Even something as simple as a program to roll a pair of virtual dice depends on entropy for good quality randomness.

When Entropy Pools Run Dry

There are two general random devices on Linux: /dev/random and /dev/urandom. The best randomness comes from /dev/random, since it's a blocking device, and will wait until sufficient entropy is available to continue providing output. Assuming your entropy is sufficient, you should see the same quality of randomness from /dev/urandom; however, since it's a non-blocking device, it will continue producing "randomâ€• data, even when the entropy pool runs out. This can result in lower quality random data, as repeats of previous data are much more likely. Lots of bad things can happen when the available entropy runs low on a production server, especially when this server performs cryptographic functions. For example, let's say you have a cloud server running the following daemons (all using SSL/TLS or block ciphers):

- Web Server
- Incoming/Outgoing Mail Server
- SSH/SFTP

Should any of these daemons require randomness when all available entropy has been exhausted, they may pause to wait for more, which can cause excessive delays in your application. Even worse, since most modern applications will either resort to using its own random seed created at program initialization, or to using /dev/urandom to avoid blocking, your applications will suffer from lower quality random data. This can affect the integrity of your secure communications, and can increase the chance of cryptanalysis on your private data.

The Userland Solution for Populating Entropy Pools

Linux already gets very good quality random data from the aforementioned hardware sources, but since a headless machine usually has no keyboard or mouse, there is much less entropy generated. Disk and network I/O represent the majority of entropy generation sources for these machines, and these produce very sparse amounts of entropy. Since very few headless machines like servers or cloud servers/virtual machines have any sort of dedicated hardware RNG solution available, there exist several userland solutions to generate additional entropy using hardware interrupts from devices that are "noisierâ€• than hard disks, like video cards, sound cards, etc. This once again proves to be an issue for servers unfortunately, as they do not commonly contain either one. Enter haveged. Based on the HAVEGE principle, and previously based on its associated library, haveged allows generating randomness based on variations in code execution time on a processor. Since it's nearly impossible for one piece of code to take the same exact time to execute, even in the same environment on the same hardware, the timing of running a single or multiple programs should be suitable to seed a random source. The haveged implementation seeds your system's random source (usually /dev/random) using differences in your processor's time stamp counter (TSC) after executing a loop

repeatedly. Though this sounds like it should end up creating predictable data, you may be surprised to view the FIPS test results in the bottom of this article.

Installing haveged on Debian/Ubuntu

You can easily install haveged on Debian and Ubuntu by running the following command:

```
# apt-get install haveged
```

Should this package not be available in your default repositories, you will need to compile from source (see below)

Once you have the package installed, you can simply edit the configuration file located in /etc/default/haveged, ensuring the following options are set (usually already the default options):

```
DAEMON_ARGS="-w 1024"
```

Finally, just make sure it's configured to start on boot:

```
# update-rc.d haveged defaults
```

Installing haveged on RHEL/CentOS/Fedora

To install haveged on RHEL/CentOS (skip this step for Fedora), you first need to add the EPEL repository by following the instructions on the official site. Once you've installed and enabled the EPEL repo (on RHEL/CentOS), you can install haveged by running the following command:

```
# yum install haveged
```

Fedora users can run the above yum install command with no repository changes. The default options are usually fine, so just make sure it's configured to start at boot:

```
# chkconfig haveged on
```

Installing from Source

On systems where there simply isn't any pre-packaged binary available for haveged, you will need to build it from the source tarball. This is actually much easier than you might expect. First, you will visit the download page and choose the latest release tarball (1.7a at the time of this writing). After downloading the tarball, untar it into your current working directory:

```
# tar zxvf /path/to/haveged-x.x.tar.gz
```

Now you compile and install:

```
# cd /path/to/haveged-x.x
```

```
# ./configure
# make
# make install
```

By default, this will install with a prefix of /usr/local, so you should add something similar to the following to /etc/rc.local (or your system's equivalent) to make it automatically start on boot (adjust the path if necessary):

```
# Autostart haveged
/usr/local/sbin/haveged -w 1024
```

Run the same command manually (as root) to start the daemon without rebooting, or just reboot if you're a Windows-kinda-guy.

Testing Availability of Entropy & Quality of Random Data

After some very minimal installation/configuration work, you should now have a working installation of haveged, and your system's entropy pool should already be filling up from the randomness it produces. Security wouldn't be security if you blindly trusted others and their claims of effectiveness, so why not test your random data using a standard test? For this test, we'll use the FIPS-140 method used by rngtest, available in most or all major Linux distributions under various package names like rng-tools:

```
# cat /dev/random | rngtest -c 1000
```

You should see output similar to the following:

```
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
rngtest: starting FIPS tests...
rngtest: bits received from input: 20000032
rngtest: FIPS 140-2 successes: 999
rngtest: FIPS 140-2 failures: 1
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 1
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=1.139; avg=22.274; max=19073.486)Mibits/s
rngtest: FIPS tests speed: (min=19.827; avg=110.859; max=115.597)Mibits/s
rngtest: Program run time: 1028784 microseconds
```

A very small amount of failures is acceptable in any random number generator, but you can expect to see 998-1000 successes very often when using haveged. To test the amount of available entropy, you can run the following command:

```
# cat /proc/sys/kernel/random/entropy_avail
```

The idea of haveged is to fill this pool back up whenever the available bits gets near 1024. So while this number will fluctuate, it shouldn't drop below 1000 or so unless you're really demanding lots of randomness (SSH key generation, etc).