Red Hat Enterprise Linux 7

Kernel Administration Guide

Examples of Tasks for Managing the Kernel

Last Updated: 2018-05-21

Red Hat Enterprise Linux 7 Kernel Administration Guide

Examples of Tasks for Managing the Kernel

Marie Dolezelova
Red Hat Customer Content Services
mdolezel@redhat.com

Mark Flitter
Red Hat Customer Content Services

Douglas Silas
Red Hat Customer Content Services

Eliska Slobodova
Red Hat Customer Content Services

Jaromir Hradilek
Red Hat Customer Content Services

Maxim Svistunov
Red Hat Customer Content Services

Robert Krátký
Red Hat Customer Content Services

Stephen Wadeley
Red Hat Customer Content Services

Florian Nadge
Red Hat Customer Content Services

Abstract

The Kernel Administration Guide documents tasks for maintaining the Red Hat Enterprise Linux 7 kernel. This release, includes information on using kpatch, managing kernel modules, and manually updating the kernel.

Table of Contents

PREFACE

The *Kernel Administration Guide* describes working with the kernel and shows several practical tasks. Beginning with the Kernel crash dump guide which steps through the process of setting up and testing **vmcore** collection in the event of a kernel failure, and including information on using kernel modules, manually upgrading a kernel, using kpatch, and how to interact with the **sysfs** facility.

The *Kernel Administration Guide* also covers selected use cases of managing the kernel and includes reference material about command line options, kernel tunables (also knowns as switches), and an inexhaustive discussion of kernel features.

CHAPTER 1. KERNEL CRASH DUMP GUIDE

1.1. INTRODUCTION TO KDUMP

1.1.1. About kdump and kexec

Kdump is a kernel crash dumping mechanism that allows you to save the contents of the system's memory for later analysis. It relies on **kexec**, which can be used to boot a Linux kernel from the context of another kernel, bypass BIOS, and preserve the contents of the first kernel's memory that would otherwise be lost.

In case of a system crash, kdump uses kexec to boot into a second kernel (a *capture kernel*). This second kernel resides in a reserved part of the system memory that is inaccessible to the first kernel. The second kernel then captures the contents of the crashed kernel's memory (a *crash dump*) and saves it.

IMPORTANT

A kernel crash dump can be the only information available in the event of a failure, the importance of having this data in a business critical environment cannot be underestimated. Red Hat advise that System Administrators regularly update and test **`kexec-tools`** in your normal kernel update cycle. This is especially important when new kernel features are implemented.

1.1.2. Memory requirements

In order for kdump to be able to capture a kernel crash dump and save it for further analysis, a part of the system memory has to be permanently reserved for the capture kernel. When reserved, this part of the system memory is not available to main kernel.

The memory requirements vary based on certain system parameters. One of the major factors is the system's hardware architecture. To find out the exact name of the machine architecture (such as **`x86_64`**) and print it to standard output, type the following command at a shell prompt:

```
uname -m
```

Another factor which influences the amount of memory to be reserved is the total amount of installed system memory. For example, on the x86_64 architecture, the amount of reserved memory is 160 MB + 2 bits for every 4 KB of RAM. On a system with 1 TB of total physical memory installed, this means 224 MB (160 MB + 64 MB). For a complete list of memory requirements for kdump based on the system architecture and the amount of physical memory, see Section 1.7.1, "Memory requirements for kdump".

On many systems, kdump can estimate the amount of required memory and reserve it automatically. This behavior is enabled by default, but only works on systems that have more than a certain amount of total available memory, which varies based on the system architecture. See Section 1.7.2, "Minimum threshold for automatic memory reservation" for a list of minimum requirements for automatic memory reservation based on the system architecture.

If the system has less than the minimum amount of memory required for the automatic allocation to work or if your use case requires a different value, you can configure the amount of reserved memory manually. For information on how to do so on the command line, see Section 1.2.2.1, "Configuring the memory usage". For information on how to configure the amount of reserved memory in the graphical user interface, see Section 1.2.3.1, "Configuring the memory usage".

IMPORTANT

It is highly recommended to test the configuration after setting up the kdump service, even when using the automatic memory reservation. For instructions on how to test your configuration, see Section 1.3, "Testing the kdump configuration".

1.2. INSTALLING AND CONFIGURING KDUMP

1.2.1. Installing kdump

In many cases, the **kdump** service is installed and activated by default on new Red Hat Enterprise Linux 7 installations. The **Anaconda** installer provides a screen for kdump configuration when performing an interactive installation using the graphical or text interface. The installer screen is titled **Kdump** and is available from the main **Installation Summary** screen, and only allows limited configuration - you can only select whether kdump is enabled and how much memory is reserved. Information about memory requirements for kdump is available in Section 1.7.1, "Memory requirements for kdump". The Kdump configuration screen in the installer is documented in the *Red Hat Enterprise Linux 7 Installation Guide*.

NOTE

In previous releases of Red Hat Enterprise Linux, kdump configuration was available in the **Firstboot** utility which was automatically executed **after** the installation finished and the system rebooted for the first time. Starting with Red Hat Enterprise Linux 7.1, kdump configuration has been moved into the installer.

Some installation options, such as custom Kickstart installations, do not have to install or enable kdump by default. If this is the case on your system, and you want to install kdump additionally, execute the following command as **root** at a shell prompt:

```
# yum install kexec-tools
```

The command above secures installation of kdump and all other necessary packages, assuming your system has an active subscription or a custom repository containing the **kexec-tools** package for your system's architecture.

NOTE

If you do not know whether kdump is installed on your system, you can check using **rpm**:

```
$ rpm -q kexec-tools
```

Additionally, a graphical configuration tool is available, but not installed by default if you use the command described above. To install this utility, which is described in Section 1.2.3, "Configuring kdump in the graphical user interface", use the following command as **root**:

```
# yum install system-config-kdump
```

For more information on how to install new packages in Red Hat Enterprise Linux 7 using the **Yum** package manager, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*.

IMPORTANT

Starting with Red Hat Enterprise Linux 7.4 the **`Intel IOMMU`** driver is supported with **kdump**. When running kernels from version 7.3 or earlier, it is advised that **`Intel IOMMU`** support is disabled.

1.2.2. Configuring kdump on the command line

1.2.2.1. Configuring the memory usage

Memory reserved for the kdump kernel is always reserved during system boot, which means that the amount of memory is specified in the system's boot loader configuration.

To specify the memory reserved for kdump kernel, set the **`crashkernel=`** option to the required value. For example, to reserve 128 MB of memory, use the following:

```
crashkernel=128M
```

For information about how to change the **`crashkernel=`** option on AMD64 and Intel 64 systems and IBM Power Systems servers using the **GRUB2** boot loader, and on IBM System z using **zipl**, see Section 7.1.1, "Setting kernel command-line parameters".

The **`crashkernel=`** option can be defined in multiple ways. The **`auto`** value enables automatic configuration of reserved memory based on the total amount of memory in the system, following the guidelines described in Section 1.7.1, "Memory requirements for kdump". Larger memory systems, up to the established limits of the operating system <link to OS Limits> are calculated according to architecture with the **`crashkernel=auto`** option.

Replace the **`auto`** value with a specific amount of memory to change this behavior.

The **`crashkernel=`** option can be particularly useful with smaller memory systems. For example, to reserve 128 MB of memory, use the following:

```
crashkernel=128M
```

You can also set the amount of reserved memory to be variable, depending on the total amount of installed memory. The syntax for variable memory reservation is **`crashkernel=<range1>:<size1>,<range2>:<size2>`**. For example:

```
crashkernel=512M-2G:64M,2G-:128M
```

The above example reserves 64 MB of memory if the total amount of system memory is 512 MB or higher and lower than 2 GB. If the total amount of memory is more than 2 GB, 128 MB is reserved for kdump instead.

Some systems require to reserve memory with a certain fixed offset. If the offset is set, the reserved memory begins there. To offset the reserved memory, use the following syntax:

```
crashkernel=128M@16M
```

The example above means that kdump reserves 128 MB of memory starting at 16 MB (physical address 0x01000000). If the offset parameter is set to 0 or omitted entirely, kdump offsets the reserved memory automatically. This syntax can also be used when setting a variable memory reservation as described above; in this case, the offset is always specified last (for example, **crashkernel=512M-2G:64M,2G-:128M@16M**).

1.2.2.2. Configuring the kdump type

When a kernel crash is captured, the core dump can be either stored as a file in a local file system, written directly to a device, or sent over a network using the **NFS** (Network File System) or **SSH** (Secure Shell) protocol. Only one of these options can be set at the moment, and the default option is to store the **vmcore** file in the **/var/crash/** directory of the local file system. To change this, as **root**, open the **/etc/kdump.conf** configuration file in a text editor and edit the options as described below.

To change the local directory in which the core dump is to be saved, remove the hash sign ("#") from the beginning of the **#path /var/crash** line, and replace the value with a desired directory path.

```
path /usr/local/cores
```

IMPORTANT

In Red Hat Enterprise Linux 7, the directory defined as the kdump target using the **path** directive must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of Red Hat Enterprise Linux, where the directory was being created automatically if it did not exist when starting the service.

Optionally, if you wish to write the file to a different partition, follow the same procedure with the one of the lines beginning with **#ext4**. Here, you can use either a device name (the **#ext4 /dev/vg/lv_kdump** line), a file system label (the **#ext4 LABEL=/boot** line) or a UUID (the **#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** line). Change the file system type as well as the device name, label or UUID to the desired values. For example:

```
ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
```

IMPORTANT

Specifying storage devices using a **LABEL=** or **UUID=** is recommended. Disk device names such as **/dev/sda3** are not guaranteed to be consistent across reboot. See the *Red Hat Enterprise Linux 7 Storage Administration Guide* for information about persistent disk device naming.

IMPORTANT

When dumping to DASD on s390x hardware, it is essential that the dump devices are correctly specified in **/etc/dasd.conf** before proceeding.

To write the dump directly to a device, remove the hash sign ("#") from the beginning of the **#raw /dev/vg/lv_kdump** line, and replace the value with a desired device name. For example:

```
raw /dev/sdb1
```

To store the dump to a remote machine using the **NFS** protocol, remove the hash sign ("#") from the beginning of the #**nfs my.server.com:/export/tmp** line, and replace the value with a valid hostname and directory path. For example:

```
nfs penguin.example.com:/export/cores
```

To store the dump to a remote machine using the **SSH** protocol, remove the hash sign ("#") from the beginning of the #**ssh user@my.server.com** line, and replace the value with a valid username and hostname. To include your SSH key in the configuration as well, remove the hash sign from the beginning of the #**sshkey /root/.ssh/kdump_id_rsa** line and change the value to the location of a key valid on the server you are trying to dump to. For example:

```
ssh john@penguin.example.com
sshkey /root/.ssh/mykey
```

For information on how to configure an SSH server and set up a key-based authentication, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*.

For a complete list of currently supported and unsupported targets sorted by type, see Table 1.3, "Supported kdump Targets".

1.2.2.3. Configuring the core collector

To reduce the size of the **vmcore** dump file, **kdump** allows you to specify an external application (a *core collector*) to compress the data, and optionally leave out all irrelevant information. Currently, the only fully supported core collector is **makedumpfile**.

To enable the core collector, as **root**, open the **/etc/kdump.conf** configuration file in a text editor, remove the hash sign ("#") from the beginning of the #**core_collector makedumpfile -l --message-level 1 -d 31** line, and edit the command line options as described below.

To enable the dump file compression, add the **-c** parameter. For example:

```
core_collector makedumpfile -c
```

To remove certain pages from the dump, add the **-d *value*** parameter, where *value* is a sum of values of pages you want to omit as described in Table 1.4, "Supported Filtering Levels". For example, to remove both zero and free pages, use the following:

```
core_collector makedumpfile -d 17 -c
```

See the **makedumpfile(8)** man page for a complete list of available options.

1.2.2.4. Configuring the default action

By default, when **kdump** fails to create a core dump at the target location specified in Section 1.2.2.2, "Configuring the kdump type", the root file system is mounted and **kdump** attempts to save the core locally. To change this behavior, as **root**, open the **/etc/kdump.conf** configuration file in a text editor, remove the hash sign ("#") from the beginning of the #**default shell** line, and replace the value with a desired action as described in Table 1.5, "Supported Default Actions".

For example:

```
default reboot
```

1.2.2.5. Enabling the service

To start the **kdump** daemon at boot time, type the following at a shell prompt as **root**:

```
systemctl enable kdump.service
```

This enables the service for **multi-user.target**. Similarly, typing **systemctl stop kdump** disables it. To start the service in the current session, use the following command as **root**:

```
systemctl start kdump.service
```

> **IMPORTANT**
>
> In Red Hat Enterprise Linux 7, the directory defined as the kdump target must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of Red Hat Enterprise Linux, where the directory was being created automatically if it did not exist when starting the service.

For more information on systemd and configuring services in general, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*.

1.2.3. Configuring kdump in the graphical user interface

To start the **Kernel Dump Configuration** utility, select **Activities** → **Other** → **Kernel crash dumps** from the panel, or type **system-config-kdump** at a shell prompt. As a result a window appears as shown in Figure 1.1, "Basic Settings".

The utility allows you to configure **kdump** as well as to enable or disable starting the service at boot time. When you are done, click **Apply** to save the changes. Unless you are already authenticated, enter the superuser password. The utility presents you with a reminder that you must reboot the system in order to apply any changes you have made to the configuration.

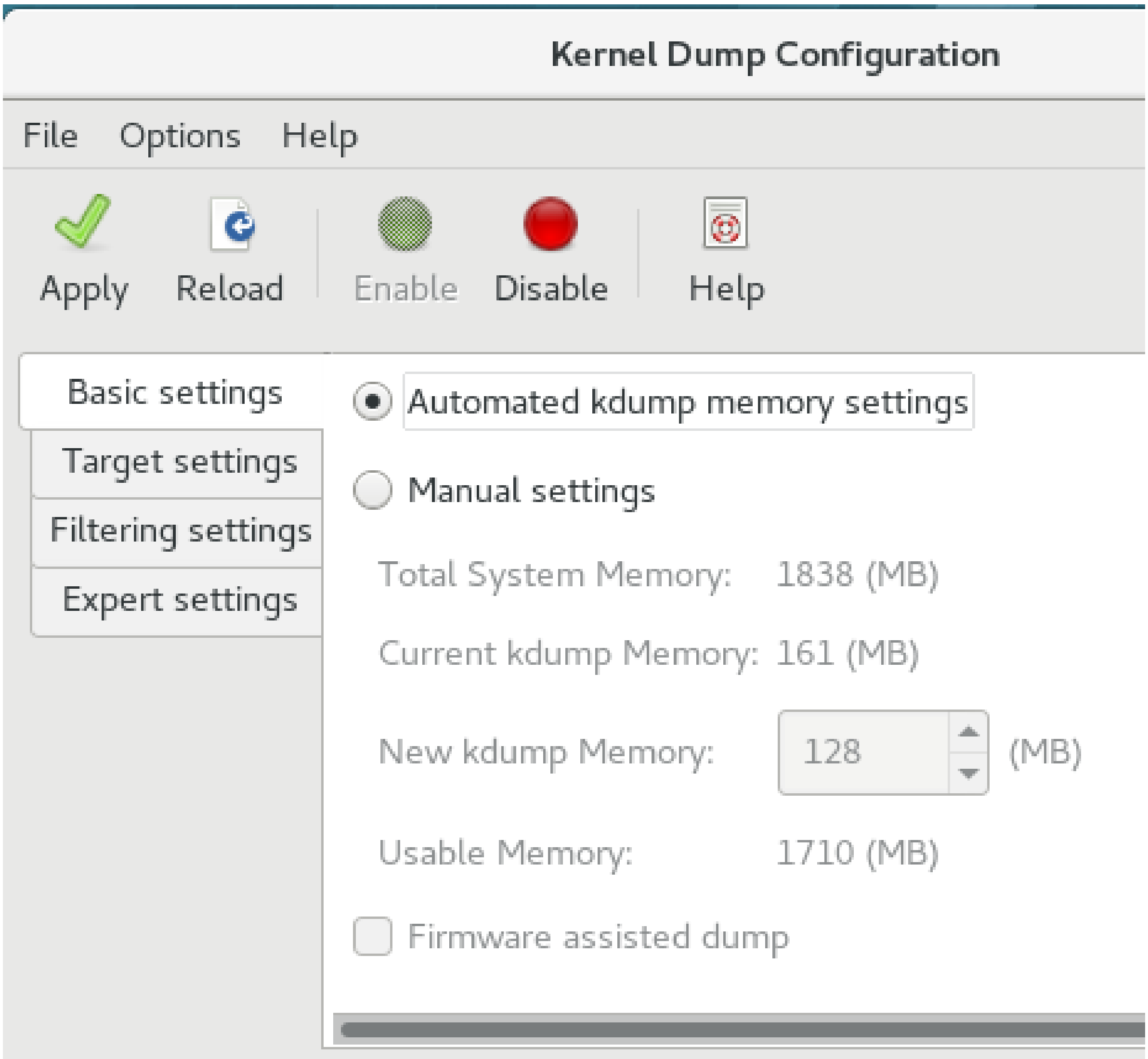
> **IMPORTANT**
>
> On IBM System z or PowerPC systems with **SELinux** running in Enforcing mode, the kdumpgui_run_bootloader Boolean must be enabled before launching the Kernel Dump Configuration utility. This Boolean allows system-config-kdump to run the boot loader in the bootloader_t SELinux domain. To permanently enable the Boolean, run the following command as root;
>
> ```
> # setsebool -P kdumpgui_run_bootloader 1
> ```

> **IMPORTANT**
>
> When dumping to DASD on s390x hardware, it is essential that the dump devices are correctly specified in **/etc/dasd.conf** before proceeding.

1.2.3.1. Configuring the memory usage

The **Basic Settings** tab enables you to configure the amount of memory that is reserved for the **kdump** kernel. To do so, select the **Manual settings** radio button, and click the up and down arrow buttons next to the **New kdump Memory** field to increase or decrease the amount of memory to be reserved. Notice that the **Usable Memory** field changes accordingly showing you the remaining memory that is available to the system. See Section 1.1.2, "Memory requirements" for more information on kdump's memory requirements.

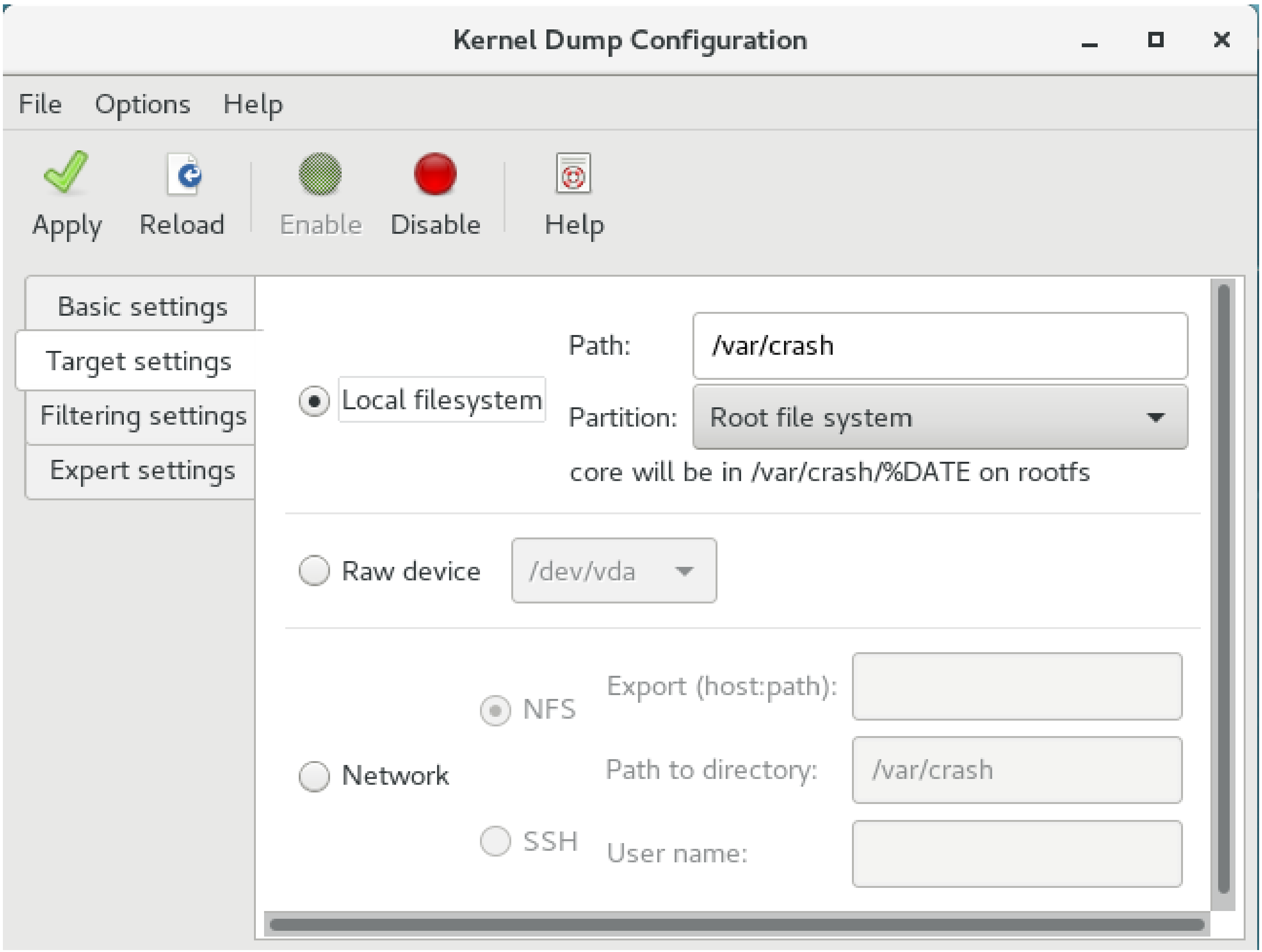
Figure 1.1. Basic Settings



1.2.3.2. Configuring the kdump type

The **Target Settings** tab allows you to specify the target location for the **vmcore** dump. The dump can be either stored as a file in a local file system, written directly to a device, or sent over a network using the **NFS** (Network File System) or **SSH** (Secure Shell) protocol.

Figure 1.2. Target Settings



To save the dump to the local file system, select the `Local filesystem` radio button. Optionally, you can customize the settings by choosing a different partition from the `Partition` drop-down list and a target directory using the `Path` field.

IMPORTANT

In Red Hat Enterprise Linux 7, the directory defined as the kdump target must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of Red Hat Enterprise Linux, where the directory was being created automatically if it did not exist when starting the service.

To write the dump directly to a device, select the `Raw device` radio button, and choose the desired target device from the drop-down list next to it.

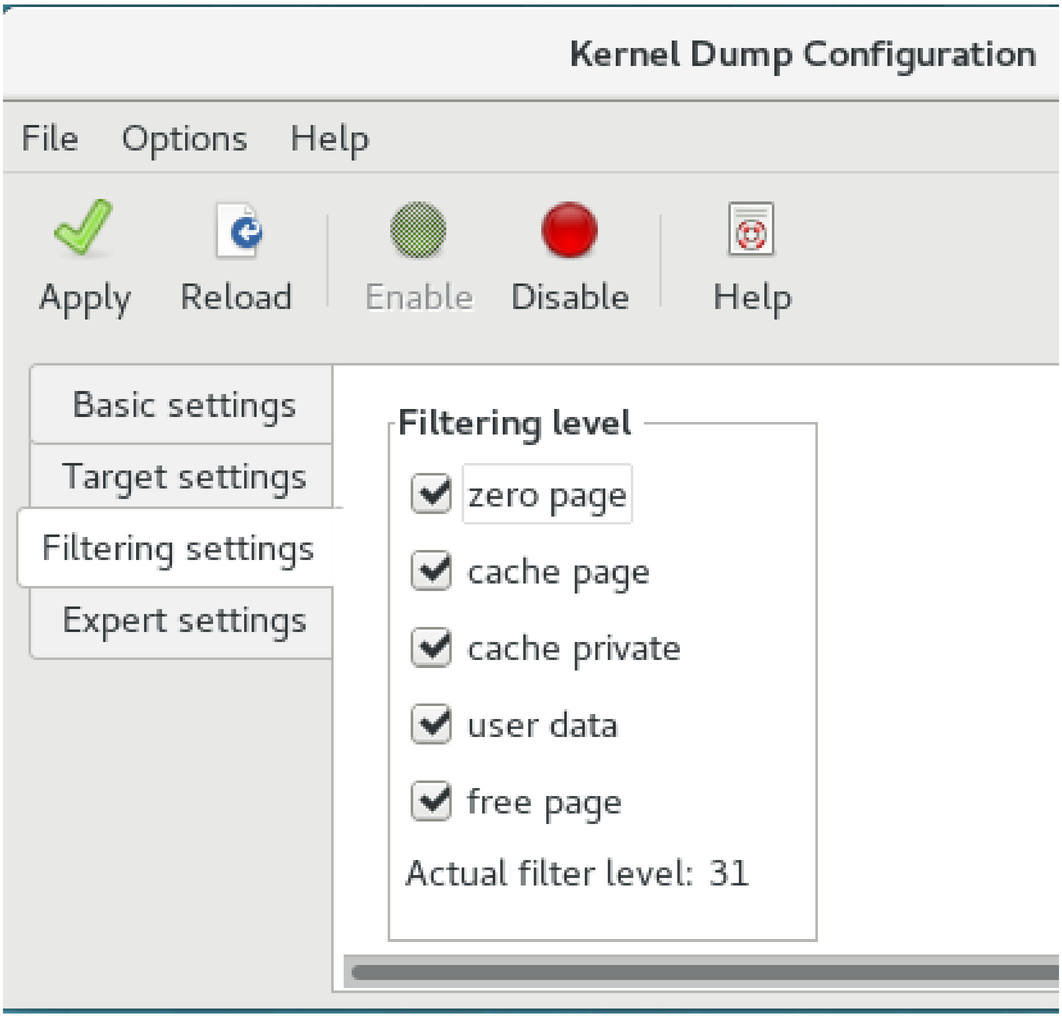
To send the dump to a remote machine over a network connection, select the `Network` radio button. To use the **NFS** protocol, select the `NFS` radio button, and fill the `Server name` and `Path to directory` fields. To use the **SSH** protocol, select the `SSH` radio button, and fill the `Server name`, `Path to directory`, and `User name` fields with the remote server address, target directory, and a valid user name respectively.

For information on how to configure an SSH server and set up a key-based authentication, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*. For a complete list of currently supported targets, see Table 1.3, "Supported kdump Targets".

1.2.3.3. Configuring the core collector

The **Filtering Settings** tab enables you to select the filtering level for the **vmcore** dump.

Figure 1.3. Filtering Settings

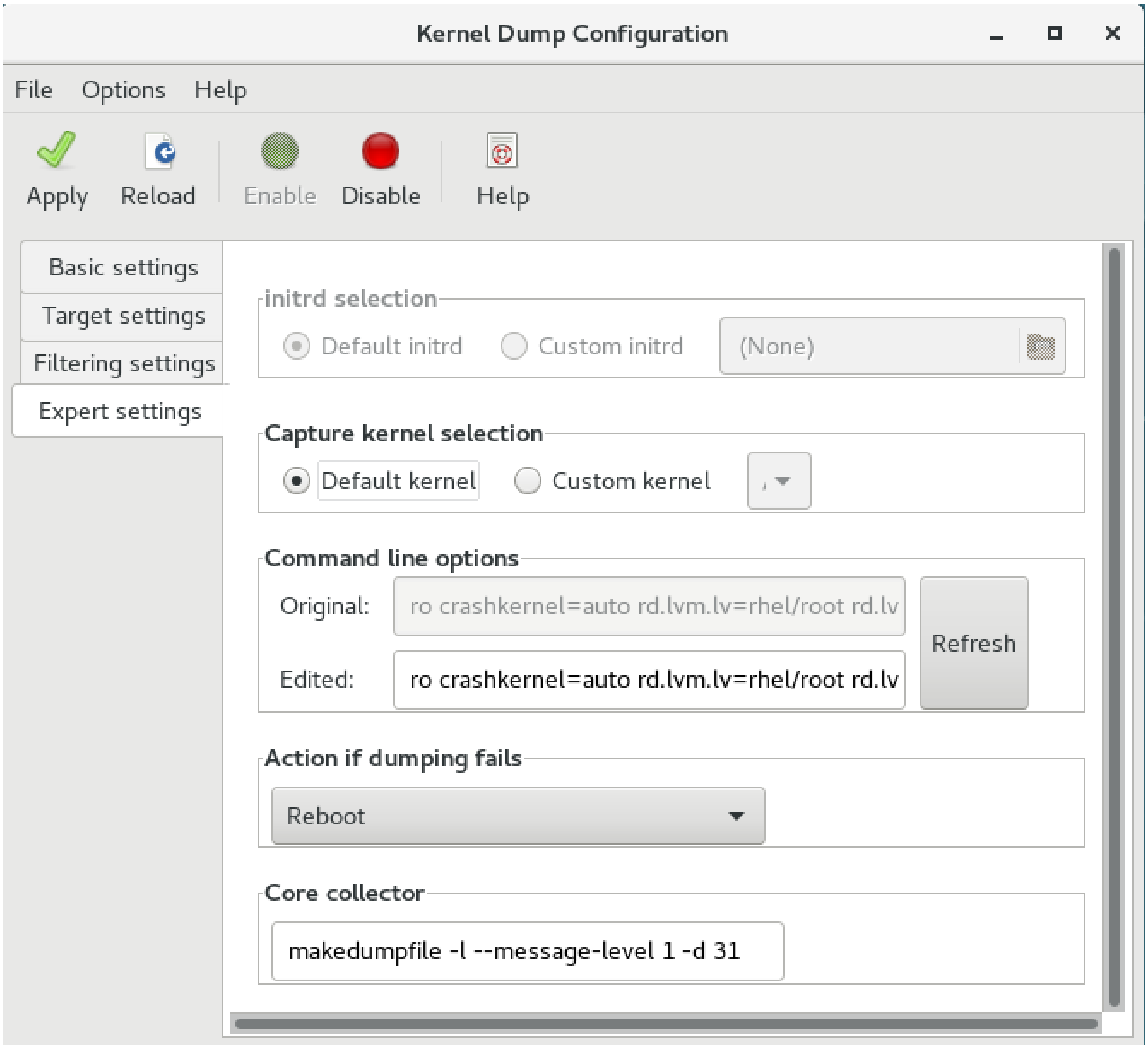


To exclude the **zero page**, **cache page**, **cache private**, **user data**, or **free page** from the dump, select the checkbox next to the appropriate label.

1.2.3.4. Configuring the default action

To choose what action to perform when **kdump** fails to create a core dump, select an appropriate option from the **Default action** drop-down list. Available options are **dump to rootfs and reboot** (the default action which attempts to save the core locally and then reboots the system), **reboot** (to reboot the system), **shell** (to present a user with an interactive shell prompt), **halt** (to halt the system), and **poweroff** (to power the system off).

Figure 1.4. Filtering Settings



To customize the options that are passed to the **`makedumpfile`** core collector, edit the **`Core collector`** text field; see Section 1.2.2.3, "Configuring the core collector" for more information.

1.2.3.5. Enabling the service

To start the **kdump** service at boot time, click the **`Enable`** button on the toolbar and then click the **`Apply`** button. This enables and activates the service for **`multi-user.target`**. Click the **`Disable`** button and confirm by clicking the **`Apply`** button to disable the service immediately.

IMPORTANT

In Red Hat Enterprise Linux 7, the directory defined as the kdump target must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of Red Hat Enterprise Linux, where the directory was being created automatically if it did not exist when starting the service.

For more information on systemd targets and configuring services in general, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*.

1.3. TESTING THE KDUMP CONFIGURATION

WARNING

The commands below cause the kernel to crash. Use caution when following these steps, and by no means use them on a production system.

To test the configuration, reboot the system with **kdump** enabled, and make sure that the service is running:

```
~]# systemctl is-active kdump
active
```

Then type the following commands at a shell prompt:

```
echo 1 > /proc/sys/kernel/sysrq
echo c > /proc/sysrq-trigger
```

This forces the Linux kernel to crash, and the ***address*-*YYYY*-*MM*-*DD*-*HH*:*MM*:*SS*/vmcore** file is copied to the location you have selected in the configuration (that is, to **/var/crash/** by default).

NOTE

In addition to confirming the validity of the configuration, this action can also be used to record how long it takes to a crash dump to complete if it is performed under a representative test load.

1.3.1. Additional resources

1.3.1.1. Installed documentation

- **kdump.conf**(5) — a manual page for the **/etc/kdump.conf** configuration file containing the full documentation of available options.
- **zipl.conf**(5) — a manual page for the **/etc/zipl.conf** configuration file.
- **zipl**(8) — a manual page for the **zipl** boot loader utility for IBM System z.
- **makedumpfile**(8) — a manual page for the **makedumpfile** core collector.
- **kexec**(8) — a manual page for **kexec**.
- **crash**(8) — a manual page for the **crash** utility.
- **/usr/share/doc/kexec-tools-*version*/kexec-kdump-howto.txt** — an overview of the **kdump** and **kexec** installation and usage.

1.3.1.2. Online documentation

https://access.redhat.com/site/solutions/6038

The Red Hat Knowledgebase article about the **kexec** and **kdump** configuration.

https://access.redhat.com/site/solutions/223773

The Red Hat Knowledgebase article about supported **kdump** targets.

http://people.redhat.com/anderson/

The **crash** utility homepage.

https://www.gnu.org/software/grub/

The **GRUB2** boot loader homepage and documentation.

1.4. FIRMWARE ASSISTED DUMP MECHANISMS

1.4.1. The case for firmware assisted dump

The **kexec** and **kdump** mechanisms are a reliable and proven method of capturing a core dump on AMD64 and Intel 64 systems. However, some hardware with a longer history, particularly mini and mainframe systems, allows us to leverage the onboard firmware to isolate regions of memory and prevent any accidental overwriting of data that is important to the crash analysis.

This chapter covers some of the available firmware assisted dump methods and how they integrate with Red Hat Enterprise Linux.

1.4.2. Using fadump on IBM PowerPC hardware

Firmware-assisted dump (**fadump**) is a reliable alternative to **kexec-kdump** available on IBM PowerPC LPARS. It captures vmcore from a fully-reset system with PCI and I/O devices reinitialized. While this mechanism uses the firmware to preserve the memory in case of a crash, it reuses the **kdump** userspace scripts to save the vmcore"

To achieve this, **fadump** registers the regions of memory that must be preserved in the event of a crash with the system firmware. These regions consist of all the system memory contents, except the boot memory, system registers and hardware Page Table Entries (PTEs).

For further details about the **fadump** mechanism, including PowerPC-specific methods of resetting hardware, review **/usr/share/doc/kexec-tools-*X.y.z*/fadump-howto.txt** where "X.y.z" correspond to the version number of **kexec-tools** installed on your system.

NOTE

The area of memory not preserved and known as **boot memory** is the amount of RAM required to successfully boot the kernel after a crash event. By default, the boot memory size is 256MB or 5% of total system RAM, whichever is larger.

Unlike a **kexec**-initiated event, the **fadump** process uses the production kernel to recover a crash dump. When booting after a crash, PowerPC hardware makes the device node **/proc/device-tree/rtas/ibm,kernel-dump** available to **procfs**, which the fadump-aware **kdump** scripts check for to save the vmcore. After this has completed, the system is rebooted cleanly.

Enabling fadump

1. Install and configure **kdump** as described in Section 1.2, “Installing and configuring kdump” .

2. Add **fadump=on** to the **GRUB_CMDLINE_LINUX** line in **/etc/default/grub**:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet fadump=on"
```

3. (optional) If you want to specify reserved boot memory instead of accepting the defaults, add **fadump_reserve_mem=*xx*M** to **GRUB_CMDLINE_LINUX** in **/etc/default/grub**, where *xx* is the amount of the memory required in megabytes:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet fadump=on fadump_reserve_mem=xxM"
```

IMPORTANT

As with all boot configuration options, it is strongly recommended that you test the configuration before it is needed. If you observe Out of Memory (OOM) errors when booting from the crash kernel, increase the value specified in **fadump_reserve_mem=** until the crash kernel can boot cleanly. Some trial and error can be required in this case.

1.4.3. Firmware assisted dump methods on IBM z Systems

There are two firmware assisted dump mechanisms on IBM z Systems. They are **Stand-alone Dump** and **VMDUMP**.

The **kdump** infrastructure is supported and utilized on these systems and configuration from Red Hat Enterprise Linux is described in Section 1.2, "Installing and configuring kdump" . However, there are potentially some advantages to using either of the firmware assisted methods IBM z System hardware provides.

The Stand-alone Dump (SADMP) mechanism is initiated and controlled from the system console, and must be stored on an IPL bootable device.

Similar to SADMP is VMDUMP. This tool is also initiated from the system console, but has a mechanism to retrieve the resulting dump from hardware and copy it to a system for analysis.

One advantage of these methods (and similarly to other hardware based dump mechanisms), is the ability to capture the state of a machine in the Early Boot phase (before the kdump service is started)

Although VMDUMP contains a mechanism to receive the dump file into a Red Hat Enterprise Linux system, the configuration and control of both SADMP and VMDUMP are managed from the IBM z System Hardware console.

IBM discuss SADMP in detail, at
http://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.ieav100/standa.htm
and VMDUMP at
http://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lgdt/lgdt_t_vmdump.htm

IBM also have a documentation set for using the dump tools on Red Hat Enterprise Linux 7 at
http://www.ibm.com/support/knowledgecenter/linuxonibm/com.ibm.linux.z.lgdt/lgdt_t_usingdumptools

1.4.4. Using sadump on Fujitsu PRIMEQUEST systems

The Fujitsu **sadump** mechanism is designed to provide a fallback dump capture in the event **kdump** is unable to complete successfully.

The **sadump** process is invoked manually from the system ManageMent Board (MMB) interface.

With this system, configure kdump as normal for an X86_64 server and then perform the following additional steps to enable **sadump**.

Add or edit the following lines in `/etc/sysctl.conf` to ensure that **kdump** starts as expected for **sadump**.

```
kernel.panic=0
kernel.unknown_nmi_panic=1
```

In addition to the above, you must also add some options to `/etc/kdump.conf` to ensure that **kdump** behaves correctly for sadump.

In particular, ensure that after **kdump**, the system does not reboot. If the system reboots after **kdump** has failed to save core, then you have no opportunity to invoke **sadump**.

Set the **`default`** action in `/etc/kdump.conf` to be either *halt* or *shell* to achieve this.

```
default shell
blacklist kvm-intel
```

IMPORTANT

For details on configuring your hardware for **sadump**, see the FUJITSU Server PRIMEQUEST 2000 Series Installation Manual.

1.5. ANALYZING A CORE DUMP

To determine the cause of the system crash, you can use the **crash** utility, which provides an interactive prompt very similar to the GNU Debugger (GDB). This utility allows you to interactively analyze a running Linux system as well as a core dump created by **`netdump`**, **`diskdump`**, **`xendump`**, or **`kdump`**.

1.5.1. Installing the crash utility

To install the **crash** analyzing tool, execute the following command from a shell prompt as **`root`**:

```
yum install crash
```

In addition to **crash**, it is also necessary to install the **kernel-debuginfo** package that corresponds to your running kernel, which provides the data necessary for dump analysis. To install **kernel-debuginfo** we use the **`debuginfo-install`** command as **`root`**:

```
debuginfo-install kernel
```

For more information on how to install new packages in Red Hat Enterprise Linux using the **Yum** package manager, see the *Red Hat Enterprise Linux 7 System Administrator's Guide*.

1.5.2. Running the crash utility

To start the utility, type the command in the following form at a shell prompt:

```
crash /usr/lib/debug/lib/modules/<kernel>/vmlinux \
/var/crash/<timestamp>/vmcore
```

Use the same *<kernel>* version that was captured by **kdump**. To find out which kernel you are currently running, use the **uname -r** command.

Example 1.1. Running the crash utility

```
~]# crash /usr/lib/debug/lib/modules/2.6.32-69.el6.i686/vmlinux \
/var/crash/127.0.0.1-2010-08-25-08:45:02/vmcore

crash 5.0.0-23.el6
Copyright (C) 2002-2010  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public
License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for
details.

GNU gdb (GDB) 7.0
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...

      KERNEL: /usr/lib/debug/lib/modules/2.6.32-69.el6.i686/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2010-08-25-08:45:02/vmcore  [PARTIAL
DUMP]
        CPUS: 4
        DATE: Wed Aug 25 08:44:47 2010
      UPTIME: 00:09:02
LOAD AVERAGE: 0.00, 0.01, 0.00
       TASKS: 140
    NODENAME: hp-dl320g5-02.lab.bos.redhat.com
     RELEASE: 2.6.32-69.el6.i686
     VERSION: #1 SMP Tue Aug 24 10:31:45 EDT 2010
     MACHINE: i686  (2394 Mhz)
      MEMORY: 8 GB
       PANIC: "Oops: 0002 [#1] SMP " (check log for details)
         PID: 5591
     COMMAND: "bash"
        TASK: f196d560  [THREAD_INFO: ef4da000]
         CPU: 2
```

```
        STATE: TASK_RUNNING (PANIC)

crash>
```

1.5.3. Displaying the message buffer

To display the kernel message buffer, type the **log** command at the interactive prompt.

Example 1.2. Displaying the kernel message buffer

```
crash> log
... several lines omitted ...
EIP: 0060:[<c068124f>] EFLAGS: 00010096 CPU: 2
EIP is at sysrq_handle_crash+0xf/0x20
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000
ESI: c0a09ca0 EDI: 00000286 EBP: 00000000 ESP: ef4dbf24
 DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process bash (pid: 5591, ti=ef4da000 task=f196d560 task.ti=ef4da000)
Stack:
 c068146b c0960891 c0968653 00000003 00000000 00000002 efade5c0 c06814d0
<0> fffffffb c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002
b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560
ef4dbfb4
Call Trace:
 [<c068146b>] ? __handle_sysrq+0xfb/0x160
 [<c06814d0>] ? write_sysrq_trigger+0x0/0x50
 [<c068150f>] ? write_sysrq_trigger+0x3f/0x50
 [<c0569ec4>] ? proc_reg_write+0x64/0xa0
 [<c0569e60>] ? proc_reg_write+0x0/0xa0
 [<c051de50>] ? vfs_write+0xa0/0x190
 [<c051e8d1>] ? sys_write+0x41/0x70
 [<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04 09 d0
88 41 03 f3 c3 90 c7 05 c8 1b 9e c0 01 00 00 00 0f ae f8 89 f6 <c6> 05
00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00 00 8d 50 d0 83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000
```

Type **help log** for more information on the command usage.

NOTE

The kernel message buffer includes the most essential information about the system crash and, as such, it is always dumped first in to the **vmcore-dmesg.txt** file. This is useful when an attempt to get the full **vmcore** file failed, for example because of lack of space on the target location. By default, **vmcore-dmesg.txt** is located in the **/var/crash/** directory.

1.5.4. Displaying a backtrace

To display the kernel stack trace, type the **bt** command at the interactive prompt. You can use **bt** **<*pid*>** to display the backtrace of a single process.

Example 1.3. Displaying the kernel stack trace

```
crash> bt
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
 #0 [ef4dbdcc] crash_kexec at c0494922
 #1 [ef4dbe20] oops_end at c080e402
 #2 [ef4dbe34] no_context at c043089d
 #3 [ef4dbe58] bad_area at c0430b26
 #4 [ef4dbe6c] do_page_fault at c080fb9b
 #5 [ef4dbee4] error_code (via page_fault) at c080d809
    EAX: 00000063  EBX: 00000063  ECX: c09e1c8c  EDX: 00000000  EBP:
00000000
    DS:  007b      ESI: c0a09ca0  ES:  007b      EDI: 00000286  GS:
00e0
    CS:  0060      EIP: c068124f  ERR: ffffffff  EFLAGS: 00010096
 #6 [ef4dbf18] sysrq_handle_crash at c068124f
 #7 [ef4dbf24] __handle_sysrq at c0681469
 #8 [ef4dbf48] write_sysrq_trigger at c068150a
 #9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
    EAX: ffffffda  EBX: 00000001  ECX: b7776000  EDX: 00000002
    DS:  007b      ESI: 00000002  ES:  007b      EDI: b7776000
    SS:  007b      ESP: bfcb2088  EBP: bfcb20b4  GS:  0033
    CS:  0073      EIP: 00edc416  ERR: 00000004  EFLAGS: 00000246
```

Type **help bt** for more information on the command usage.

1.5.5. Displaying a process status

To display status of processes in the system, type the **ps** command at the interactive prompt. You can use **ps** **<*pid*>** to display the status of a single process.

Example 1.4. Displaying the status of processes in the system

```
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM     VSZ    RSS  COMM
>     0      0   0  c09dc560  RU   0.0       0      0  [swapper]
>     0      0   1  f7072030  RU   0.0       0      0  [swapper]
      0      0   2  f70a3a90  RU   0.0       0      0  [swapper]
>     0      0   3  f70ac560  RU   0.0       0      0  [swapper]
      1      0   1  f705ba90  IN   0.0    2828   1424  init
... several lines omitted ...
   5566      1   1  f2592560  IN   0.0   12876    784  auditd
   5567      1   2  ef427560  IN   0.0   12876    784  auditd
   5587   5132   0  f196d030  IN   0.0   11064   3184  sshd
>  5591   5587   2  f196d560  RU   0.0    5084   1648  bash
```

Type **help ps** for more information on the command usage.

1.5.6. Displaying virtual memory information

To display basic virtual memory information, type the **vm** command at the interactive prompt. You can use **vm *<pid>*** to display information on a single process.

Example 1.5. Displaying virtual memory information of the current context

```
crash> vm
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
   MM       PGD      RSS    TOTAL_VM
f19b5900  ef9c6000  1648k    5084k
  VMA       START      END    FLAGS  FILE
f1bb0310    242000    260000 8000875  /lib/ld-2.12.so
f26af0b8    260000    261000 8100871  /lib/ld-2.12.so
efbc275c    261000    262000 8100873  /lib/ld-2.12.so
efbc2a18    268000    3ed000 8000075  /lib/libc-2.12.so
efbc23d8    3ed000    3ee000 8000070  /lib/libc-2.12.so
efbc2888    3ee000    3f0000 8100071  /lib/libc-2.12.so
efbc2cd4    3f0000    3f1000 8100073  /lib/libc-2.12.so
efbc243c    3f1000    3f4000 100073
efbc28ec    3f6000    3f9000 8000075  /lib/libdl-2.12.so
efbc2568    3f9000    3fa000 8100071  /lib/libdl-2.12.so
efbc2f2c    3fa000    3fb000 8100073  /lib/libdl-2.12.so
f26af888    7e6000    7fc000 8000075  /lib/libtinfo.so.5.7
f26aff2c    7fc000    7ff000 8100073  /lib/libtinfo.so.5.7
efbc211c    d83000    d8f000 8000075  /lib/libnss_files-2.12.so
efbc2504    d8f000    d90000 8100071  /lib/libnss_files-2.12.so
efbc2950    d90000    d91000 8100073  /lib/libnss_files-2.12.so
f26afe00    edc000    edd000 4040075
f1bb0a18   8047000   8118000 8001875  /bin/bash
f1bb01e4   8118000   811d000 8101873  /bin/bash
f1bb0c70   811d000   8122000 100073
f26afae0   9fd9000   9ffa000 100073
... several lines omitted ...
```

Type **help vm** for more information on the command usage.

1.5.7. Displaying open files

To display information about open files, type the **files** command at the interactive prompt. You can use **files *<pid>*** to display files opened by only one selected process.

Example 1.6. Displaying information about open files of the current context

```
crash> files
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
ROOT: /    CWD: /root
 FD    FILE     DENTRY    INODE    TYPE  PATH
  0  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
  1  efade5c0  eee14090  f00431d4  REG   /proc/sysrq-trigger
```

```
  2  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
 10  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
255  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
```

Type **help files** for more information on the command usage.

1.5.8. Exiting the utility

To exit the interactive prompt and terminate **crash**, type **exit** or **q**.

Example 1.7. Exiting the crash utility

```
crash> exit
~]#
```

1.6. FREQUENTLY ASKED QUESTIONS

What considerations need to be made for using Kdump in a clustered environment?

How do I configure kdump for use with the RHEL 6, 7 High Availability Add-On? shows the options available to system administrators using the High Availability Add-On.

Kdump fails during early boot, How do I capture the boot log?

If there is a problem booting the second kernel, it is necessary to review the early boot logs, these can be obtained by enabling a serial console to the affected machine.

How do I setup serial console in RHEL7? shows the configuration needed to enable access to the early boot messages.

How do I increase the messaging from makedumpfile for debugging?

In the event that **makedumpfile** fails, then it is necessary to increase the log level to understand what is going wrong. This is different from setting the dump level and is achieved by editing **/etc/kdump.conf** and increasing the **message_level** option to **makedumpfile** on the **core_collector** line entry.

By default **makedumpfile** is set to level 7, which includes the progress indicator, common message, and error message output. Set this level to 31 to get further debugging information.

Ensure that your core_collector config line looks similar to this when set:

```
core_collector makedumpfile -l --message-level 1 -d 31
```

How do I debug Dracut?

Sometimes **dracut** can fail to build an initramfs. If this happens, increase the log level in **dracut** to isolate the issue.

Edit **/etc/kdump.conf** and change the **dracut_args** line to include the option **-L 5** in addition to any other dracut arguments you require.

If you have no other options configured in **dracut_args**, the result looks similar to this:

```
dracut_args -L 5
```

What methods of dumping are available for virtual machines?

In most cases, the **kdump** mechanism is sufficient for obtaining a memory dump from a machine after a crash or panic. This can be set up in the same manner as installations to bare metal.

However, in some cases, it is necessary to work directly with the hypervisor to obtain a crash dump. There are two mechanisms available with **libvirt** to achieve this; **pvpanic** and **virsh dump**. Both of these methods are described in the Virtualization Deployment and Administration Guide.

The **pvpanic** mechanism can be found at Virtualization Deployment and Administration Guide - Setting a Panic Device.

The **virsh dump** command is discussed in Virtualization Deployment and Administration Guide - Creating a Dump File of a Domain's Core.

How do I upload a large dump file to Red Hat Support Services?

In some cases, it might be necessary to send a kernel crash dump file to Red Hat Global Support Services for analysis. However, the dump file can be very large, even after being filtered. Since files larger than 250 MB cannot be uploaded directly through the Red Hat Customer Portal when opening a new support case, an FTP server is provided by Red Hat for uploading large files.

The FTP server's address is **dropbox.redhat.com** and the files are to be uploaded in the **/incoming/** directory. Your FTP client needs to be set into passive mode; if your firewall does not allow this mode, use the **origin-dropbox.redhat.com** server using active mode.

Make sure that the uploaded files are compressed using a program such as **gzip** and properly and descriptively named. Using your support case number in the file name is recommended. After successfully uploading all necessary files, provide the engineer in charge of your support case with the exact file name and its SHA1 or MD5 checksum.

For more specific instructions and additional information, see How to provide files to Red Hat Support .

How much time is needed for a crash dump to complete?

It is often necessary, for the purposes of disaster recovery planning, to know how long a dump takes to complete. However, the length of time it takes is highly dependent on the amount of memory being copied to disk and the speed of the interfaces between RAM and storage.

For any test of timings, the system must be operating under a representative load, otherwise the page exclusion choices can present a false view of kdump behavior with a fully loaded production system. This discrepancy is present especially when working with very large quantities of RAM.

Also consider storage interfaces in your planning when assessing time to dump. Because of network constraints, a connection dumping over **ssh** for example, can take longer to complete than a locally attached SATA disk.

How is Kdump configured during installation?

You can configure **kdump** during installation with a limited set of options in kickstart or the interactive GUI.

The **kdump** configuration using the **anaconda** installation GUI is documented in the KDUMP section of the Installation Guide.

The **kickstart** syntax is:

```
%addon com_redhat_kdump [--disable,enable] [--reserve-mb=[auto,value]]
%end
```

With this add-on to Kickstart, you can disable or enable kdump functionality, optionally defining the reserved memory size, either by explicitly invoking the default option of auto (which is also the case if the entire switch is omitted) or specifying a numeric value in megabytes.

To learn how Kickstart can be used to automate system deployments, read Kickstart Installations in the Installation Guide.

For further details about Kickstart add-on syntax, review the Kickstart Syntax Reference in the Installation Guide.

1.7. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS

1.7.1. Memory requirements for kdump

In order for kdump to be able to capture a kernel crash dump and save it for further analysis, a part of the system memory has to be permanently reserved for the capture kernel. The table below contains a list of minimum memory requirements for kdump based on the system's architecture and total available physical memory.

For information on how to change memory settings on the command line, see Section 1.2.2.1, “Configuring the memory usage”. For instructions on how to set up the amount of reserved memory in the graphical user interface, see Section 1.2.3.1, “Configuring the memory usage”.

Table 1.1. Minimum Amount of Reserved Memory Required for kdump

Architecture	Available Memory	Minimum Reserved Memory
AMD64 and Intel 64 (**x86_64**)	2 GB and more	160 MB + 2 bits for every 4 KB of RAM. For a system with 1 TB of memory, 224 MB is the minimum (160 + 64 MB).
IBM POWER (**ppc64**)	2 GB to 4 GB	256 MB of RAM.
	4 GB to 32 GB	512 MB of RAM.
	32 GB to 64 GB	1 GB of RAM.
	64 GB to 128 GB	2 GB of RAM.
	128 GB and more	4 GB of RAM.

Architecture	Available Memory	Minimum Reserved Memory
IBM System z (`s390x`)	2 GB and more	160 MB + 2 bits for every 4 KB of RAM. For a system with 1 TB of memory, 224 MB is the minimum (160 + 64 MB).

For more information about various Red Hat Enterprise Linux technology capabilities and limits, see https://access.redhat.com/articles/rhel-limits.

1.7.2. Minimum threshold for automatic memory reservation

On some systems, it is possible to allocate memory for kdump automatically, either by using the **`crashkernel=auto`** parameter in the bootloader's configuration file, or by enabling this option in the graphical configuration utility. For this automatic reservation to work, however, a certain amount of total memory needs to be available in the system. This amount differs based on the system's architecture.

The table below lists the thresholds for automatic memory allocation. If the system has less memory than specified in the table, memory needs to be reserved manually.

For information on how to change these settings on the command line, see Section 1.2.2.1, "Configuring the memory usage". For instructions on how to change the amount of reserved memory in the graphical user interface, see Section 1.2.3.1, "Configuring the memory usage".

Table 1.2. Minimum Amount of Memory Required for Automatic Memory Reservation

Architecture	Required Memory
AMD64 and Intel 64 (`x86_64`)	2 GB
IBM POWER (`ppc64`)	2 GB
IBM System z (`s390x`)	4 GB

1.7.3. Supported kdump targets

When a kernel crash is captured, the core dump can be either written directly to a device, stored as a file on a local file system, or sent over a network. The table below contains a complete list of dump targets that are currently supported or explicitly unsupported by kdump.

For information on how to configure the target type on the command line, see Section 1.2.2.2, "Configuring the kdump type". For information on how to do so in the graphical user interface, see Section 1.2.3.2, "Configuring the kdump type" .

Table 1.3. Supported kdump Targets

Type	Supported Targets	Unsupported Targets
Raw device	All locally attached raw disks and partitions.	

Type	Supported Targets	Unsupported Targets
Local file system	`ext2`, `ext3`, `ext4`, `btrfs` and `xfs` file systems on directly attached disk drives, hardware RAID logical drives, LVM devices, and `mdraid` arrays.	Any local file system not explicitly listed as supported in this table, including the `auto` type (automatic file system detection).
Remote directory	Remote directories accessed using the **NFS** or **SSH** protocol over `IPv4`.	Remote directories on the `rootfs` file system accessed using the **NFS** protocol.
Remote directories accessed using the `iSCSI` protocol over both hardware and software initiators.	Remote directories accessed using the `iSCSI` protocol on `be2iscsi` hardware.	Multipath-based storages.
		Remote directories accessed over `IPv6`.
		Remote directories accessed using the **SMB** or `CIFS` protocol.
		Remote directories accessed using the **FCoE** (*Fibre Channel over Ethernet*) protocol.
		Remote directories accessed using wireless network interfaces.

1.7.4. Supported kdump filtering levels

To reduce the size of the dump file, kdump uses the `makedumpfile` core collector to compress the data and optionally leave out irrelevant information. The table below contains a complete list of filtering levels that are currently supported by the `makedumpfile` utility.

For instructions on how to configure the core collector on the command line, see Section 1.2.2.3, "Configuring the core collector". For information on how to do so in the graphical user interface, see Section 1.2.3.3, "Configuring the core collector".

Table 1.4. Supported Filtering Levels

Option	Description
1	Zero pages
2	Cache pages
4	Cache private

Option	Description
8	User pages
16	Free pages

NOTE

The **makedumpfile** command supports removal of transparent huge pages and hugetlbfs pages on Red Hat Enterprise Linux 7.3 and later. Consider both these types of hugepages User Pages and remove them using the **-8** level.

1.7.5. Supported default actions

By default, when kdump fails to create a core dump, the operating system reboots. You can, however, configure kdump to perform a different operation in case it fails to save the core dump to the primary target. The table below lists all default actions that are currently supported by kdump.

For detailed information on how to set up the default action on the command line, see Section 1.2.2.4, "Configuring the default action". For information on how to do so in the graphical user interface, see Section 1.2.3.4, "Configuring the default action".

Table 1.5. Supported Default Actions

Option	Description
`dump_to_rootfs`	Attempt to save the core dump to the root file system. This option is especially useful in combination with a network target: if the network target is unreachable, this option configures kdump to save the core dump locally. The system is rebooted afterwards.
`reboot`	Reboot the system, losing the core dump in the process.
`halt`	Halt the system, losing the core dump in the process.
`poweroff`	Power off the system, losing the core dump in the process.
`shell`	Run a shell session from within the initramfs, allowing the user to record the core dump manually.

1.7.6. Estimating kdump size

When planning and building your **kdump** environment it is necessary to know how much space is required for the dump file before one is produced. The **makedumpfile** command can help with this.

The **--mem-usage** option provides a useful report about excludable pages, that can be used to determine which dump level you want to assign. Run this command when the system is under representative load, otherwise **makedumpfile** returns a smaller value than is expected in your production environment.

```
[root@hostname ~]# makedumpfile --mem-usage /proc/kcore

TYPE            PAGES                   EXCLUDABLE      DESCRIPTION
----------------------------------------------------------------------
ZERO            501635                  yes             Pages filled
with zero
CACHE           51657                   yes             Cache pages
CACHE_PRIVATE   5442                    yes             Cache pages +
private
USER            16301                   yes             User process
pages
FREE            77738211                yes             Free pages
KERN_DATA       1333192                 no              Dumpable kernel
data
```

IMPORTANT

The **makedumpfile** command reports in **pages**. This means that you must calculate the size of memory in use against the kernel page size, which in the Red Hat Enterprise Linux kernel, is 4 kilobytes for AMD64 and Intel 64 architectures, and 64 kilobytes for IBM POWER architecture.

https://access.redhat.com/articles/1351013

1.8. PORTAL LABS RELEVANT TO KDUMP

The Portal Labs are small web applications that can help system administrators perform several system tasks. There are currently two labs focused on Kdump. The Kdump Helper and the Kernel Oops Analyzer.

1.8.1. Kdump helper

The Kdump Helper is a series of questions and actions that assist in preparing the configuration files for **kdump**.

The Lab’s workflow includes steps for both clustered and standalone environments.

1.8.2. Kernel oops analyzer

The Kernel Oops Analyzer is a tool to process Oops messages and search for known solutions without having to unwind the crash dump stack.

The Kernel Oops Analyzer uses information from **makedumpfile** to compare the oops message from a crashed machine with known issues in the knowledge base. This can enable System Administrators to rule out known issues quickly after an unexpected outage, and before opening a support ticket for a further analysis.

CHAPTER 2. WORKING WITH KERNEL MODULES

This Chapter explains

- What a kernel module is
- How to use the **kmod** utilities to manage modules and their dependencies
- How to configure module parameters to control the behavior of your kernel modules
- How to load modules at boot time

NOTE

In order to use the kernel module utilities described in this chapter, first ensure the **kmod** package is installed on your system by running, as root:

```
# yum install kmod
```

2.1. WHAT IS A KERNEL MODULE?

The Linux kernel is a monolithic type by design. However, it is compiled with optional or additional modules as required by each use case. This means that you can extend the kernel's capabilities through the use of dynamically-loaded *kernel modules*. A kernel module can provide:

- a device driver which adds support for new hardware; or,
- support for a file system such as **GFS2** or **NFS**.

Like the kernel itself, modules can take parameters that customize their behavior, though the default parameters work well in most cases. User-space tools can list the modules currently loaded into a running kernel; query all available modules for available parameters and module-specific information; and load or unload (remove) modules dynamically into or from a running kernel. Many of these utilities, which are provided by the **kmod** package, take module dependencies into account when performing operations so that manual dependency-tracking is rarely necessary.

On modern systems, kernel modules are automatically loaded by various mechanisms when the conditions call for it. However, there are occasions when it is necessary to load or unload modules manually, such as when one module is preferred over another although either could provide basic functionality, or when a module is misbehaving.

2.2. LISTING CURRENTLY-LOADED MODULES

You can list all kernel modules that are currently loaded into the kernel by running the **lsmod** command, for example:

```
# lsmod
Module                  Size  Used by
tcp_lp                 12663  0
bnep                   19704  2
bluetooth             372662  7 bnep
rfkill                 26536  3 bluetooth
fuse                   87661  3
ebtable_broute         12731  0
```

```
bridge                 110196  1 ebtable_broute
stp                     12976  1 bridge
llc                     14552  2 stp,bridge
ebtable_filter          12827  0
ebtables                30913  3 ebtable_broute,ebtable_nat,ebtable_filter
ip6table_nat            13015  1
nf_nat_ipv6             13279  1 ip6table_nat
iptable_nat             13011  1
nf_conntrack_ipv4       14862  4
nf_defrag_ipv4          12729  1 nf_conntrack_ipv4
nf_nat_ipv4             13263  1 iptable_nat
nf_nat                  21798  4
nf_nat_ipv4,nf_nat_ipv6,ip6table_nat,iptable_nat
[output truncated]
```

The **lsmod** output specifies three columns:

- **Module**
  - The name of a kernel module currently loaded in memory
- **Size**
  - The amount of memory it uses in kilobytes
- **Used by**
  - A field which contains two items of information about dependencies
    - A decimal number representing how many dependencies there are on the **Module** field
    - A comma separated string of dependent **Module** names. Using this list, you can first unload all the modules depending the module you want to unload.

Finally, note that **lsmod** output is less verbose and considerably easier to read than the content of the **/proc/modules** pseudo-file.

2.3. DISPLAYING INFORMATION ABOUT A MODULE

You can display detailed information about a kernel module using the **modinfo** ***module_name*** command.

NOTE

When entering the name of a kernel module as an argument to one of the **kmod** utilities, do not append a **.ko** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

Example 2.1. Listing information about a kernel module with lsmod

To display information about the **e1000e** module, which is the Intel PRO/1000 network driver, enter the following command as **root**:

```
# modinfo e1000e
filename:       /lib/modules/3.10.0-
```

```
121.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000e/e1000e.ko
version:        2.3.2-k
license:        GPL
description:    Intel(R) PRO/1000 Network Driver
author:         Intel Corporation,
```

CHAPTER 3. MANUALLY UPGRADING THE KERNEL

The Red Hat Enterprise Linux kernel is custom-built by the Red Hat Enterprise Linux kernel team to ensure its integrity and compatibility with supported hardware. Before Red Hat releases a kernel, it must first pass a rigorous set of quality assurance tests.

Red Hat Enterprise Linux kernels are packaged in the RPM format so that they are easy to upgrade and verify using the **Yum** or **PackageKit** package managers. **PackageKit** automatically queries the Red Hat Content Delivery Network servers and informs you of packages with available updates, including kernel packages.

This chapter is therefore **only** useful for users who need to manually update a kernel package using the `rpm` command instead of `yum`.

WARNING

Whenever possible, use either the **Yum** or **PackageKit** package manager to install a new kernel because they always **install** a new kernel instead of replacing the current one, which could potentially leave your system unable to boot.

WARNING

Custom kernels are **not** supported by Red Hat. However, guidance can be obtained from the knowledgebase document https://access.redhat.com/solutions/25039.

For more information on installing kernel packages with **yum**, see https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/ch-yum.html#sec-Updating_Packages.

For information on Red Hat Content Delivery Network, see https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Subscription_and_Support-Registering_a_System_and_Managing_Subscriptions.html.

3.1. OVERVIEW OF KERNEL PACKAGES

Red Hat Enterprise Linux contains the following kernel packages:

- **kernel** – Contains the kernel for single-core, multi-core, and multi-processor systems.
- **kernel-debug** – Contains a kernel with numerous debugging options enabled for kernel diagnosis, at the expense of reduced performance.
- **kernel-devel** – Contains the kernel headers and makefiles sufficient to build modules against the **kernel** package.

- **kernel-debug-devel** – Contains the development version of the kernel with numerous debugging options enabled for kernel diagnosis, at the expense of reduced performance.
- **kernel-doc** – Documentation files from the kernel source. Various portions of the Linux kernel and the device drivers shipped with it are documented in these files. Installation of this package provides a reference to the options that can be passed to Linux kernel modules at load time.
  By default, these files are placed in the `/usr/share/doc/kernel-doc-kernel_version/` directory.
- **kernel-headers** – Includes the C header files that specify the interface between the Linux kernel and user-space libraries and programs. The header files define structures and constants that are needed for building most standard programs.
- **linux-firmware** – Contains all of the firmware files that are required by various devices to operate.
- **perf** – This package contains the **perf** tool, which enables performance monitoring of the Linux kernel.
- **kernel-abi-whitelists** – Contains information pertaining to the Red Hat Enterprise Linux kernel ABI, including a lists of kernel symbols that are needed by external Linux kernel modules and a **yum** plug-in to aid enforcement.
- **kernel-tools** – Contains tools for manipulating the Linux kernel and supporting documentation.

3.2. PREPARING TO UPGRADE

Before upgrading the kernel, it is recommended that you take some precautionary steps.

First, ensure that working boot media exists for the system in case a problem occurs. If the boot loader is not configured properly to boot the new kernel, you can use this media to boot into Red Hat Enterprise Linux

USB media often comes in the form of flash devices sometimes called *pen drives*, *thumb disks*, or *keys*, or as an externally-connected hard disk device. Almost all media of this type is formatted as a **VFAT** file system. You can create bootable USB media on media formatted as `ext2`, `ext3`, `ext4`, or **VFAT**.

You can transfer a distribution image file or a minimal boot media image file to USB media. Make sure that sufficient free space is available on the device. Around 4 GB is required for a distribution DVD image, around 700 MB for a distribution CD image, or around 10 MB for a minimal boot media image.

You must have a copy of the `boot.iso` file from a Red Hat Enterprise Linux installation DVD, or installation CD-ROM #1, and you need a USB storage device formatted with the **VFAT** file system and around 16 MB of free space.

For more information on using USB storage devices, review https://access.redhat.com/solutions/624423 How to format a USB key and https://access.redhat.com/solutions/39373 How to manually mount a USB flash drive in a non-graphical environment.

The following procedure does not affect existing files on the USB storage device unless they have the same path names as the files that you copy onto it. To create USB boot media, perform the following commands as the `root` user:

1. Install the **syslinux** package if it is not installed on your system. To do so, as root, run the **`yum install syslinux`** command.

2. Install the **SYSLINUX** bootloader on the USB storage device:

```
# syslinux /dev/sdX1
```

   ...where *sdX* is the device name.

3. Create mount points for **`boot.iso`** and the USB storage device:

```
# mkdir /mnt/isoboot /mnt/diskboot
```

4. Mount **`boot.iso`**:

```
# mount -o loop boot.iso /mnt/isoboot
```

5. Mount the USB storage device:

```
# mount /dev/sdX1 /mnt/diskboot
```

6. Copy the **ISOLINUX** files from the **`boot.iso`** to the USB storage device:

```
# cp /mnt/isoboot/isolinux/* /mnt/diskboot
```

7. Use the **`isolinux.cfg`** file from **`boot.iso`** as the **`syslinux.cfg`** file for the USB device:

```
# grep -v local /mnt/isoboot/isolinux/isolinux.cfg >
/mnt/diskboot/syslinux.cfg
```

8. Unmount **`boot.iso`** and the USB storage device:

```
# umount /mnt/isoboot /mnt/diskboot
```

9. Reboot the machine with the boot media and verify that you are able to boot with it before continuing.

Alternatively, on systems with a floppy drive, you can create a boot diskette by installing the **mkbootdisk** package and running the **`mkbootdisk`** command as **`root`**. See **`man mkbootdisk`** man page after installing the package for usage information.

To determine which kernel packages are installed, execute the command **`yum list installed "kernel-*"`** at a shell prompt. The output comprises some or all of the following packages, depending on the system's architecture, and the version numbers might differ:

```
# yum list installed "kernel-*"
kernel.x86_64                   3.10.0-54.0.1.el7           @rhel7/7.0
kernel-devel.x86_64             3.10.0-54.0.1.el7           @rhel7
kernel-headers.x86_64           3.10.0-54.0.1.el7           @rhel7/7.0
```

From the output, determine which packages need to be downloaded for the kernel upgrade. For a single processor system, the only required package is the **kernel** package. See Section 3.1, "Overview of kernel packages" for descriptions of the different packages.

3.3. DOWNLOADING THE UPGRADED KERNEL

There are several ways to determine if an updated kernel is available for the system.

- Security Errata – See https://access.redhat.com/site/security/updates/active/ for information on security errata, including kernel upgrades that fix security issues.

- The Red Hat Content Delivery Network – For a system subscribed to the Red Hat Content Delivery Network, the **yum** package manager can download the latest kernel and upgrade the kernel on the system. The **Dracut** utility creates an initial RAM disk image if needed, and configure the boot loader to boot the new kernel. For more information on installing packages from the Red Hat Content Delivery Network, see https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/ch-yum.html. For more information on subscribing a system to the Red Hat Content Delivery Network, see https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Subscription_and_Support-Registering_a_System_and_Managing_Subscriptions.html.

If **yum** was used to download and install the updated kernel from the Red Hat Network, follow the instructions in Section 3.5, “Verifying the initial RAM disk image” and Section 3.6, “Verifying the boot loader” only, **do not** change the kernel to boot by default. Red Hat Network automatically changes the default kernel to the latest version. To install the kernel manually, continue to Section 3.4, “Performing the upgrade”.

3.4. PERFORMING THE UPGRADE

After retrieving all of the necessary packages, it is time to upgrade the existing kernel.

IMPORTANT

It is strongly recommended that you keep the old kernel in case there are problems with the new kernel.

At a shell prompt, change to the directory that contains the kernel RPM packages. Use `-i` argument with the `rpm` command to keep the old kernel. Do **not** use the `-U` option, since it overwrites the currently installed kernel, which creates boot loader problems. For example:

```
# rpm -ivh kernel-kernel_version.arch.rpm
```

The next step is to verify that the initial RAM disk image has been created. See Section 3.5, “Verifying the initial RAM disk image” for details.

3.5. VERIFYING THE INITIAL RAM DISK IMAGE

The job of the initial RAM disk image is to preload the block device modules, such as for IDE, SCSI or RAID, so that the root file system, on which those modules normally reside, can then be accessed and mounted. On Red Hat Enterprise Linux 7 systems, whenever a new kernel is installed using either the **Yum**, **PackageKit**, or **RPM** package manager, the **Dracut** utility is always called by the installation scripts to create an *initramfs* (initial RAM disk image).

If you make changes to the kernel attributes by modifying the `/etc/sysctl.conf` file or another `sysctl` configuration file, and if the changed settings are used early in the boot process, then rebuilding the Initial RAM Disk Image by running the `dracut -f` command might be necessary. An example is if you have made changes related to networking and are booting from network-attached storage.

On all architectures other than IBM eServer System i (see the section called “Verifying the initial RAM disk image and kernel on IBM eServer System i”), you can create an **initramfs** by running the **dracut** command. However, you usually do not need to create an **initramfs** manually: this step is automatically performed if the kernel and its associated packages are installed or upgraded from RPM packages distributed by Red Hat.

You can verify that an **initramfs** corresponding to your current kernel version exists and is specified correctly in the **grub.cfg** configuration file by following this procedure:

Verifying the Initial RAM Disk Image

1. As **root**, list the contents in the **/boot** directory and find the kernel (**vmlinuz-*kernel_version***) and **initramfs-*kernel_version*** with the latest (most recent) version number:

   Example 3.1. Ensuring that the kernel and initramfs versions match

   ```
   # ls /boot
   config-3.10.0-67.el7.x86_64
   config-3.10.0-78.el7.x86_64
   efi
   grub
   grub2
   initramfs-0-rescue-07f43f20a54c4ce8ada8b70d33fd001c.img
   initramfs-3.10.0-67.el7.x86_64.img
   initramfs-3.10.0-67.el7.x86_64kdump.img
   initramfs-3.10.0-78.el7.x86_64.img
   initramfs-3.10.0-78.el7.x86_64kdump.img
   initrd-plymouth.img
   symvers-3.10.0-67.el7.x86_64.gz
   symvers-3.10.0-78.el7.x86_64.gz
   System.map-3.10.0-67.el7.x86_64
   System.map-3.10.0-78.el7.x86_64
   vmlinuz-0-rescue-07f43f20a54c4ce8ada8b70d33fd001c
   vmlinuz-3.10.0-67.el7.x86_64
   vmlinuz-3.10.0-78.el7.x86_64
   ```

   Example 3.1, “Ensuring that the kernel and initramfs versions match” shows that:

   - we have three kernels installed (or, more correctly, three kernel files are present in the **/boot** directory),
   - the latest kernel is **vmlinuz-3.10.0-78.el7.x86_64**, and
   - an **initramfs** file matching our kernel version, **initramfs-3.10.0-78.el7.x86_64kdump.img**, also exists.

> **IMPORTANT**
>
> In the **/boot** directory you might find several **initramfs-*kernel_version*kdump.img** files. These are special files created by the **Kdump** mechanism for kernel debugging purposes, are not used to boot the system, and can safely be ignored. For more information on **kdump**, see the Red Hat Enterprise Linux 7 Kernel Crash Dump Guide .

2. If your **initramfs-*kernel_version*** file does not match the version of the latest kernel in the **/boot** directory, or, in certain other situations, you might need to generate an **initramfs** file with the **Dracut** utility. Simply invoking **dracut** as **root** without options causes it to generate an **initramfs** file in **/boot** for the latest kernel present in that directory:

```
# dracut
```

You must use the **-f**, **--force** option if you want **dracut** to overwrite an existing **initramfs** (for example, if your **initramfs** has become corrupt). Otherwise **dracut** refuses to overwrite the existing **initramfs** file:

```
# dracut
          Does not override existing initramfs (/boot/initramfs-
3.10.0-78.el7.x86_64.img) without --force
```

You can create an initramfs in the current directory by calling **dracut *initramfs_name kernel_version***:

```
# dracut "initramfs-$(uname -r).img" $(uname -r)
```

If you need to specify specific kernel modules to be preloaded, add the names of those modules (minus any file name suffixes such as **.ko**) inside the parentheses of the **add_dracutmodules+="*module more_modules* "** directive of the **/etc/dracut.conf** configuration file. You can list the file contents of an **initramfs** image file created by dracut by using the **lsinitrd *initramfs_file*** command:

```
# lsinitrd /boot/initramfs-3.10.0-78.el7.x86_64.img
Image: /boot/initramfs-3.10.0-78.el7.x86_64.img: 11M
====================================================================
====
dracut-033-68.el7
====================================================================
====

drwxr-xr-x  12 root     root            0 Feb  5 06:35 .
drwxr-xr-x   2 root     root            0 Feb  5 06:35 proc
lrwxrwxrwx   1 root     root           24 Feb  5 06:35 init ->
/usr/lib/systemd/systemd
drwxr-xr-x  10 root     root            0 Feb  5 06:35 etc
drwxr-xr-x   2 root     root            0 Feb  5 06:35
usr/lib/modprobe.d
[output truncated]
```

See **man dracut** and **man dracut.conf** for more information on options and usage.

3. Examine the **/boot/grub2/grub.cfg** configuration file to ensure that an **initramfs-*kernel_version*.img** file exists for the kernel version you are booting. For example:

```
# grep initramfs /boot/grub2/grub.cfg
initrd16 /initramfs-3.10.0-123.el7.x86_64.img
initrd16 /initramfs-0-rescue-6d547dbfd01c46f6a4c1baa8c4743f57.img
```

See Section 3.6, “Verifying the boot loader” for more information.

Verifying the initial RAM disk image and kernel on IBM eServer System i

On IBM eServer System i machines, the initial RAM disk and kernel files are combined into a single file, which is created with the **addRamDisk** command. This step is performed automatically if the kernel and its associated packages are installed or upgraded from the RPM packages distributed by Red Hat thus, it does not need to be executed manually. To verify that it was created, run the following command as **root** to make sure the **/boot/vmlinitrd-*kernel_version*** file already exists:

```
# ls -l /boot/
```

The *kernel_version* needs to match the version of the kernel just installed.

Reversing the changes made to the initial RAM disk image

In some cases, for example, if you misconfigure the system and it no longer boots, you need to reverse the changes made to the Initial RAM Disk Image by following this procedure:

Reversing Changes Made to the Initial RAM Disk Image

1. Reboot the system choosing the rescue kernel in the GRUB menu.
2. Change the incorrect setting that caused the **initramfs** to malfunction.
3. Recreate the **initramfs** with the correct settings by running the following command as root:

```
# dracut --kver kernel_version --force
```

The above procedure might be useful if, for example, you incorrectly set the **vm.nr_hugepages** in the **sysctl.conf** file. Because the **sysctl.conf** file is included in **initramfs**, the new **vm.nr_hugepages** setting gets applied in **initramfs** and causes rebuilding of the **initramfs**. However, because the setting is incorrect, the new **initramfs** is broken and the newly built kernel does not boot, which necessitates correcting the setting using the above procedure.

Listing the contents of the initial RAM disk image

To list the files that are included in the **initramfs**, run the following command as root:

```
# lsinitrd
```

To only list files in the **/etc** directory, use the following command:

```
# lsinitrd | grep etc/
```

To output the contents of a specific file stored in the **initramfs** for the current kernel, use the **-f** option:

```
# lsinitrd -f filename
```

For example, to output the contents of **sysctl.conf**, use the following command:

```
# lsinitrd -f /etc/sysctl.conf
```

To specify a kernel version, use the **--kver** option:

```
# lsinitrd --kver kernel_version -f /etc/sysctl.conf
```

For example, to list the information about kernel version 3.10.0-327.10.1.el7.x86_64, use the following command:

```
# lsinitrd --kver 3.10.0-327.10.1.el7.x86_64 -f /etc/sysctl.conf
```

3.6. VERIFYING THE BOOT LOADER

You can install a kernel either with the **yum** command or with the **rpm** command.

When you install a kernel using **rpm**, the kernel package creates an entry in the boot loader configuration file for that new kernel.

Note that both commands configure the new kernel to boot as the default kernel only when you include the following setting in the **/etc/sysconfig/kernel** configuration file:

```
DEFAULTKERNEL=kernel
UPDATEDEFAULT=yes
```

The **DEFAULTKERNEL** option specifies the default kernel package type. The **UPDATEDEFAULT** option specifies whether the new kernel package makes the new kernels the default.

CHAPTER 4. WORKING WITH KPATCH

4.1. WHAT IS KPATCH?

kpatch is a live kernel patching solution that allows you to patch a running kernel without rebooting or restarting any processes. It enables system administrators to apply critical security patches to the kernel immediately, without having to wait for long-running tasks to complete, for users to log off, or for scheduled downtime. It gives more control over uptime without sacrificing security or stability.

> **WARNING**
>
> Some incompatibilities exist between **kpatch** and other kernel subcomponents. Read the Section 4.4, “Limitations” section carefully before using **kpatch**.

4.2. WHAT IS THE SCOPE OF SUPPORT FOR KPATCH?

- Live kernel patching with **kpatch** is supported from Red Hat Enterprise Linux 7.2 onwards.
- Live kernel patching is supported for customers with Premium SLA subscriptions.
- Live kernel patching is **only** supported on the active Red Hat Enterprise Linux 7 maintenance stream that is within the current async errata phase. See Red Hat Enterprise Linux Life Cycle for information about current support phases.
- Live kernel patching is **not** available on Extended Update Support releases.
- **kpatch** is **not** supported on the Red Hat Enterprise Linux Realtime (RT) kernel.
- **kpatch** is **not** supported on Red Hat Enterprise Linux 5 or Red Hat Enterprise Linux 6.
- Only **one** live kernel patch can be installed on the kernel at any given time.
- It is not certain that all issues are covered under live kernel patching, including hardware enablement.

4.3. ACCESS AND DELIVERY

Live kernel patching capability is implemented via a kernel module (kmod) that is delivered as an RPM package. The **kpatch** utility is used to install and remove the kernel modules for live kernel patching.

Customers with Premium subscriptions are eligible to request a live kernel patch as part of an accelerated fix solution from Red Hat Support.

Customers with Premium subscriptions who typically used **hotfix** kernels which required a reboot can now request a **kpatch** kmod that requires no down time. The **kpatch** patch is supported for 30 days after the errata that contains the fix is released, in the same manner as **hotfix** kernels.

Customers have the option to open a support case directly in the Red Hat Customer Portal and discuss appropriate accelerated fix options.

4.4. LIMITATIONS

kpatch is not a general-purpose kernel upgrade mechanism. It is used for applying simple security and bug fix updates when rebooting the system is not immediately possible.

Do **not** use the **SystemTap** or **kprobe** tools during or after loading a patch. The patch could fail to take effect until after the probe has been removed.

Do **not** directly access the **ftrace** output file, for example by running **cat /sys/kernel/debug/tracing/trace**. The **trace-cmd** utility is supported instead.

Do **not** suspend or hibernate the system when using **kpatch**. This can result in a patch being temporarily disabled for a small amount of time.

NOTE

Red Hat is actively working to remove many of the limitations of **kpatch** for future releases.

4.5. HOW DO I ENABLE AND USE KPATCH?

The components of **kpatch** are as follows:

Components of kpatch

A systemd Integration Point

A **systemd** service called **kpatch.service** that is required by **multiuser.target** which loads the kpatch modules at boot time.

A Patch Module

- The delivery mechanism for new kernel code.
- This is another kernel module that is named to match the **kpatch** being applied.
- The patch module contains the compiled code from the latest hot fixes introduced to the kernel.
- The patch modules register with the core module, **kpatch.ko** and provide information about original functions to be replaced, with corresponding pointers to the replacement functions.

The kpatch Utility

A command-line tool which allows you to manage patch modules.

4.5.1. Installing the kpatch tools

Before you can install a **kpatch** module, you must install the **kpatch** tools package. To do so, type the following at a shell prompt as **root**

```
# yum install kpatch
```

4.5.2. Installing a kpatch hot fix

To install a **kpatch** hot fix, install the supplied **kpatch-patch** RPM package with **yum**. For example, to install **kpatch-patch-7.0-1.el7.x86_64.rpm**, issue the following command as **root**

```
# yum install kpatch-patch-7.0-1.el7.x86_64.rpm
```

4.5.3. Listing installed kpatch hot fixes

To verify a patch is loaded and installed, run the **kpatch list** command as **root**:

```
# kpatch list
Loaded patch modules:
kpatch_7_0_1_el7
Installed patch modules:
kpatch-7-0-1-el7.ko (3.10.0-121.el7.x86_64)
kpatch-7-0-1-el7.ko (3.10.0-123.el7.x86_64)
```

The example output above shows that the module has been loaded into the kernel, meaning the kernel is now patched with the latest hot fixes contained in the **kpatch-patch-7.0-1.el7.x86_64.rpm** package. It also shows that it has been saved to **/var/lib/kpatch** to be loaded by **systemd** during future reboots for kernel versions 3.10.0-121 and 3.10.0-123.

4.5.4. Updating a kpatch hot fix

If a new version of the **kpatch-patch** RPM package is later released, upgrade the applied patch with **yum** For example, to upgrade to **kpatch-patch-7.0-2.el7.x86_64.rpm** run as **root**

```
# yum update kpatch-patch-7.0-2.el7.x86_64.rpm
```

Upgrading the RPM package automatically replaces the patch module in the running kernel and updates the **/var/lib/kpatch** structures used by **systemd** on reboot.

NOTE

The patch modules in the **kpatch-patch** RPM packages are cumulative. Consequently, you could skip installing kpatch-patch-7.0-1 and instead start with installing kpatch-patch-7.0-2 if it were available.

Loading a patch module sets the **TAINT_LIVEPATCH** kernel taint flag (which corresponds to bit 15) and **TAINT_OOT_MODULE** (which corresponds to bit 12). To determine whether the kernel has been patched, use the **cat** command to display the contents of **/proc/sys/kernel/tainted** and check the value in the file. Unless you have other taint flags set, the value is **36864** when the kernel is patched.

4.5.5. Removing a kpatch Hot Fix

To unload a **kpatch** module from the running kernel, use the **kpatch unload** command, specifying the name of the patch module. For example, to unload **kpatch_7_0_2_el7** type the following at a shell prompt as **root**

```
# kpatch unload kpatch_7_0_2_el7
```

The patch module must also be uninstalled from **/var/lib/kpatch** with the **kpatch** uninstall command as follows:

```
# kpatch uninstall kpatch_7_0_2_el7
```

The default behavior of this command is to uninstall the **kpatch** from the kernel corresponding to the current kernel version, but you can specify a different kernel version by using the **kernel-version** option:

```
# kpatch uninstall --kernel-version 3.10.0-121.el7.x86_64 kpatch_7_0_2_el7
```

Alternatively, you can uninstall the **kpatch-patch** RPM package, which also removes the patch module from **/var/lib/kpatch**

> **NOTE**
>
> Uninstalling the RPM package does not unload the **kpatch** module from the kernel. An explicit call to **kpatch unload** as described above is required.

4.6. HOW DOES KPATCH WORK?

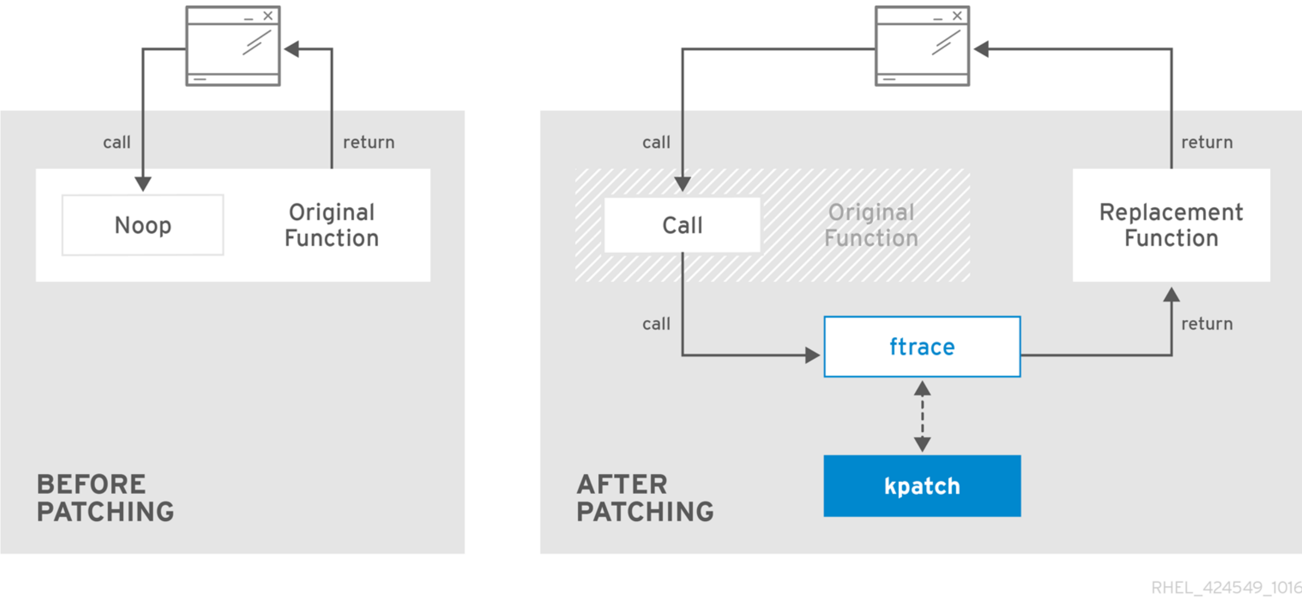
The **kpatch** utilities use **ftrace** for arbitrary remapping of pointers to kernel functions. When a live kernel patch is applied to a system, the following things happen:

1. The new compiled code in the module is copied to **/var/lib/kpatch** and registered for re-application to the kernel via **systemd** on next boot.

2. The **kpatch** module is loaded into the running kernel and the new functions are registered to the **ftrace** mechanism with a pointer to the location in memory of the new code.

3. When the kernel accesses the patched function, it is redirected to the **ftrace** mechanism which bypasses the original functions and redirects the kernel to patched version of the function.

Figure 4.1. How kpatch Works



4.7. ARE ANY THIRD-PARTY LIVE PATCHING SOLUTIONS SUPPORTED?

Although several third-party and proprietary tools that provide live kernel patching are available, Red Hat only supports **kpatch** and the RPM modules supplied through your Red Hat support contract. Red Hat cannot support third-party live-patches, however requests for Engineering and an official Red Hat **kpatch** can be opened at any time.

For any Red Hat review of a third-party live-patch, the source code would need to be supplied to determine if they meet the following criteria:

1. Impact the same subsystems and codepaths as the kernel encountered during a failure.

2. Applying the same patches using supported means, within the applicable streams, result in no failure being encountered.

Red Hat recommends that you open a case with the live kernel patching vendor at the outset of any investigation in which a root cause determination is necessary. This allows the source code to be supplied if the vendor allows, and for their support organization to provide assistance in root cause determination prior to escalating the investigation to Red Hat Support.

For any system running with third-party live kernel patches, Red Hat reserves the right to ask for reproduction with Red Hat shipped and supported software. In the event that this is not possible, we require a similar system and workload be deployed on your test environment without live patches applied, to confirm if the same behavior is observed.

For more information about third-party software support policies, see How does Red Hat Global Support Services handle third-party software, drivers, and/or uncertified hardware/hypervisors or guest operating systems? in the knowledgebase at https://access.redhat.com/articles/1067.

CHAPTER 5. WORKING WITH SYSCTL AND KERNEL TUNABLES

5.1. WHAT IS A KERNEL TUNABLE?

Kernel tunables are used to customize the behavior of Red Hat Enterprise Linux at boot, or on demand while the system is running. Some hardware parameters are specified at boot time only and cannot be altered once the system is running, most however, can be altered as required and set permanent for the next boot.

5.2. HOW TO WORK WITH KERNEL TUNABLES

There are three ways to modify kernel tunables.

1. Using the `sysctl` command
2. By manually modifying configuration files in the `/etc/sysctl.d/` directory
3. Through a shell, interacting with the virtual file system mounted at `/proc/sys`

NOTE

Not all boot time parameters are under control of the sysfs subsystem, some hardware specific option must be set on the kernel command line, the Kernel Parameters section of this guide addresses those options

5.2.1. Using the sysctl command

The `sysctl` command is used to list, read, and set kernel tunables. It can filter tunables when listing or reading and set tunables temporarily or permanently.

1. Listing variables

```
# sysctl -a
```

2. Reading variables

```
# sysctl kernel.version +
kernel.version = #1 SMP Fri Jan 19 13:19:54 UTC 2018
```

3. Writing variables temporarily

```
# sysctl <tunable class>.<tunable>=<value>
```

4. Writing variables permanently

```
# sysctl -w <tunable class>.<tunable>=<value>
```

5.2.2. Modifying files in /etc/sysctl.d

To override a default at boot, you can also manually populate files in `/etc/sysctl.d`.

1. Create a new file in **/etc/sysctl.d**

```
# vim /etc/sysctl.d/99-custom.conf
```

2. Include the variables you wish to set, one per line, in the following form

```
<tunable class>.<tunable> = <value> +
<tunable class>.<tunable> = <value>
```

3. Save the file

4. Either reboot the machine to make the changes take effect
   or
   Execute **sysctl -p /etc/sysctl.d/99-custom.conf** to apply the changes without rebooting

5.3. WHAT TUNABLES CAN BE CONTROLLED?

Tunables are diveded into groups by kernel sybsystem. A Red Hat Enterprise Linux system has the following classes of tunables:

Table 5.1. Table of sysctl interfaces

Class	Subsystem
abi	Execution domains and personalities
crypto	Cryptographic interfaces
debug	Kernel debugging interfaces
dev	Device specific information
fs	Global and specific filesystem tunables
kernel	Global kernel tunables
net	Network tunables
sunrpc	Sun Remote Procedure Call (NFS)
user	User Namespace limits
vm	Tuning and management of memory, buffer, and cache

5.3.1. Network interface tunables

System administrators are able to adjust the network configuration on a running system through the networking tunables.

Networking tunables are included in the `/proc/sys/net` directory, which contains multiple subdirectories for various networking topics. To adjust the network configuration, system administrators need to modify the files within such subdirectories.

The most frequently used directories are:

1. `/proc/sys/net/core/`
2. `/proc/sys/net/ipv4/`

The `/proc/sys/net/core/` directory contains a variety of settings that control the interaction between the kernel and networking layers. By adjusting some of those tunables, you can improve performance of a system, for example by increasing the size of a receive queue, increasing the maximum connections or the memory dedicated to network interfaces. Note that the performance of a system depends on different aspects according to the individual issues.

The `/proc/sys/net/ipv4/` directory contains additional networking settings, which are useful when preventing attacks on the system or when using the system to act as a router. The directory contains both IP and TCP variables. For detailed explaination of those variables, see `/usr/share/doc/kernel-doc-<version>/Documentation/networking/ip-sysctl.txt`.

Other directories within the `/proc/sys/net/ipv4/` directory cover different aspects of the network stack:

1. `/proc/sys/net/ipv4/conf/` - allows you to configure each system interface in different ways, including the use of default settings for unconfigured devices and settings that override all special configurations
2. `/proc/sys/net/ipv4/neigh/` - contains settings for communicating with a host directly connected to the system and also contains different settings for systems more than one step away
3. `/proc/sys/net/ipv4/route/` - contains specifications that apply to routing with any interfaces on the system

This list of network tunables is relevant to IPv4 interfaces and are accessible from the `/proc/sys/net/ipv4/{all,<interface_name>}/` directory.

log_martians

Log packets with impossible addresses to kernel log.

Type	Default
Boolean	0

Enabled if one or more of `conf/{all,interface}/log_martians` is set to TRUE

Further Resources

- What is the kernel parameter net.ipv4.conf.all.log_martians for?
- Why do I see "martian source" logs in the messages file ?

accept_redirects

Accept ICMP redirect messages.

Type	Default
Boolean	1

accept_redirects for the interface is enabled under the following conditions:

- Both `conf/{all,interface}/accept_redirects` are TRUE (when forwarding for the interface is enabled)
- At least one of `conf/{all,interface}/accept_redirects` is TRUE (forwarding for the interface is disabled)

Further Resources

- How to enable or disable ICMP redirects

forwarding

Enable IP forwarding on an interface.

Type	Default
Boolean	0

Further Resources

- Turning on Packet Forwarding and Nonlocal Binding

mc_forwarding

Do multicast routing.

Type	Default
Boolean	0

- Read only value
- A multicast routing daemon is required.
- `conf/all/mc_forwarding` must also be set to TRUE to enable multicast routing for the interface

Further Resources

- For an explanation of the read only behavior, see Why system reports "permission denied on key" while setting the kernel parameter "net.ipv4.conf.all.mc_forwarding"?

medium_id

Arbitrary value used to differentiate the devices by the medium they are attached to.

Type	Default
Integer	0

Notes

- Two devices on the same medium can have different id values when the broadcast packets are received only on one of them.
- The default value 0 means that the device is the only interface to its medium
- value of -1 means that medium is not known.
- Currently, it is used to change the proxy_arp behavior:
- the proxy_arp feature is enabled for packets forwarded between two devices attached to different media.

Further Resources - For examples, see Using the "medium_id" feature in Linux 2.2 and 2.4

proxy_arp

Do proxy arp.

Type	Default
Boolean	0

proxy_arp for the interface is enabled if at least one of `conf/{all,interface}/proxy_arp` is set to TRUE, otherwise it is disabled

proxy_arp_pvlan

Private VLAN proxy arp.

Type	Default
Boolean	0

Allow proxy arp replies back to the same interface, to support features like RFC 3069

shared_media

Send(router) or accept(host) RFC1620 shared media redirects.

Type	Default
Boolean	1

Notes

- Overrides secure_redirects.
- **shared_media** for the interface is enabled if at least one of **`conf/{all,interface}/shared_media`** is set to TRUE

secure_redirects

Accept ICMP redirect messages only to gateways listed in the interface's current gateway list.

Type	Default
Boolean	1

Notes

- Even if disabled, RFC1122 redirect rules still apply.
- Overridden by shared_media.
- secure_redirects for the interface is enabled if at least one of **`conf/{all,interface}/secure_redirects`** is set to TRUE

send_redirects

Send redirects, if router.

Type	Default
Boolean	1

Notes
send_redirects for the interface is enabled if at least one of **`conf/{all,interface}/send_redirects`** is set to TRUE

bootp_relay

Accept packets with source address 0.b.c.d destined not to this host as local ones.

Type	Default
Boolean	0

Notes

- A BOOTP daemon must be enabled to manage these packets
- **`conf/all/bootp_relay`** must also be set to TRUE to enable BOOTP relay for the interface
- Not implemented, see DHCP Relay Agent in the Red Hat Enterprise Linux Networking Guide

accept_source_route

Accept packets with SRR option.

Type	Default
Boolean	1

Notes

- `conf/all/accept_source_route` must also be set to TRUE to accept packets with SRR option on the interface

accept_local

Accept packets with local source addresses.

Type	Default
Boolean	0

Notes

- In combination with suitable routing, this can be used to direct packets between two local interfaces over the wire and have them accepted properly.
- `rp_filter` must be set to a non-zero value in order for accept_local to have an effect.

route_localnet

Do not consider loopback addresses as martian source or destination while routing.

Type	Default
Boolean	0

Notes

- This enables the use of **127/8** for local routing purposes.

rp_filter

Enable source Validation

Type	Default
Integer	0

Value	Effect
0	No source validation.
1	Strict mode as defined in RFC3704 Strict Reverse Path
2	Loose mode as defined in RFC3704 Loose Reverse Path

Notes

- Current recommended practice in RFC3704 is to enable strict mode to prevent IP spoofing from DDos attacks.
- If using asymmetric routing or other complicated routing, then loose mode is recommended.
- The highest value from `conf/{all,interface}/rp_filter` is used when doing source validation on the {interface}

arp_filter

Type	Default
Boolean	0

Value	Effect
0	(default) The kernel can respond to arp requests with addresses from other interfaces. It usually makes sense, because it increases the chance of successful communication.
1	Allows you to have multiple network interfaces on the samesubnet, and have the ARPs for each interface be answered based on whether or not the kernel would route a packet from the ARP'd IP out that interface (therefore you must use source based routing for this to work). In other words it allows control of cards (usually 1) that respond to an arp request.

Note

- IP addresses are owned by the complete host on Linux, not by particular interfaces. Only for more complex setups like load-balancing, does this behavior cause problems.
- `arp_filter` for the interface is enabled if at least one of `conf/{all,interface}/arp_filter` is set to TRUE

arp_announce

Define different restriction levels for announcing the local source IP address from IP packets in ARP requests sent on interface

Type	Default
Integer	0

Value	Effect
0	(default) Use any local address, configured on any interface
1	Try to avoid local addresses that are not in the target's subnet for this interface. This mode is useful when target hosts reachable via this interface require the source IP address in ARP requests to be part of their logical network configured on the receiving interface. When we generate the request we check all our subnets that include the target IP and preserve the source address if it is from such subnet. If there is no such subnet we select source address according to the rules for level 2.
2	Always use the best local address for this target. In this mode we ignore the source address in the IP packet and try to select local address that we prefer for talks with the target host. Such local address is selected by looking for primary IP addresses on all our subnets on the outgoing interface that include the target IP address. If no suitable local address is found we select the first local address we have on the outgoing interface or on all other interfaces, with the hope we receive reply for our request and even sometimes no matter the source IP address we announce.

Notes

- The highest value from **`conf/{all,interface}/arp_announce`** is used.
- Increasing the restriction level gives more chance for receiving answer from the resolved target while decreasing the level announces more valid sender's information.

arp_ignore

Define different modes for sending replies in response to received ARP requests that resolve local target IP addresses

Type	Default
Integer	0

Value	Effect
0	(default): reply for any local target IP address, configured on any interface
1	reply only if the target IP address is local address configured on the incoming interface
2	reply only if the target IP address is local address configured on the incoming interface and both with the sender's IP address are part from same subnet on this interface

Value	Effect
3	do not reply for local addresses configured with scope host, only resolutions for global and link addresses are replied
4-7	reserved
8	do not reply for all local addresses The max value from conf/{all,interface}/arp_ignore is used when ARP request is received on the {interface}

Notes

arp_notify

Define mode for notification of address and device changes.

Type	Default
Boolean	0

Value	Effect
0	do nothing
1	Generate gratuitous arp requests when device is brought up or hardware address changes.

Notes

arp_accept

Define behavior for gratuitous ARP frames who’s IP is not already present in the ARP table

Type	Default
Boolean	0

Value	Effect
0	do not create new entries in the ARP table
1	create new entries in the ARP table.

Notes

Both replies and requests type gratuitous arp trigger the ARP table to be updated, if this setting is on. If the ARP table already contains the IP address of the gratuitous arp frame, the arp table is updated regardless if this setting is on or off.

app_solicit

The maximum number of probes to send to the user space ARP daemon via netlink before dropping back to multicast probes (see mcast_solicit).

Type	Default
Integer	0

Notes
See **mcast_solicit**

disable_policy

Disable IPSEC policy (SPD) for this interface

Type	Default
Boolean	0

needinfo

disable_xfrm

Disable IPSEC encryption on this interface, whatever the policy

Type	Default
Boolean	0

needinfo

igmpv2_unsolicited_report_interval

The interval in milliseconds in which the next unsolicited IGMPv1 or IGMPv2 report retransmit takes place.

Type	Default
Integer	10000

Notes
Milliseconds

igmpv3_unsolicited_report_interval

The interval in milliseconds in which the next unsolicited IGMPv3 report retransmit takes place.

Type	Default
Integer	1000

Notes
Milliseconds

tag

Allows you to write a number, which can be used as required.

Type	Default
Integer	0

needinfo

xfrm4_gc_thresh

The threshold at which we start garbage collecting for IPv4 destination cache entries.

Type	Default
Integer	1

Notes
At twice this value the system refuses new allocations.

CHAPTER 6. KERNEL FEATURES

This chapter explains the purpose and use of kernel features that enable many user space tools and includes resources for further investigation of those tools.

6.1. CONTROL GROUPS

6.1.1. What is a control group?

NOTE

Control Group Namespaces are a Technology Preview in Red Hat Enterprise Linux 7.5

Linux Control Groups (cgroups) enable limits on the use of system hardware, ensuring that an individual process running inside a **cgroup** only utilizes as much as has been allowed in the **cgroups** configuration.

Control Groups restrict the volume of usage on a resource that has been enabled by a **namespace**. For example, the network namespace allows a process to access a particular network card, the cgroup ensures that the process does not exceed 50% usage of that card, ensuring bandwidth is available for other processes.

Control Group Namespaces provide a virtualized view of individual cgroups through the /proc/self/ns/cgroup interface.

The purpose is to prevent leakage of privileged data from the global namespaces to the cgroup and to enable other features, such as container migration.

Because it is now much easier to associate a container with a single cgroup, containers have a much more coherent cgroup view, it also enables tasks inside the container to have a virtualized view of the cgroup it belongs to.

6.1.2. What is a namespace?

Namespaces are a kernel feature that allow a virtual view of isolated system resources. By isolating a process from system resources, you can specify and control what a process is able to interact with. Namespaces are an essential part of Control Groups.

6.1.3. Supported namespaces

The following namespaces are supported from Red Hat Enterprise Linux 7.5 and later

- Mount
    - The mount namespace isolates file system mount points, enabling each process to have a distinct filesystem space within wich to operate.
- UTS
    - Hostname and NIS domain name
- IPC
    - System V IPC, POSIX message queues

- PID
  - Process IDs
- Network
  - Network devices, stacks, ports, etc.
- User
  - User and group IDs
- Control Groups
  - Isolates cgroups

NOTE

Usage of Control Groups is documented in the Resource Management Guide

6.2. KERNEL SOURCE CHECKER

The Linux Kernel Module Source Checker (ksc) is a tool to check for non whitelist symbols in a given kernel module. Red Hat Partners can also use the tool to request review of a symbol for whitelist inclusion, by filing a bug in Red Hat bugzilla database.

6.2.1. Usage

The tool accepts the path to a module with the "-k" option

```
# ksc -k e1000e.ko
Checking against architecture x86_64
Total symbol usage: 165 Total Non white list symbol usage: 74

# ksc -k /path/to/module
```

Output is saved in **$HOME/ksc-result.txt**. If review of the symbols for whitelist addition is requested, then the usage description for each non-whitelisted symbol must be added to the **ksc-result.txt** file. The request bug can then be filed by running **ksc** with the "-p" option.

NOTE

KSC currently does not support **xz** compression The **ksc** tool is unable to process the **xz** compression method and reports the following error:

```
Invalid architecture, (Only kernel object files are supported)
```

Until this limitation is resolved, system administrators need to manually uncompress any third party modules using **xz** compression, before running the **ksc** tool.

6.3. DIRECT ACCESS FOR FILES (DAX)

Direct Access for Files known as DAX, enables applications to read and write data on a storage device without using the page cache to buffer access to the file system.

This functionality is available on filesystems that implement the 'direct_access' block device operation and is enabled either by mounting the file system with '-o' or by adding 'dax' to the options section of **/etc/fstab**.

Further information, including code examples can be found in the **kernel-doc** package and is stored at **/usr/share/doc/kernel-doc-<version>/Documentation/filesystems/dax.txt** where '<version>' is the corresponding kernel version number.

6.4. MEMORY PROTECTION KEYS FOR USERSPACE (ALSO KNOWN AS PKU, OR PKEYS)

Memory Protection Keys provide a mechanism for enforcing page-based protections, but without requiring modification of the page tables when an application changes protection domains. It works by dedicating 4 previously ignored bits in each page table entry to a "protection key", giving 16 possible keys.

Memory Protection Keys are hardware feature of some Intel CPU chipsets. To determine if your processor supports this feature, check for the presence of **pku** in **/proc/cpuinfo**

```
$ grep pku /proc/cpuinfo
```

To support this feature, the CPUs provide a new user-accessible register (PKRU) with two separate bits (Access Disable and Write Disable) for each key. Two new instructions (RDPKRU and WRPKRU) exist for reading and writing to the new register.

Further documentation, including programming examples can be found in **/usr/share/doc/kernel-doc-*/Documentation/x86/protection-keys.txt** which is provided by the **kernel-doc** package.

6.5. KERNEL ADRESS SPACE LAYOUT RANDOMIZATION

Kernel Adress Space Layout Randomization (KASLR) consists of two parts which work together to enhance the security of the Linux kernel:

- kernel text KASLR
- memory management KASLR

The physical address and virtual address of kernel text itself are randomized to a different position separately. The physical address of the kernel can be anywhere under 64TB, while the virtual address of the kernel is restricted between [0xffffffff80000000, 0xffffffffc0000000], the 1GB space.

Memory management KASLR has three sections whose starting address is randomized in a specific area. KASLR can thus prevent inserting and redirecting the execution of the kernel to a malicious code if this code relies on knowing where symbols of interest are located in the kernel address space.

Memory management KASLR sections are:

- direct mapping section
- vmalloc section
- vmemmap section

KASLR code is now compiled into the Linux kernel, and it is enabled by default. To disable it explicitly, add the **`nokaslr`** kernel option to the kernel command line.

CHAPTER 7. LISTING OF KERNEL PARAMETERS AND VALUES

7.1. KERNEL COMMAND-LINE PARAMETERS

Kernel command-line parameters, also known as kernel arguments, are used to customize the behavior of Red Hat Enterprise Linux at boot time only.

7.1.1. Setting kernel command-line parameters

This section explains how to change a kernel command-line parameter on AMD64 and Intel 64 systems and IBM Power Systems servers using the **GRUB2** boot loader, and on IBM System z using **zipl**.

Kernel command-line parameters are saved in the **`boot/grub/grub.cfg`** configuration file, which is generated by the **GRUB2** boot loader. Do not edit this configuration file. Changes to this file are only made by configuration scripts.

Changing kernel command-line parameters in GRUB2 for AMD64 and Intel 64 systems and IBM Power Systems Hardware.

1. Open the **`/etc/default/grub`** configuration file as **`root`** using a plain text editor such as **vim** or **Gedit**.

2. In this file, locate the line beginning with **`GRUB_CMDLINE_LINUX`** similar to the following:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet"
```

3. Change the value of the required kernel command-line parameter. Then, save the file and exit the editor.

4. Regenerate the **GRUB2** configuration using the edited **`default`** file. If your system uses BIOS firmware, execute the following command:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

On a system with UEFI firmware, execute the following instead:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

After finishing the procedure above, the boot loader is reconfigured, and the kernel command-line parameter that you have specified in its configuration file is applied after the next reboot.

Changing kernel command-line parameters in zipl for IBM System z Hardware

1. Open the **`/etc/zipl.conf`** configuration file as **`root`** using a plain text editor such as **vim** or **Gedit**.

2. In this file, locate the **`parameters=`** section, and edit the requiremed parameter, or add it if not present. Then, save the file and exit the editor.

3. Regenerate the **zipl** configuration:

```
# zipl
```

NOTE

Executing only the `zipl` command with no additional options uses default values. See the `zipl(8)` man page for information about available options.

After finishing the procedure above, the boot loader is reconfigured, and the kernel command-line parameter that you have specified in its configuration file is applied after the next reboot.

7.1.2. What kernel command-line parameters can be controlled

For complete list of kernel command-line parameters, see https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt.

7.1.2.1. Hardware specific kernel command-line parameters

pci=option[,option…]

Specify behavior of the PCI hardware subsystem

Setting	Effect
earlydump	[X86] Dump the PCI configuration space before the kernel changes anything
off	[X86] Do not probe for the PCI bus
noaer	[PCIE] If the PCIEAER kernel parameter is enabled, this kernel boot option can be used to disable the use of PCIE advanced error reporting.
noacpi	[X86] Do not use the Advanced Configuration and Power Interface (ACPI) for Interrupt Request (IRQ) routing or for PCI scanning.
bfsort	Sort PCI devices into breadth-first order. This sorting is done to get a device order compatible with older (⇐ 2.4) kernels.
nobfsort	Do not sort PCI devices into breadth-first order.

Additional PCI options are documented in the on disk documentation found in the `kernel-doc-<version>.noarch` package. Where '<version>' needs to be replaced with the corresponding kernel version.

acpi=option

Specify behavior of the Advanced Configuration and Power Interface

Setting	Effect
acpi=off	Disable ACPI
acpi=ht	Use ACPI boot table parsing, but do not enable ACPI interpreter This disables any ACPI functionality that is **not** required for Hyper Threading.
acpi=force	Require the ACPI subsystem to be enabled
acpi=strict	Make the ACPI layer be less tolerant of platforms that are not fully compliant with the ACPI specification.
acpi_sci=<value>	Set up ACPI SCI interrupt, where <value> is one of edge,level,high,low.
acpi=noirq	Do not use ACPI for IRQ routing
acpi=nocmcff	Disable firmware first (FF) mode for corrected errors. This disables parsing the HEST CMC error source to check if firmware has set the FF flag. This can result in duplicate corrected error reports.

CHAPTER 8. REVISION HISTORY

0.1-4

Mon Mar 26 2018, Marie Doleželová (mdolezel@redhat.com)

- Document version for 7.5 GA publication.

0.1-2

Mon Jul 31 2017, Mark Flitter (mflitter@redhat.com)

- Document version for 7.4 GA publication.

0.1-0

Thu Apr 20 2017, Mark Flitter (mflitter@redhat.com)

- Initial build for review