Collecting Linux RHEL/CentOS Mass Storage Device Entropy

By Kirill Sinitski and Mike Ounsworth

Mass storage device entropy source or `add_disk_randomness()` is one of the sources of randomness continuously feeding Linux Pseudorandom Number Generator (LPRNG). It is utilized as `/dev/random` and `/dev/urandom` devices. The exact implementation is in `/drivers/char/random.c`

Installing and configuring OS

The easiest way to install CentOS7 is to burn ISO image on optical media and boot from it. However, it is likely you will want to collect entropy of your existing system. Obviously, under no circumstances, do any of this on a production system. This guide does NOT contain steps to undo these changes. When creating this document we used minimal install with no add-ons to make sure all dependencies are satisfied. However, according to Murphy's law *your* version will require additional work.

Configuring users

Add non-root user and grant it sudo powers.

```
[root@host]# useradd user
[root@host]# passwd user  # set password for user
[root@host]# usermod -aG wheel user # grant sudo privileges
```

Enable Network Connectivity

The easiest way to install missing packages is using yum from a repository. For this to work, you will need internet access. If you already have this working on your setup, it is safe to skip these next steps.

Assuming working DHCP server in the environment, here is how to enable network connectivity on basic install using command line. First, connect an Ethernet cable to the network card port. Disconnected ports are flagged `NO-CARRIER`. Second, figure out the device name that linux has assigned to this port:

```
[user@host]$ # ip link
1: p1p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    UP mode DEFAULT qlen 1000
    link/ether 00:a1:ba:51:4c:11 brd ff:ff:ff:ff:ff:ff
2: p1p2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast
    state DOWN mode DEFAULT qlen 1000
    link/ether 00:a1:ba:51:4c:12 brd ff:ff:ff:ff:ff:ff
```

Port we used showed up as `p1p1`. You can tell it apart from other ports by state UP and lack of NO-CARRIER indication.

The next step is to edit corresponding configuration file, in this case it is `ifcfg-p1p1`.

```
[user@host]$ sudo vi /etc/sysconfig/network-scripts/ifcfg-p1p1
    # change ONBOOT=yes
[user@host]$ sudo systemctl restart network
```

Note: In vi, use Shift + : to enter command mode, then use wq! to save changes.

Optionally, test connection ping something.

```
[user@host]$ ping 8.8.8.8          # Google's name server
```

Install missing packages

For this step, we are following the guide at https://wiki.centos.org/HowTos/I_need_the_Kernel_Source.

```
[user@host]# sudo yum install -y rpm-build redhat-rpm-config asciidoc
hmaccalc perl-ExtUtils-Embed pesign xmlto
[user@host]# sudo yum install -y audit-libs-devel binutils-devel elfutils-
devel elfutils-libelf-devel
[user@host]# sudo yum install -y ncurses-devel newt-devel numactl-devel
pciutils-devel python-devel zlib-devel
```

Some additional packages that had to be installed when using minimal install:

```
[user@host]# sudo yum install -y gcc bc perl net-tools bison
[user@host]# sudo yum groupinstall "Development Tools"
```

Downloading CentOS Kernel Source

Download the source code from CentOS or RHEL depository.

Note: Red Hat / CentOS uses a modified kernel, you need to fetch the kernel from the appropriate mirror. Getting a vanilla Linux kernel to compile is much more complicated and is outside the scope of this guide.

```
 [user@host]$ rpm -i
http://vault.centos.org/7.2.1511/updates/Source/SPackages/kernel-3.10.0-
327.28.3.el7.src.rpm 2>&1 | grep -v exist
```

Note: We used kernel version `3.10.0-327.28.3.el7` to write this guide, your exact version will be different. Make sure to use matching version of the source code, as during compiling we re-use existing system configuration and modules. This approach may stop working if you mismatch kernel versions.

Unpack the Kernel Source

First, create the directory structure:

```
 [user@host]$ mkdir -p
~/rpmbuild/{BUILD,BUILDROOT,RPMS,SOURCES,SPECS,SPRMS}
```

```
[user@host]$ echo '%_topdir %(echo $HOME)/rpmbuild/' > ~/rpmmacros
```

Next, unpack the rpm containing source code:

```
[user@host]$ cd ~/rpmbuild/SPECS
[user@host]$ rpmbuild -bp --target=$(uname -m) kernel.spec
```

Optionally, verify that it unpacked, check that the following directory is no longer empty:

```
[user@host]$ ls ~/rpmbuild/SOURCES
```

Expand (untar) the kernel source:

```
[user@host]$ cd ~/rpmbuild/SOURCES
[user@host]$ tar xJf linux-3.10.0-327.28.3.el7.tar.xz linux-3.10.0-
327.28.3.el7
```

Note: In this case, `linux-3.10.0-327.28.3.el7` is the destination directory containing the source code and `linux-3.10.0-327.28.3.el7.tar.xz` is the source code that was uploaded.

Implementing test harness

Copy over the test harness files

Place the files `random.c.harness.3.10` and `Reboot.sh` on a USB stick.

Note: RHEL7 uses XFS file system, but will recognize FAT32.

Mount the USB stick

First, determine what device name was assigned to your USB stick. There are various "proper" ways to do this, but the quick hack we used was as follows:

```
[user@host]$ ls /dev/sd*
```

Observe which devices are present in the list that approximately looks like this:

```
sda  sda1  sda2  sdb  sdc  sdc1
```

Insert the USB stick into a USB port and repeat:

```
[user@host]$ ls /dev/sd*
```

Note which device was added - this should be your USB. In our case it was `/dev/sdc1`.

Mount `/dev/sdc1 and` copy the data:

```
[user@host]$ sudo mkdir /media/usb
[user@host]$ sudo mount /dev/sdc1 /media/usb
```

Copy over files into your home directory:

```
[user@host]$ sudo cp /media/usb/* ~/
```

When finished, unmount it:

```
 [user@host]$ sudo umount /media/usb
```

Optionally, check file permissions:

```
[user@host]$ ls -l ~/
```

Note: depending on the type of filesystem of your computer and the USB stick uses, you may need to look at the permissions of those files (`ls -l`) and fix them (`chmod` and / or `chown`).

Install test harness

Copy file containing modified `random.c` containing test harness into kernel directory, in our case `random.c.harness.3.10` into `linux-3.10.0-327.28.3.el7` directory:

```
[user@host]$ cp ~/random.c.harness.3.10 ~/rpmbuild/SOURCES/linux-3.10.0-
…/drivers/char/random.c
```

Re-tar (repack) the kernel source:

```
[user@host]$ cd ~/rpmbuild/SOURCES/
[user@host]$ tar cJf linux-3.10.0-….tar.xz linux-3.10.0-…
```

Note: This process will take considerable time.

Modify kernel identifier

The following steps are based on CentOS HowTos/Custom Kernel Wiki.

```
[user@host]$ cd ~/rpmbuild/SPECS/
[user@host]$ vi kernel.spec
```

Modify `buildid` variable to identify the build:

```
%define buildid .Test_Build
```

Note: There should be no space between the “%” and the word “define”.

Build the modified kernel

Build kernel as a user.

```
[user@host]$ cd ~/rpmbuild/SPECS/
[user@host]$ rpmbuild -bb --target=`uname -m` kernel.spec 2> build-err.log
| tee build-out.log
```

Note: `` `uname -m` `` will substitute CPU architecture, for example x86_64. ` used here is the same key as ~.

If something goes wrong, like unsatisfied dependency, check the log file for details:

```
[user@host]$ vi build-err.log
```

Note: If you have any custom patches applied, you have to perform additional steps to enable them.

Install the modified kernel

```
[user@host]$ cd ~/rpmbuild/RPMS/’uname -m’
[user@host]$ sudo yum localinstall kernel-3.10.0-…
```

Note: This process will take considerable time.

Reboot:

```
[user@host]$ sudo reboot now
```

Optionally, verify you booted into modified kernel.

```
[user@host]$ uname -a
     Linux hostname 3.10.0-327.28.3.el7.x86_64 #1 SMP Sat Mar 2 14:36:21
     EST 2014 x86_64 x86_64 GNU/Linux
```

Optionally, verify test harness is working.

```
[user@host]$ journalctl -k -p 7 | grep DISK_RANDOMNESS | wc -l
```

Note: Non-zero number indicates harness is working.

Collecting Data

The script we provide `Reboot.sh` is meant to be inserted into the system's startup scripts. It will capture all the kernel output from our harness in a data file and reboot the system until either a certain number of datapoints has been captured, or a certain amount of time has elapsed. RHEL7 does not retain `KERN_DEBUG` level of events past reboot. They are stored only in RAM.

Setting up the data collection / reboot script

In a previous step, you should have copied the script from USB to ~/Reboot.sh. Make sure it has execute permissions. There are many ways to have this script run as part of the boot sequence. We are aware this is an older method, but it is simple and still works.

Enable `rc.local` and add the reboot script to it:

```
[user@host]$ sudo chmod +x /etc/rc.d/rc.local
     # enable the older-style boot script

[user@host]$ sudo vi /etc/rc.local
     # add this line to the end:
     sh /home/user/Reboot.sh
```

Optionally, to test that everything is configured correctly, use:

```
[user@host] sudo sh /home/user/Reboot.sh -test
```

Copying experimental data off the machine

Mount USB drive as described elsewhere in the guide.

Create a folder:

```
[user@host]$ mkdir DirName
```

Copy files:

```
[user@host]$ cp entropyExperiment.data ~/media/usb/DirName
```

Unmount.