Red Hat Enterprise Linux 9.0 Beta

Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 9
Red Hat Enterprise Linux 9.0 Beta Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 9
Abstract

This document provides the users and administrators with necessary information about configuring their workstations on the Linux kernel level. Such adjustments bring performance enhancements, easier troubleshooting or optimized system.
Table of Contents

RHEL BETA RELEASE ................................................................. 4
MAKING OPEN SOURCE MORE INCLUSIVE .................................. 5
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .................. 6

CHAPTER 1. THE LINUX KERNEL RPM ........................................ 7
1.1. WHAT AN RPM IS ......................................................... 7
    Types of RPM packages .............................................. 7
1.2. THE LINUX KERNEL RPM PACKAGE OVERVIEW .................... 7
1.3. DISPLAYING CONTENTS OF THE KERNEL PACKAGE ................. 8

CHAPTER 2. UPDATING KERNEL WITH YUM ................................ 10
2.1. WHAT IS THE KERNEL .................................................. 10
2.2. WHAT IS YUM .......................................................... 10
2.3. UPDATING THE KERNEL ............................................... 10
2.4. INSTALLING THE KERNEL ............................................. 10

CHAPTER 3. MANAGING KERNEL MODULES ............................... 12
3.1. INTRODUCTION TO KERNEL MODULES .............................. 12
3.2. KERNEL MODULE DEPENDENCIES ................................ 12
3.3. LISTING CURRENTLY LOADED KERNEL MODULES .................. 13
3.4. SETTING A KERNEL AS DEFAULT .................................. 13
3.5. DISPLAYING INFORMATION ABOUT KERNEL MODULES .......... 14
3.6. LOADING KERNEL MODULES AT SYSTEM RUNTIME ............... 15
3.7. UNLOADING KERNEL MODULES AT SYSTEM RUNTIME .......... 16
3.8. UNLOADING KERNEL MODULES AT EARLY STAGES OF THE BOOT PROCESS 17
3.9. LOADING KERNEL MODULES AUTOMATICALLY AT SYSTEM BOOT TIME 19
3.10. PREVENTING KERNEL MODULES FROM BEING AUTOMATICALLY LOADED AT SYSTEM BOOT TIME 19
3.11. COMPILING CUSTOM KERNEL MODULES ............................ 21

CHAPTER 4. CONFIGURING KERNEL COMMAND-LINE PARAMETERS ... 24
4.1. UNDERSTANDING KERNEL COMMAND-LINE PARAMETERS .......... 24
4.2. WHAT GRUBBY IS ...................................................... 24
4.3. WHAT BOOT ENTRIES ARE ........................................... 24
4.4. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR ALL BOOT ENTRIES 25
4.5. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR A SINGLE BOOT ENTRY 26
4.6. CHANGING KERNEL COMMAND-LINE PARAMETERS TEMPORARILY AT BOOT TIME 27

CHAPTER 5. CONFIGURING KERNEL PARAMETERS AT RUNTIME ........ 29
5.1. WHAT ARE KERNEL PARAMETERS ................................... 29
5.2. CONFIGURING KERNEL PARAMETERS TEMPORARILY WITH SYSCTL 30
5.3. CONFIGURING KERNEL PARAMETERS PERMANENTLY WITH SYSCTL 30
5.4. USING CONFIGURATION FILES IN /ETC/SYSCTL.D/ TO ADJUST KERNEL PARAMETERS 31
5.5. CONFIGURING KERNEL PARAMETERS TEMPORARILY THROUGH /PROC/SYS/ 32

CHAPTER 6. INSTALLING KDUMP ................................................. 33
6.1. WHAT IS KDUMP ......................................................... 33

CHAPTER 7. CONFIGURING KDUMP ON THE COMMAND LINE .......... 34
7.1. CONFIGURING KDUMP MEMORY USAGE ............................ 34
7.2. CONFIGURING THE KDUMP TARGET ............................... 36
7.3. CONFIGURING THE CORE COLLECTOR ............................ 38
7.4. CONFIGURING THE KDUMP DEFAULT FAILURE RESPONSES 40
7.5. CONFIGURATION FILE FOR KDUMP 40
7.6. ENABLING AND DISABLING THE KDUMP SERVICE 41
7.7. TESTING THE KDUMP CONFIGURATION 42
7.8. PREVENTING KERNEL DRIVERS FROM LOADING FOR KDUMP 43

CHAPTER 8. CONFIGURING KDUMP IN THE WEB CONSOLE 45
8.1. ADDITIONAL RESOURCES 45
8.2. CONFIGURING KDUMP MEMORY USAGE AND TARGET LOCATION IN WEB CONSOLE 45

CHAPTER 9. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS 48
9.1. MEMORY REQUIREMENTS FOR KDUMP 48
9.2. MINIMUM THRESHOLD FOR AUTOMATIC MEMORY RESERVATION 49
9.3. SUPPORTED KDUMP TARGETS 50
9.4. SUPPORTED KDUMP FILTERING LEVELS 51
9.5. SUPPORTED DEFAULT FAILURE RESPONSES 51
9.6. USING FINAL_ACTION PARAMETER 52

CHAPTER 10. ANALYZING A CORE DUMP 53
10.1. INSTALLING THE CRASH UTILITY 53
10.2. RUNNING AND EXITING THE CRASH UTILITY 53
10.3. DISPLAYING VARIOUS INDICATORS IN THE CRASH UTILITY 55
10.4. USING KERNEL OOPS ANALYZER 57
10.5. THE KDUMP HELPER TOOL 58

CHAPTER 11. APPLYING PATCHES WITH KERNEL LIVE PATCHING 59
11.1. LIMITATIONS OF KPATCH 59
11.2. SUPPORT FOR THIRD-PARTY LIVE PATCHING 59
11.3. ACCESS TO KERNEL LIVE PATCHES 60
11.4. COMPONENTS OF KERNEL LIVE PATCHING 60
11.5. HOW KERNEL LIVE PATCHING WORKS 60
11.6. SUBSCRIBING THE CURRENTLY INSTALLED KERNELS TO THE LIVE PATCHING STREAM 61
11.7. AUTOMATICALLY SUBSCRIBING ANY FUTURE KERNEL TO THE LIVE PATCHING STREAM 62
11.8. UPDATING KERNEL PATCH MODULES 64
11.9. REMOVING THE LIVE PATCHING PACKAGE 65
11.10. UNINSTALLING THE KERNEL PATCH MODULE 66
11.11. DISABLING KPATCH.SERVICE 67
RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.

- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.

Red Hat Enterprise Linux 9.0 Beta Managing, monitoring, and updating the kernel
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the Multi-page HTML format. In addition, ensure you see the Feedback button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the Add Feedback pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use Documentation.
  3. Fill in the Description field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click Submit Bug.
CHAPTER 1. THE LINUX KERNEL RPM

The following sections describe the Linux kernel RPM package provided and maintained by Red Hat.

1.1. WHAT AN RPM IS

An RPM package is a file containing other files and their metadata (information about the files that are needed by the system).

Specifically, an RPM package consists of the `cpio` archive.

The `cpio` archive contains:

- Files
- RPM header (package metadata)
  The `rpm` package manager uses this metadata to determine dependencies, where to install files, and other information.

Types of RPM packages
There are two types of RPM packages. Both types share the file format and tooling, but have different contents and serve different purposes:

- Source RPM (SRPM)
  An SRPM contains source code and a SPEC file, which describes how to build the source code into a binary RPM. Optionally, the patches to source code are included as well.

- Binary RPM
  A binary RPM contains the binaries built from the sources and patches.

1.2. THE LINUX KERNEL RPM PACKAGE OVERVIEW

The `kernel` RPM is a meta package that does not contain any files, but rather ensures that the following required sub-packages are properly installed:

- `kernel-core` - contains the binary image of the kernel, all initramfs-related objects to bootstrap the system, and a minimal number of kernel modules to ensure core functionality. This sub-package alone could be used in virtualized and cloud environments to provide a Red Hat Enterprise Linux 8 kernel with a quick boot time and a small disk size footprint.

- `kernel-modules` - contains the remaining kernel modules that are not present in `kernel-core`.

The small set of `kernel` sub-packages above aims to provide a reduced maintenance surface to system administrators especially in virtualized and cloud environments.

Optional kernel packages are for example:

- `kernel-modules-extra` - contains kernel modules for rare hardware and modules which loading is disabled by default.

- `kernel-debug` – contains a kernel with numerous debugging options enabled for kernel diagnosis, at the expense of reduced performance.

- `kernel-tools` – contains tools for manipulating the Linux kernel and supporting documentation.
● **kernel-devel** – contains the kernel headers and makefiles sufficient to build modules against the kernel package.

● **kernel-abi-whitelists** – contains information pertaining to the RHEL kernel ABI, including a list of kernel symbols that are needed by external Linux kernel modules and a `yum` plugin to aid enforcement.

● **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.

### Additional resources

- What are the kernel-core, kernel-modules, and kernel-modules-extras packages?

## 1.3. DISPLAYING CONTENTS OF THE KERNEL PACKAGE

The following procedure describes how to view the contents of the kernel package and its sub-packages without installing them using the `rpm` command.

### Prerequisites

- Obtained kernel, kernel-core, kernel-modules, kernel-modules-extra RPM packages for your CPU architecture

### Procedure

- List modules for kernel:

  ```
  $ rpm -qlp <kernel_rpm>
  (contains no files)
  ...
  ```

- List modules for kernel-core:

  ```
  $ rpm -qlp <kernel-core_rpm>
  ...
  /lib/modules/5.14.0-1.el9.x86_64/kernel/fs/udf/udf.ko.xz
  /lib/modules/5.14.0-1.el9.x86_64/kernel/fs/xfs
  /lib/modules/5.14.0-1.el9.x86_64/kernel/fs/xfs/xfs.ko.xz
  /lib/modules/5.14.0-1.el9.x86_64/kernel/kernel
  /lib/modules/5.14.0-1.el9.x86_64/kernel/kernel/trace
  /lib/modules/5.14.0-1.el9.x86_64/kernel/kernel/trace/ring_buffer_benchmark.ko.xz
  /lib/modules/5.14.0-1.el9.x86_64/kernel/lib
  /lib/modules/5.14.0-1.el9.x86_64/kernel/lib/cordic.ko.xz
  ...
  ```

- List modules for kernel-modules:

  ```
  $ rpm -qlp <kernel-modules_rpm>
  ...
  /lib/modules/5.14.0-1.el9.x86_64/kernel/drivers/infiniband/hw/mlx4/mlx4_ib.ko.xz
  /lib/modules/5.14.0-1.el9.x86_64/kernel/drivers/infiniband/hw/mlx5/mlx5_ib.ko.xz
  /lib/modules/5.14.0-1.el9.x86_64/kernel/drivers/infiniband/hw/qedr/qedr.ko.xz
  ```
List modules for `kernel-modules-extra`:

```
$ rpm -qlp kernel-modules-extra_rpm
```

Additional resources

- `rpm(8)` manual page
- `RPM packages`
CHAPTER 2. UPDATING KERNEL WITH YUM

The following sections bring information about the Linux kernel provided and maintained by Red Hat (Red Hat kernel), and how to keep the Red Hat kernel updated. As a consequence, the operating system will have all the latest bug fixes, performance enhancements, and patches ensuring compatibility with new hardware.

2.1. WHAT IS THE KERNEL

The kernel is a core part of a Linux operating system, which manages the system resources, and provides interface between hardware and software applications. The Red Hat kernel is a custom-built kernel based on the upstream Linux mainline kernel that Red Hat engineers further develop and harden with a focus on stability and compatibility with the latest technologies and hardware.

Before Red Hat releases a new kernel version, the kernel needs to pass a set of rigorous quality assurance tests.

The Red Hat kernels are packaged in the RPM format so that they are easy to upgrade and verify by the yum package manager.

WARNING

Kernels that have not been compiled by Red Hat are not supported by Red Hat.

2.2. WHAT IS YUM

This section refers to description of the yum package manager.

Additional resources

- Configuring basic system settings in RHEL

2.3. UPDATING THE KERNEL

The following procedure describes how to update the kernel using the yum package manager.

Procedure

1. To update the kernel, use the following:

   # yum update kernel

   This command updates the kernel along with all dependencies to the latest available version.

2. Reboot your system for the changes to take effect.

2.4. INSTALLING THE KERNEL
The following procedure describes how to install new kernels using the `yum` package manager.

**Procedure**

- To install a specific kernel version, use the following:

  ```bash
  # yum install kernel-{version}
  ```

**Additional resources**

- [Red Hat Code Browser](#)
- [Red Hat Enterprise Linux Release Dates](#)
CHAPTER 3. MANAGING KERNEL MODULES

The following sections explain what kernel modules are, how to display their information, and how to perform basic administrative tasks with kernel modules.

3.1. INTRODUCTION TO KERNEL MODULES

The Red Hat Enterprise Linux kernel can be extended with optional, additional pieces of functionality, called kernel modules, without having to reboot the system. On Red Hat Enterprise Linux 8, kernel modules are extra kernel code which is built into compressed `<KERNEL_MODULE_NAME>.ko.xz` object files.

The most common functionality enabled by kernel modules are:

- Device driver which adds support for new hardware
- Support for a file system such as GFS2 or NFS
- System calls

On modern systems, kernel modules are automatically loaded when needed. However, in some cases it is necessary to load or unload modules manually.

Like the kernel itself, the modules can take parameters that customize their behavior if needed.

Tooling is provided to inspect which modules are currently running, which modules are available to load into the kernel and which parameters a module accepts. The tooling also provides a mechanism to load and unload kernel modules into the running kernel.

3.2. KERNEL MODULE DEPENDENCIES

Certain kernel modules sometimes depend on one or more other kernel modules. The `/lib/modules/<KERNEL_VERSION>/modules.dep` file contains a complete list of kernel module dependencies for the respective kernel version.

The dependency file is generated by the `depmod` program, which is a part of the `kmod` package. Many of the utilities provided by `kmod` take module dependencies into account when performing operations so that manual dependency-tracking is rarely necessary.

**WARNING**

The code of kernel modules is executed in kernel-space in the unrestricted mode. Because of this, you should be mindful of what modules you are loading.

Additional resources

- `modules.dep(5)` manual page
- `depmod(8)` manual page
3.3. LISTING CURRENTLY LOADED KERNEL MODULES

The following procedure describes how to view the currently loaded kernel modules.

Prerequisites

- The kmod package is installed.

Procedure

- To list all currently loaded kernel modules, execute:

```
$ lsmod
```

<table>
<thead>
<tr>
<th>Module</th>
<th>Size</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuse</td>
<td>126976</td>
<td>3</td>
</tr>
<tr>
<td>uinput</td>
<td>20480</td>
<td>1</td>
</tr>
<tr>
<td>xt_CHECKSUM</td>
<td>16384</td>
<td>1</td>
</tr>
<tr>
<td>ipt_MASQUERADE</td>
<td>16384</td>
<td>1</td>
</tr>
<tr>
<td>xt_conntrack</td>
<td>16384</td>
<td>1</td>
</tr>
<tr>
<td>ipt_REJECT</td>
<td>16384</td>
<td>1</td>
</tr>
<tr>
<td>nft_counter</td>
<td>16384</td>
<td>16</td>
</tr>
<tr>
<td>nf_nat_tftp</td>
<td>16384</td>
<td>0</td>
</tr>
<tr>
<td>nf_conntrack_tftp</td>
<td>16384</td>
<td>1 nf_nat_tftp</td>
</tr>
<tr>
<td>tun</td>
<td>49152</td>
<td>1</td>
</tr>
<tr>
<td>bridge</td>
<td>192512</td>
<td>0</td>
</tr>
<tr>
<td>stp</td>
<td>16384</td>
<td>1 bridge</td>
</tr>
<tr>
<td>llc</td>
<td>16384</td>
<td>2 bridge,stp</td>
</tr>
<tr>
<td>nf_tables_set</td>
<td>32768</td>
<td>5</td>
</tr>
<tr>
<td>nft_fib_inet</td>
<td>16384</td>
<td>1</td>
</tr>
</tbody>
</table>

In the example above:

- The first column provides the names of currently loaded modules.
- The second column displays the amount of memory per module in kilobytes.
- The last column shows the number, and optionally the names of modules that are dependent on a particular module.

Additional resources

- /usr/share/doc/kmod/README file
- lsmod(8) manual page

3.4. SETTING A KERNEL AS DEFAULT

The following procedure describes how to set a specific kernel as default using the grubby command-line tool and GRUB2.

Procedure

Setting the kernel as default, using the grubby tool
Execute the following command to set the kernel as default using the `grubby` tool:

```
# grubby --set-default $kernel_path
```

The command uses a machine ID without the `.conf` suffix as an argument.

**NOTE**

The machine ID is located in the `/boot/loader/entries/` directory.

Setting the kernel as default, using the `id` argument

- List the boot entries using the `id` argument and then set an intended kernel as default:

  ```
  # grubby --info ALL | grep id
  # grubby --set-default /boot/vmlinuz-<version>.<architecture>
  ```

  **NOTE**

  To list the boot entries using the `title` argument, execute the `# grubby --info=ALL | grep title` command.

Setting the default kernel for only the next boot

- Execute the following command to set the default kernel for only the next reboot using the `grub2-reboot` command:

  ```
  # grub2-reboot <index|title|id>
  ```

**WARNING**

Set the default kernel for only the next boot with care. Installing new kernel RPM’s, self-built kernels, and manually adding the entries to the `/boot/loader/entries/` directory may change the index values.

### 3.5. DISPLAYING INFORMATION ABOUT KERNEL MODULES

When working with a kernel module, you may want to see further information about that module. This procedure describes how to display extra information about kernel modules.

**Prerequisites**

- The `kmod` package is installed.

**Procedure**
To display information about any kernel module, execute:

```bash
$ modinfo <KERNEL_MODULE_NAME>
```

For example:

```bash
$ modinfo virtio_net
```

filename:       /lib/modules/5.14.0-1.el9.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:        GPL
description:    Virtio network driver
rhelversion:    9.0
srcversion:     8809CDDDBE7202A1B00B9F1C
alias:          virtio:d00000001v*
depends:        net_failover
retpoline:      Y
intree:         Y
name:           virtio_net
vermagic:       5.14.0-1.el9.x86_64 SMP mod_unload modversions
...  
parm:           napi_weight:int
parm:           csum:bool
parm:           gso:bool
parm:           napi_tx:bool

+ The `modinfo` command displays some detailed information about the specified kernel module. You can query information about all available modules, regardless of whether they are loaded or not. The `parm` entries show parameters the user is able to set for the module, and what type of value they expect.

+ 

**NOTE**

When entering the name of a kernel module, do not append the `.ko.xz` extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

Additional resources

- `modinfo(8)` manual page

**3.6. LOADING KERNEL MODULES AT SYSTEM RUNTIME**

The optimal way to expand the functionality of the Linux kernel is by loading kernel modules. The following procedure describes how to use the `modprobe` command to find and load a kernel module into the currently running kernel.

**Prerequisites**

- Root permissions
- The `kmod` package is installed.
- The respective kernel module is not loaded. To ensure this is the case, list the loaded kernel modules.
**Procedure**

1. Select a kernel module you want to load. The modules are located in the `/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/` directory.

2. Load the relevant kernel module:

   ```
   # modprobe <MODULE_NAME>
   ```

   **NOTE**
   
   When entering the name of a kernel module, do not append the `.ko.xz` extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Optionally, verify the relevant module was loaded:

   ```
   $ lsmod | grep <MODULE_NAME>
   ```

   If the module was loaded correctly, this command displays the relevant kernel module. For example:

   ```
   $ lsmod | grep serio_raw
   serio_raw             16384  0
   ```

   **IMPORTANT**
   
   The changes described in this procedure will not persist after rebooting the system. For information on how to load kernel modules to persist across system reboots, see [Loading kernel modules automatically at system boot time](#).

**Additional resources**

- [modprobe(8)](modprobe(8)) manual page

### 3.7. UNLOADING KERNEL MODULES AT SYSTEM RUNTIME

At times, you find that you need to unload certain kernel modules from the running kernel. The following procedure describes how to use the `modprobe` command to find and unload a kernel module at system runtime from the currently loaded kernel.

**Prerequisites**

- Root permissions
- The `kmod` package is installed.

**Procedure**

1. Execute the `lsmod` command and select a kernel module you want to unload.
   If a kernel module has dependencies, unload those prior to unloading the kernel module. For details on identifying modules with dependencies, see [Listing currently loaded kernel modules](#) and [Kernel module dependencies](#).
2. Unload the relevant kernel module:

```
# modprobe -r <MODULE_NAME>
```

When entering the name of a kernel module, do not append the .ko.xz extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

**WARNING**

Do not unload kernel modules when they are used by the running system. Doing so can lead to an unstable or non-operational system.

3. Optionally, verify the relevant module was unloaded:

```
$ lsmod | grep <MODULE_NAME>
```

If the module was unloaded successfully, this command does not display any output.

**IMPORTANT**

After finishing this procedure, the kernel modules that are defined to be automatically loaded on boot, will not stay unloaded after rebooting the system. For information on how to counter this outcome, see Preventing kernel modules from being automatically loaded at system boot time.

Additional resources

- [modprobe(8)] manual page

### 3.8. UNLOADING KERNEL MODULES AT EARLY STAGES OF THE BOOT PROCESS

In certain situations it is necessary to unload a kernel module very early in the booting process. For example, when the kernel module contains a code, which causes the system to become unresponsive, and the user is not able to reach the stage to permanently disable the rogue kernel module. In that case it is possible to temporarily block the loading of the kernel module using a bootloader.

**IMPORTANT**

The changes described in this procedure will not persist after the next reboot. For information on how to add a kernel module to a denylist so that it will not be automatically loaded during the boot process, see Preventing kernel modules from being automatically loaded at system boot time.

**Prerequisites**

- You have a loadable kernel module, which you want to prevent from loading for some reason.
Procedure

- Edit the relevant bootloader entry to unload the desired kernel module before the booting sequence continues.
  
  - Use the cursor keys to highlight the relevant bootloader entry.
  
  - Press e key to edit the entry.

Figure 3.1. Kernel boot menu

Use the cursor keys to navigate to the line that starts with `linux`.

Append `modprobe.blacklist=module_name` to the end of the line.

Figure 3.2. Kernel boot entry

The `serio_raw` kernel module illustrates a rogue module to be unloaded early in the boot process.

- Press CTRL+x keys to boot using the modified configuration.

Verification

- Once the system fully boots, verify that the relevant kernel module is not loaded.
# Ismod | grep serio_raw

Additional resources
- Managing kernel modules

3.9. LOADING KERNEL MODULES AUTOMATICALLY AT SYSTEM BOOT TIME

The following procedure describes how to configure a kernel module so that it is loaded automatically during the boot process.

Prerequisites
- Root permissions
- The kmod package is installed.

Procedure
1. Select a kernel module you want to load during the boot process. The modules are located in the /lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/ directory.

2. Create a configuration file for the module:

   # echo <MODULE_NAME> > /etc/modules-load.d/<MODULE_NAME>.conf

   **NOTE**

   When entering the name of a kernel module, do not append the .ko.xz extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Optionally, after reboot, verify the relevant module was loaded:

   $ lsmod | grep <MODULE_NAME>

   The example command above should succeed and display the relevant kernel module.

   **IMPORTANT**

   The changes described in this procedure will persist after rebooting the system.

Additional resources
- modules-load.d(5) manual page

3.10. PREVENTING KERNEL MODULES FROM BEING AUTOMATICALLY LOADED AT SYSTEM BOOT TIME
The following procedure describes how to add a kernel module to a denylist so that it will not be automatically loaded during the boot process.

Prerequisites

- Root permissions
- The kmod package is installed.
- Ensure that a kernel module in a denylist is not vital for your current system configuration.

Procedure

1. Select a kernel module that you want to put in a denylist:

   ```
   $ lsmod
   Module                  Size  Used by
   fuse                  126976  3
   xt_CHECKSUM            16384  1
   ipt_MASQUERADE         16384  1
   uinput                 20480  1
   xt_conntrack           16384  1
   ...
   ```

   The `lsmod` command displays a list of modules loaded to the currently running kernel.

   - Alternatively, identify an unloaded kernel module you want to prevent from potentially loading.
   - All kernel modules are located in the `/lib/modules/<KERNEL_VERSION>/kernel/<SUBSYSTEM>/` directory.

2. Create a configuration file for a denylist:

   ```
   # vim /etc/modprobe.d/blacklist.conf

   # Blacklists <KERNEL_MODULE_1>
   blacklist <MODULE_NAME_1>
   install <MODULE_NAME_1> /bin/false

   # Blacklists <KERNEL_MODULE_2>
   blacklist <MODULE_NAME_2>
   install <MODULE_NAME_2> /bin/false

   # Blacklists <KERNEL_MODULE_n>
   blacklist <MODULE_NAME_n>
   install <MODULE_NAME_n> /bin/false
   ...
   ```

   The example shows the contents of the `blacklist.conf` file, edited by the `vim` editor. The `blacklist` line ensures that the relevant kernel module will not be automatically loaded during the boot process. The `blacklist` command, however, does not prevent the module from being loaded as a dependency for another kernel module that is not in a denylist. Therefore the `install` line causes the `/bin/false` to run instead of installing a module.

   The lines starting with a hash sign are comments to make the file more readable.
NOTE

When entering the name of a kernel module, do not append the `.ko.xz` extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Create a backup copy of the current initial ramdisk image before rebuilding:

```bash
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
```

The command above creates a backup initramfs image in case the new version has an unexpected problem.

- Alternatively, create a backup copy of other initial ramdisk image which corresponds to the kernel version for which you want to put kernel modules in a denylist:

```bash
# cp /boot/initramfs-<SOME_VERSION>.img /boot/initramfs-<SOME_VERSION>.img.bak.$(date +%m-%d-%H%M%S)
```

4. Generate a new initial ramdisk image to reflect the changes:

```bash
# dracut -f -v
```

- If you are building an initial ramdisk image for a different kernel version than you are currently booted into, specify both target initramfs and kernel version:

```bash
# dracut -f -v /boot/initramfs-<TARGET_VERSION>.img <CORRESPONDING_TARGET_KERNEL_VERSION>
```

5. Reboot the system:

```bash
$ reboot
```

IMPORTANT

The changes described in this procedure will take effect and persist after rebooting the system. If you improperly put a key kernel module in a denylist, you can face an unstable or non-operational system.

Additional resources

- How do I prevent a kernel module from loading automatically?
- dracut(8) manual page

3.11. COMPILING CUSTOM KERNEL MODULES

You can build a sampling kernel module as requested by various configurations at hardware and software level.

Prerequisites
You installed the `kernel-devel`, `gcc`, and `elfutils-libelf-devel` packages.

You have root permissions.

You created the `/root/testmodule/` directory where you compile the custom kernel module.

### Procedure

1. Create the `/root/testmodule/test.c` file with the following content:

   ```c
   #include <linux/module.h>
   
   int init_module(void)
   {
       printk("Hello World\n This is a test\n");
       return 0;
   }
   
   void cleanup_module(void)
   {
       printk("Good Bye World");
   }
   ``

   The `test.c` file is a source file that provides the main functionality to the kernel module. The file has been created in a dedicated `/root/testmodule/` directory for organizational purposes. After the module compilation, the `/root/testmodule/` directory will contain multiple files.

   The `test.c` file includes from the system libraries:

   - The `linux/kernel.h` header file is necessary for the `printk()` function in the example code.
   - The `linux/module.h` file contains function declarations and macro definitions to be shared between several source files written in C programming language.

   Next follow the `init_module()` and `cleanup_module()` functions to start and end the kernel logging function `printk()`, which prints text.

2. Create the `/root/testmodule/Makefile` file with the following content:

   ```
   obj-m := test.o
   ``

   The Makefile contains instructions that the compiler has to produce an object file specifically named `test.o`. The `obj-m` directive specifies that the resulting `test.ko` file is going to be compiled as a loadable kernel module. Alternatively, the `obj-y` directive would instruct to build `test.ko` as a built-in kernel module.

3. Compile the kernel module:

   ```bash
   # make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
   make: Entering directory '/usr/src/kernels/5.14.0-1.el9.x86_64'
   CC [M] /root/testmodule/test.o
   Building modules, stage 2.
   MODPOST 1 modules
   WARNING: modpost: missing MODULE_LICENSE() in /root/testmodule/test.o
   see include/linux/module.h for more information
   CC /root/testmodule/test.mod.o
   LD [M] /root/testmodule/test.ko
   make: Leaving directory '/usr/src/kernels/5.14.0-1.el9.x86_64'
   ```
The compiler creates an object file (test.o) for each source file (test.c) as an intermediate step before linking them together into the final kernel module (test.ko).

After a successful compilation, /root/testmodule/ contains additional files that relate to the compiled custom kernel module. The compiled module itself is represented by the test.ko file.

Verification

1. Optional: check the contents of the /root/testmodule/ directory:

   ```
   ls -l /root/testmodule/
   total 452
   -rw-r--r-- 1 root root  16 Jul 12 10:16 Makefile
   -rw-r--r-- 1 root root  32 Jul 12 10:16 modules.order
   -rw-r--r-- 1 root root  0 Jul 12 10:16 Module.symvers
   -rw-r--r-- 1 root root 197 Jul 12 10:15 test.c
   -rw-r--r-- 1 root root 219736 Jul 12 10:16 test.ko
   -rw-r--r-- 1 root root  826 Jul 12 10:16 test.mod.c
   -rw-r--r-- 1 root root 113760 Jul 12 10:16 test.mod.o
   -rw-r--r-- 1 root root 107424 Jul 12 10:16 test.o
   ```

2. Copy the kernel module to the /lib/modules/$(uname -r)/ directory:

   ```
   # cp /root/testmodule/test.ko /lib/modules/$(uname -r)/
   ```

3. Update the modular dependency list:

   ```
   # depmod -a
   ```

4. Load the kernel module:

   ```
   # modprobe -v test
   insmod /lib/modules/5.14.0-1.el9.x86_64/test.ko
   ```

   1. Verify that the kernel module was successfully loaded:

      ```
      # lsmod | grep test
      test 16384 0
      ```

   2. Read the latest messages from the kernel ring buffer:

      ```
      # dmesg
      ...
      [74422.545004] Hello World
      This is a test
      ```

Additional resources

- Managing kernel modules
CHAPTER 4. CONFIGURING KERNEL COMMAND-LINE PARAMETERS

Kernel command-line parameters are a way to change the behavior of certain aspects of the Red Hat Enterprise Linux kernel at boot time. As a system administrator, you have full control over what options get set at boot. Certain kernel behaviors are only able to be set at boot time, so understanding how to make these changes is a key administration skill.

**IMPORTANT**

Opting to change the behavior of the system by modifying kernel command-line parameters may have negative effects on your system. You should therefore test changes prior to deploying them in production. For further guidance, contact Red Hat Support.

4.1. UNDERSTANDING KERNEL COMMAND-LINE PARAMETERS

Kernel command-line parameters are used for boot time configuration of:

- The Red Hat Enterprise Linux kernel
- The initial RAM disk
- The user space features

Kernel boot time parameters are often used to overwrite default values and for setting specific hardware settings.

By default, the kernel command-line parameters for systems using the GRUB2 bootloader are defined in the `kernelopts` variable of the `/boot/grub2/grubenv` file for all kernel boot entries.

**NOTE**

For IBM Z, the kernel command-line parameters are stored in the boot entry configuration file because the zipl bootloader does not support environment variables. Thus, the `kernelopts` environment variable cannot be used.

Additional resources

- `kernel-command-line(7)`, `bootparam(7)` and `dracut.cmdline(7)` manual pages
- How to install and boot custom kernels in Red Hat Enterprise Linux 8

4.2. WHAT GRUBBY IS

`grubby` is a utility for manipulating bootloader-specific configuration files.

You can use `grubby` also for changing the default boot entry, and for adding/removing arguments from a GRUB2 menu entry.

For more details see the `grubby(8)` manual page.

4.3. WHAT BOOT ENTRIES ARE
A boot entry is a collection of options which are stored in a configuration file and tied to a particular kernel version. In practice, you have at least as many boot entries as your system has installed kernels. The boot entry configuration file is located in the `/boot/loader/entries/` directory and can look like this:

```
d8712ab6d4f14683c5625e87b52b6b6e-5.14.0-1.el9.x86_64.conf
```

The file name above consists of a machine ID stored in the `/etc/machine-id` file, and a kernel version.

The boot entry configuration file contains information about the kernel version, the initial ramdisk image, and the `kernelopts` environment variable, which contains the kernel command-line parameters. The contents of a boot entry config can be seen below:

```
title Red Hat Enterprise Linux (5.14.0-1.el9.x86_64) 9.0 (Plow)
version 5.14.0-1.el9.x86_64
linux /vmlinuz-5.14.0-1.el9.x86_64
initrd /initramfs-5.14.0-1.el9.x86_64.img
options root=/dev/mapper/rhel_kvm--02--guest08-root ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=/dev/mapper/rhel_kvm--02--guest08-swap rd.lvm.lv=rhel_kvm-02-guest08/root rd.lvm.lv=rhel_kvm-02-guest08/swap console=ttyS0,115200 grub_users $grub_users grub_arg --unrestricted grub_class kernel
```

The `kernelopts` environment variable is defined in the `/boot/grub2/grubenv` file.

Additional resources

- How to install and boot custom kernels in Red Hat Enterprise Linux 8

### 4.4. Changing Kernel Command-Line Parameters for All Boot Entries

This procedure describes how to change kernel command-line parameters for all boot entries on your system.

**Prerequisites**

- Verify that the `grubby` and `zipl` utilities are installed on your system.

**Procedure**

- To add a parameter:

  ```
  # grubby --update-kernel=ALL --args="<NEW_PARAMETER>"
  ```

  For systems that use the GRUB2 bootloader, the command updates the `/boot/grub2/grubenv` file by adding a new kernel parameter to the `kernelopts` variable in that file.

  On IBM Z that use the zIPL bootloader, the command adds a new kernel parameter to each `/boot/loader/entries/<ENTRY>.conf` file.

  - On IBM Z, execute the `zipl` command with no options to update the boot menu.

- To remove a parameter:
# grubby --update-kernel=ALL --remove-args="<PARAMETER_TO_REMOVE>"
- On IBM Z, execute the `zipl` command with no options to update the boot menu.

Additional resources
- Understanding kernel command-line parameters
- `grubby(8)` and `zipl(8)` manual pages
- `grubby tool`

4.5. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR A SINGLE BOOT ENTRY

This procedure describes how to change kernel command-line parameters for a single boot entry on your system.

Prerequisites
- Verify that the `grubby` and `zipl` utilities are installed on your system.

Procedure
- To add a parameter:
  
  ```
  # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="<NEW_PARAMETER>"
  ```
  - On IBM Z, execute the `zipl` command with no options to update the boot menu.

- To remove a parameter use the following:
  
  ```
  # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --remove-args="<PARAMETER_TO_REMOVE>"
  ```
  - On IBM Z, execute the `zipl` command with no options to update the boot menu.

**NOTE**

On systems that use the `grub.cfg` file, there is, by default, the `options` parameter for each kernel boot entry, which is set to the `kernelopts` variable. This variable is defined in the `/boot/grub2/grubenv` configuration file.
IMPORTANT

On GRUB2 systems:

- If the kernel command-line parameters are modified for all boot entries, the `grubby` utility updates the `kernelopts` variable in the `/boot/grub2/grubenv` file.
- If kernel command-line parameters are modified for a single boot entry, the `kernelopts` variable is expanded, the kernel parameters are modified, and the resulting value is stored in the respective boot entry's `/boot/loader/entries/<RELEVANTKERNEL_BOOT_ENTRY>.conf` file.

On zIPL systems:

- `grubby` modifies and stores the kernel command-line parameters of an individual kernel boot entry in the `/boot/loader/entries/<ENTRY>.conf` file.

Additional resources

- Understanding kernel command-line parameters
- `grubby(8)` and `zipl(8)` manual pages
- `grubby tool`

4.6. CHANGING KERNEL COMMAND-LINE PARAMETERS TEMPORARILY AT BOOT TIME

The following procedure allows you to make temporary changes to a Kernel Menu Entry by changing the kernel parameters only during a single boot process.

Procedure

1. Select the kernel you want to start when the GRUB 2 boot menu appears and press the `e` key to edit the kernel parameters.

2. Find the kernel command line by moving the cursor down. The kernel command line starts with `linux` on 64-Bit IBM Power Series and x86-64 BIOS-based systems, or `linuxefi` on UEFI systems.

3. Move the cursor to the end of the line.

   **NOTE**

   Press `Ctrl+a` to jump to the start of the line and `Ctrl+e` to jump to the end of the line. On some systems, `Home` and `End` keys might also work.

4. Edit the kernel parameters as required. For example, to run the system in emergency mode, add the `emergency` parameter at the end of the `linux16` line:

   ```
   linux16 /vmlinuz-3.10.0-0.rc4.59.el7.x86_64 root=/devmapper/rhel-root ro rd.md=0 rd.dm=0 rd.lvm.lv=rhel/swap crashkernel=auto rd.luks=0 vconsole.keymap=us rd.lvm.lv=rhel/root rhgb quiet emergency
   ```
To enable the system messages, remove the `rhgb` and `quiet` parameters.

5. Press **Ctrl+x** to boot with the selected kernel and the modified command line parameters.

**IMPORTANT**

Press **Esc** key to leave command line editing and it will drop all the user made changes.

**NOTE**

This procedure applies only for a single boot and does not persistently make the changes.
CHAPTER 5. CONFIGURING KERNEL PARAMETERS AT RUNTIME

As a system administrator, you can modify many facets of the Red Hat Enterprise Linux kernel’s behavior at runtime. This section describes how to configure kernel parameters at runtime by using the `sysctl` command and by modifying the configuration files in the `/etc/sysctl.d/` and `/proc/sys/` directories.

5.1. WHAT ARE KERNEL PARAMETERS

Kernel parameters are tunable values which you can adjust while the system is running. There is no requirement to reboot or recompile the kernel for changes to take effect.

It is possible to address the kernel parameters through:

- The `sysctl` command
- The virtual file system mounted at the `/proc/sys/` directory
- The configuration files in the `/etc/sysctl.d/` directory

Tunables are divided into classes by the kernel subsystem. Red Hat Enterprise Linux has the following tunable classes:

Table 5.1. Table of `sysctl` classes

<table>
<thead>
<tr>
<th>Tunable class</th>
<th>Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>abi</td>
<td>Execution domains and personalities</td>
</tr>
<tr>
<td>crypto</td>
<td>Cryptographic interfaces</td>
</tr>
<tr>
<td>debug</td>
<td>Kernel debugging interfaces</td>
</tr>
<tr>
<td>dev</td>
<td>Device-specific information</td>
</tr>
<tr>
<td>fs</td>
<td>Global and specific file system tunables</td>
</tr>
<tr>
<td>kernel</td>
<td>Global kernel tunables</td>
</tr>
<tr>
<td>net</td>
<td>Network tunables</td>
</tr>
<tr>
<td>sunrpc</td>
<td>Sun Remote Procedure Call (NFS)</td>
</tr>
<tr>
<td>user</td>
<td>User Namespace limits</td>
</tr>
<tr>
<td>vm</td>
<td>Tuning and management of memory, buffers, and cache</td>
</tr>
</tbody>
</table>
IMPORTANT

Configuring kernel parameters on a production system requires careful planning. Unplanned changes may render the kernel unstable, requiring a system reboot. Verify that you are using valid options before changing any kernel values.

Additional resources

- `sysctl(8)`, and `sysctl.d(5)` manual pages

5.2. CONFIGURING KERNEL PARAMETERS TEMPORARILY WITH SYSCTL

The following procedure describes how to use the `sysctl` command to temporarily set kernel parameters at runtime. The command is also useful for listing and filtering tunables.

Prerequisites

- Root permissions

Procedure

1. To list all parameters and their values, use the following:

   ```
   # sysctl -a
   ```

   **NOTE**

   The `# sysctl -a` command displays kernel parameters, which can be adjusted at runtime and at boot time.

2. To configure a parameter temporarily, use the command as in the following example:

   ```
   # sysctl <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
   ```

   The sample command above changes the parameter value while the system is running. The changes take effect immediately, without a need for restart.

   **NOTE**

   The changes return back to default after your system reboots.

Additional resources

- `sysctl(8)` manual page
- Configuring kernel parameters permanently with `sysctl`
- Using configuration files in `/etc/sysctl.d/` to adjust kernel parameters

5.3. CONFIGURING KERNEL PARAMETERS PERMANENTLY WITH SYSCTL
The following procedure describes how to use the `sysctl` command to permanently set kernel parameters.

Prerequisites

- Root permissions

Procedure

1. To list all parameters, use the following:

   ```sh
   # sysctl -a
   ```

   The command displays all kernel parameters that can be configured at runtime.

2. To configure a parameter permanently:

   ```sh
   # sysctl -w <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE> >> /etc/sysctl.conf
   ```

   The sample command changes the tunable value and writes it to the `/etc/sysctl.conf` file, which overrides the default values of kernel parameters. The changes take effect immediately and persistently, without a need for restart.

   **NOTE**

   To permanently modify kernel parameters you can also make manual changes to the configuration files in the `/etc/sysctl.d/` directory.

Additional resources

- `sysctl(8)` and `sysctl.conf(5)` manual pages
- Using configuration files in `/etc/sysctl.d/` to adjust kernel parameters

5.4. USING CONFIGURATION FILES IN `/ETC/SYSCTL.D/` TO ADJUST KERNEL PARAMETERS

The following procedure describes how to manually modify configuration files in the `/etc/sysctl.d/` directory to permanently set kernel parameters.

Prerequisites

- Root permissions

Procedure

1. Create a new configuration file in `/etc/sysctl.d/`:

   ```sh
   # vim /etc/sysctl.d/<some_file.conf>
   ```

2. Include kernel parameters, one per line, as follows:
3. Save the configuration file.

4. Reboot the machine for the changes to take effect.
   - Alternatively, to apply changes without rebooting, execute:
     ```bash
     # sysctl -p /etc/sysctl.d/<some_file.conf>
     ```
     The command enables you to read values from the configuration file, which you created earlier.

Additional resources
- `sysctl(8), sysctl.d(5)` manual pages

5.5. CONFIGURING KERNEL PARAMETERS TEMPORARILY THROUGH /PROC/SYS/

The following procedure describes how to set kernel parameters temporarily through the files in the virtual file system `/proc/sys/` directory.

Prerequisites
- Root permissions

Procedure
1. Identify a kernel parameter you want to configure:
   ```bash
   # ls -l /proc/sys/<TUNABLE_CLASS>/
   ```
   The writable files returned by the command can be used to configure the kernel. The files with read-only permissions provide feedback on the current settings.

2. Assign a target value to the kernel parameter:
   ```bash
   # echo <TARGET_VALUE> > /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
   ```
   The command makes configuration changes that will disappear once the system is restarted.

3. Optionally, verify the value of the newly set kernel parameter:
   ```bash
   # cat /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
   ```

Additional resources
- Configuring kernel parameters permanently with sysctl
- Using configuration files in `/etc/sysctl.d/` to adjust kernel parameters
CHAPTER 6. INSTALLING KDUMP

In many cases, the **kdump** service is installed and activated by default on the new Red Hat Enterprise Linux installations. This section includes information about **kdump**.

6.1. WHAT IS KDUMP

**kdump** is a service providing a crash dumping mechanism. The service enables you to save the contents of the system’s memory for later analysis. **kdump** uses the **kexec** system call to boot into the second kernel (a **capture kernel**) without rebooting; and then captures the contents of the crashed kernel’s memory (a **crash dump** or a **vmcore**) and saves it. The second kernel resides in a reserved part of the system memory.

**IMPORTANT**

A kernel crash dump can be the only information available in the event of a system failure (a critical bug). Therefore, ensuring that **kdump** is operational is important in mission-critical environments. 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.
CHAPTER 7. CONFIGURING KDUMP ON THE COMMAND LINE

The memory for `kdump` is reserved during the system boot. The memory size is configured in the system’s Grand Unified Bootloader (GRUB) 2 configuration file. The memory size depends on the `crashkernel=` value specified in the configuration file and the size of the system’s physical memory.

7.1. CONFIGURING KDUMP MEMORY USAGE

This procedure describes how to configure memory usage for `kdump`.

The `crashkernel` default mechanism contains the default `crashkernel` memory reservation values for the corresponding kernel build. `kdump` uses the default value to reserve the `crashkernel` memory for each kernel. You can also use the default value as the reference base value to estimate a required memory size when setting the `crashkernel=` value manually. The minimum size of the crash kernel might vary depending on the hardware and machine specifications.

The automatic memory allocation for `kdump` varies based on the system hardware architecture and available memory size. For example, on AMD64 and Intel 64, the `crashkernel` default parameters work only when the available memory is more than 1GB. Hence, by default, on AMD64 and Intel 64, the `crashkernel` mechanism configures the following memory reserve:

```
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
```

**NOTE**

The `crashkernel` default mechanism is supported on the `kdump` tool and does not require manual editing. It is recommended to not modify the default `crashkernel` parameters manually.

Red Hat recommends testing the `kdump` after you configure the `crashkernel=` value and then configure an appropriate memory size to set an acceptable value.

The old `crashkernel=auto` option in the boot command line is no longer supported on RHEL 9 and later releases.

Prerequisites

- Fulfilled `kdump` requirements for configurations and targets

Procedure

1. To use the `crashkernel` default parameters, run the following command:

   ```
   kdumpctl reset-crashkernel
   ```

2. (Optional) To configure parameters other than the default, fetch an estimate of the `crashkernel=` value using the `kdumpctl` command and then set it to the kernel parameters:

   a. To estimate the `crashkernel=` value, run the `kdumpctl` command:

   ```
   kdumpctl estimate
   ```

   The `kdumpctl` sub-command allows you to fetch a rough estimate value without triggering a `kdump`. 
The command prints a summary report, which states the estimated memory consumption of each *kdump* component.

For example:

```
Encrypted kdump target requires extra memory, assuming using the keyslot with minimum memory requirement

Reserved crashkernel: 256M
Recommended crashkernel: 655M

Kernel image size: 47M
Kernel modules size: 12M
Initramfs size: 19M
Runtime reservation: 64M
LUKS required size: 512M
Large modules:
  xfs: 1892352
  nouveau: 2318336
WARNING: Current crashkernel size is lower than recommended size 655M.
```

Using the *kdumpctl* sub-command you can fetch a rough estimate value without triggering a *kdump*.

The estimated *crashkernel* value may not be accurate and can serve as a reference to set an appropriate *crashkernel* value.

**b. Set the *crashkernel* option to the required value.**

For example, to reserve 128 MB of memory, configure as follows:

```
crashkernel=128M
```

Alternatively, you can set the amount of reserved memory to a variable depending on the total amount of installed memory. The syntax for memory reservation into a variable is `crashkernel=<range1>:<size1>,<range2>:<size2>`. Consider the following illustration:

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

The command 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 the memory reserve for *kdump*.

- Offset the reserved memory.
  
  Some systems need you to reserve memory with a specific fixed offset because the *crashkernel* reservation is very early, and might require some reservation for special usage. If the offset is set, the reserved memory begins there. To offset the reserved memory, use the following syntax:

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

The command 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`).
3. Run the following command to update the GRUB2 configuration file:

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

4. Reboot the system for changes to take effect.

**NOTE**

The alternative way to configure memory for kdump is to append the `crashkernel=<SOME_VALUE>` parameter to the `kernelopts` variable with the `grub2-editenv`. Doing so, updates all of your boot entries. You can also use the `grubby` utility to update kernel command line parameters of just one entry.

Additional resources

- Memory requirements for kdump
- Configuring kernel command line parameters
- How to manually modify the boot parameter in grub before the system boots
- How to install and boot custom kernels in Red Hat Enterprise Linux 8

### 7.2. CONFIGURING THE KDUMP TARGET

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 a time, and the default behavior is to store the `vmcore` file in the `/var/crash/` directory of the local file system.

**Prerequisites**

- Fulfilled kdump requirements for configurations and targets.

**Procedure**

- To store the `vmcore` file in `/var/crash/` directory of the local file system, edit the `/etc/kdump.conf` file and specify the path:

  ```
  path /var/crash
  ```

  The option `path /var/crash` represents the path to the file system in which kdump saves the `vmcore` file. When you specify a dump target in the `/etc/kdump.conf` file, then the `path` is relative to the specified dump target.

  If you do not specify a dump target in the `/etc/kdump.conf` file, then the `path` represents the absolute path from the root directory. Depending on what is mounted in the current system, the dump target and the adjusted dump path are taken automatically.
WARNING

kdump saves the vmcore file in /var/crash/var/crash directory, when the dump target is mounted at /var/crash and the option path is also set as /var/crash in the /etc/kdump.conf file. For example, in the following instance, the ext4 file system is already mounted at /var/crash and the path are set as /var/crash:

```
grep -v ^# etc/kdump.conf | grep -v ^$
ext4 /dev/mapper/vg00-varcrashvol
path /var/crash
core_collector makedumpfile -c --message-level 1 -d 31
```

This results in the /var/crash/var/crash path. To solve this problem, use the option path / instead of path /var/crash

- To change the local directory in which the core dump is to be saved, as root, edit the /etc/kdump.conf configuration file as described below.
  1. Remove the hash sign ("#") from the beginning of the #path /var/crash line.
  2. Replace the value with the intended directory path. For example:

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

**IMPORTANT**

In RHEL 9, 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 RHEL, where the directory was being created automatically if it did not exist when starting the service.

- To write the file to a different partition, as root, edit the /etc/kdump.conf configuration file as described below.
  1. Remove the hash sign ("#") from the beginning of the #ext4 line, depending on your choice.
     - device name (the #ext4 /dev/vg/lv_kdump line)
     - file system label (the #ext4 LABEL=/boot line)
     - UUID (the #ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937 line)
  2. 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
```
It is recommended to specify storage devices using a **LABEL**= or **UUID**=. Disk device names such as `/dev/sda3` are not guaranteed to be consistent across reboot.

When dumping to Direct Access Storage Device (DASD) on IBM Z 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:**
  1. Remove the hash sign ("#") from the beginning of the `#raw /dev/vg/lv_kdump` line.
  2. Replace the value with the intended device name. For example:

     ```
     raw /dev/sdb1
     ```

- **To store the dump to a remote machine using the NFS protocol:**
  1. Remove the hash sign ("#") from the beginning of the `#nfs my.server.com:/export/tmp` line.
  2. 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:**
  1. Remove the hash sign ("#") from the beginning of the `#ssh user@my.server.com` line.
  2. Replace the value with a valid username and hostname.
  3. Include your **SSH** key in the configuration.
     - Remove the hash sign from the beginning of the `#sshkey /root/.ssh/kdump_id_rsa` line.
     - 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
       ```

### 7.3. Configuring the Core Collector

The **kdump** service uses a **core_collector** program to capture the **vmcore** image. In RHEL, the **makedumpfile** utility is the default core collector.

**makedumpfile** is a dump program that helps to compress the size of a dump file and copy only necessary pages using various dump levels.
**makedumpfile** makes a small dump file either by compressing dump data or by excluding unnecessary pages or both. It needs the first kernel debug information to understand how first kernel uses the memory. This helps to detect the necessary pages.

**Syntax**

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

**Options**

- `-c`, `-l` or `-p`: specify compress dump file format by each page using either, `zlib` for `-c` option, `lzo` for `-l` option or `snappy` for `-p` option.
- `-d` (**dump_level**): excludes pages so that they are not copied to the dump file.
- `--message-level`: specify the message types. You can restrict outputs printed by specifying `message_level` with this option. For example, specifying 7 as `message_level` prints common messages and error messages. The maximum value of `message_level` is 31

**Prerequisites**

- Fulfilled `kdump` requirements for configurations and targets.

**Procedure**

1. As root, edit the `/etc/kdump.conf` configuration file and remove the hash sign (“#”) from the beginning of the `#core_collector makedumpfile -l --message-level 1 -d 31`.
2. To enable dump file compression, execute:

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

where,

- `-l` specifies the dump compressed file format.
- `-d` specifies dump level as 31.
- `--message-level` specifies message level as 1.

Also, consider following examples using options `-c` and `-p`:

- To compress dump file using `-c`:

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

- To compress dump file using `-p`:

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

**Additional resources**

- [makedumpfile(8) manual page](#)
7.4. CONFIGURING THE KDUMP DEFAULT FAILURE RESPONSES

By default, when `kdump` fails to create a vmcore file at the configured target location, the system reboots and the dump is lost in the process. To change this behavior, follow the procedure below.

Prerequisites

- Fulfilled `kdump` requirements for configurations and targets.

Procedure

1. As `root`, remove the hash sign ("#") from the beginning of the `#failure_action` line in the `/etc/kdump.conf` configuration file.

2. Replace the value with a desired action.

   ```
   failure_action poweroff
   ```

Additional resources

- Configuring the `kdump` target

7.5. CONFIGURATION FILE FOR KDUMP

The configuration file for `kdump` kernel is `/etc/sysconfig/kdump`. This file controls the `kdump` kernel command line parameters.

For most configurations, use the default options. However, in some scenarios you might need to modify certain parameters to control the `kdump` kernel behavior. For example, modifying to append the `kdump` kernel command-line to obtain a detailed debugging output.

This section covers information on modifying the `KDUMP_COMMANDLINE_REMOVE` and `KDUMP_COMMANDLINE_APPEND` options for `kdump`. For information on additional configuration options refer to `Documentation/admin-guide/kernel-parameters.txt` or the `/etc/sysconfig/kdump` file.

- **KDUMP_COMMANDLINE_REMOVE**
  
  This option removes arguments from the current `kdump` command line. It removes parameters that may cause `kdump` errors or `kdump` kernel boot failures. These parameters may have been parsed from the previous `KDUMP_COMMANDLINE` process or inherited from the `/proc/cmdline` file. When this variable is not configured, it inherits all values from the `/proc/cmdline` file. Configuring this option also provides information that is helpful in debugging an issue.

  **Example**

  To remove certain arguments, add them to `KDUMP_COMMANDLINE_REMOVE` as follows:

  ```
  KDUMP_COMMANDLINE_REMOVE="hugepages hugepagesizez slub_debug quiet log_buf_len swiotlb"
  ```

- **KDUMP_COMMANDLINE_APPEND**
  
  This option appends arguments to the current command line. These arguments may have been parsed by the previous `KDUMP_COMMANDLINE_REMOVE` variable.
For the `kdump` kernel, disabling certain modules such as `mce`, `cgroup`, `numa`, `hest_disable` can help prevent kernel errors. These modules may consume a significant portion of the kernel memory reserved for `kdump` or cause `kdump` kernel boot failures.

**Example**

To disable memory `cgroups` on the `kdump` kernel command line, run the command as follows:

```
KDUMP_COMMANDLINE_APPEND="cgroup_disable=memory"
```

**Additional resources**

- Documentation/admin-guide/kernel-parameters.txt file
- `/etc/sysconfig/kdump` file

### 7.6. ENABLING AND DISABLING THE KDUMP SERVICE

To start the `kdump` service at boot time, follow the procedure below.

**Prerequisites**

- Fulfilled `kdump` requirements for configurations and targets.
- All configurations for installing `kdump` are set up according to your needs.

**Procedure**

1. To enable the `kdump` service, use the following command:

   ```
   # systemctl enable kdump.service
   ```

   This enables the service for `multi-user.target`.

2. To start the service in the current session, use the following command:

   ```
   # systemctl start kdump.service
   ```

3. To stop the `kdump` service, type the following command:

   ```
   # systemctl stop kdump.service
   ```

4. To disable the `kdump` service, execute the following command:

   ```
   # systemctl disable kdump.service
   ```
WARNING

It is recommended to set `kptr_restrict=1` as default. When `kptr_restrict` is set to (1) as default, the `kdumpctl` service loads the crash kernel even if Kernel Address Space Layout (KASLR) is enabled or not enabled.

Troubleshooting step

When `kptr_restrict` is not set to (1), and if KASLR is enabled, the contents of `/proc/kore` file are generated as all zeros. Consequently, the `kdumpctl` service fails to access the `/proc/kcore` and load the crash kernel.

To work around this problem, the `kexec-kdump-howto.txt` file displays a warning message, which specifies to keep the recommended setting as `kptr_restrict=1`.

To ensure that `kdumpctl` service loads the crash kernel, verify that:

- Kernel `kptr_restrict=1` in the `sysctl.conf` file.

Additional resources

- **Configuring basic system settings** in RHEL

7.7. TESTING THE KDUMP CONFIGURATION

The following procedure describes how to test that the kernel dump process works and is valid before the machine enters production.

WARNING

The commands below cause the kernel to crash. Use caution when following these steps, and never carelessly use them on active production system.

Procedure

1. Reboot the system with `kdump` enabled.

2. Make sure that `kdump` is running:

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

3. Force the Linux kernel to crash:

   ```bash
   echo 1 > /proc/sys/kernel/sysrq
   echo c > /proc/sysrq-trigger
   ```
WARNING
The command above crashes the kernel and a reboot is required.

Once booted again, the `address-YYYY-MM-DD-HH:MM:SS/vmcore` file is created at the location you have specified in `/etc/kdump.conf` (by default to `/var/crash/`).

NOTE
In addition to confirming the validity of the configuration, it is possible to use this action to record how long it takes for a crash dump to complete, while a representative load was running.

Additional resources

- Configuring the kdump target

7.8. PREVENTING KERNEL DRIVERS FROM LOADING FOR KDUMP

This section explains how to prevent the kdump kernel from loading certain kernel drivers using the `/etc/sysconfig/kdump` configuration file. Appending the `KDUMP_COMMANDLINE_APPEND=` variable in `/etc/sysconfig/kdump` file, prevents the kdump initramfs from loading the specified kernel module. This helps to prevent oom killer or other crash kernel failures.

You can append the `KDUMP_COMMANDLINE_APPEND=` variable using one of the following configuration options:

- `rd.driver.blacklist=<modules>`
- `modprobe.blacklist=<modules>`

Procedure

1. Select a kernel module that you intend to block from loading.

   $ lsmod
   Module                  Size  Used by
   fuse                  126976  3
   xt_CHECKSUM            16384  1
   ipt_MASQUERADE         16384  1
   uinput                 20480  1
   xt_conntrack           16384  1

   The `lsmod` command displays a list of modules that are loaded to the currently running kernel.

2. Update the `KDUMP_COMMANDLINE_APPEND=` variable in the `/etc/sysconfig/kdump` file.
KDUMP_COMMANDLINE_APPEND="rd.driver.blacklist=hv_vmbus,hv_storvsc,hv_utils,hv_netvsc,hid-hyperv"

Also, consider the following example using the `modprobe.blacklist=<modules>` configuration option.

KDUMP_COMMANDLINE_APPEND="modprobe.blacklist=emcp modprobe.blacklist=bnx2fc modprobe.blacklist=libfcoe modprobe.blacklist=fcoe"

3. Restart the `kdump` service.

   $ systemctl restart kdump

Additional resources

- `dracut.cmdline` manual page
CHAPTER 8. CONFIGURING KDUMP IN THE WEB CONSOLE

Setup and test the *kdump* configuration in the RHEL 9 web console.

The web console is part of a default installation of RHEL 9 and enables or disables the *kdump* service at boot time. Further, the web console conveniently enables you to configure the reserved memory for *kdump*; or to select the *vmcore* saving location in an uncompressed or compressed format.

### 8.1. ADDITIONAL RESOURCES

- Getting started using the RHEL web console

### 8.2. CONFIGURING KDUMP MEMORY USAGE AND TARGET LOCATION IN WEB CONSOLE

The procedure below shows you how to use the *Kernel Dump* tab in the RHEL web console interface to configure the amount of memory that is reserved for the *kdump* kernel. The procedure also describes how to specify the target location of the *vmcore* dump file and how to test your configuration.

**Procedure**

1. Open the *Kernel Dump* tab and start the *kdump* service.

2. Configure the *kdump* memory usage using the command line.

3. Click the link next to the *Crash dump location* option.

4. Select the *Local Filesystem* option from the drop-down and specify the directory you want to save the dump in.
Alternatively, select the **Remote over SSH** option from the drop-down to send the vmcore to a remote machine using the SSH protocol. Fill the **Server**, **ssh key**, and **Directory** fields with the remote machine address, ssh key location, and a target directory.

Another choice is to select the **Remote over NFS** option from the drop-down and fill the **Mount** field to send the vmcore to a remote machine using the NFS protocol.

**NOTE**

Tick the **Compression** check box to reduce the size of the vmcore file.

5. Test your configuration by crashing the kernel.
Additional resources

- Supported kdump targets
- Using secure communications between two systems with OpenSSH
CHAPTER 9. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS

9.1. 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. When reserved, this part of the system memory is not available to the 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 machine architecture (such as Intel 64 and AMD64, also known as x86_64) and print it to standard output, use the following command:

```
$ uname -m
```

Table 9.1 lists the minimum memory requirements to automatically reserve a memory size for `kdump` on the latest available versions. The size changes according to the system’s architecture and total available physical memory.

**Table 9.1. Minimum Amount of Reserved Memory Required for kdump**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Available Memory</th>
<th>Minimum Reserved Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD64 and Intel 64 (x86_64)</td>
<td>1 GB to 4 GB</td>
<td>160 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>4 GB to 64 GB</td>
<td>192 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>64 GB to 1 TB</td>
<td>256 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>1 TB and more</td>
<td>512 MB of RAM.</td>
</tr>
<tr>
<td>64-bit ARM architecture (arm64)</td>
<td>2 GB and more</td>
<td>448 MB of RAM.</td>
</tr>
<tr>
<td>IBM Power Systems (ppc64le)</td>
<td>2 GB to 4 GB</td>
<td>384 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>4 GB to 16 GB</td>
<td>512 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>16 GB to 64 GB</td>
<td>1 GB of RAM.</td>
</tr>
<tr>
<td></td>
<td>64 GB to 128 GB</td>
<td>2 GB of RAM.</td>
</tr>
<tr>
<td></td>
<td>128 GB and more</td>
<td>4 GB of RAM.</td>
</tr>
<tr>
<td>IBM Z (s390x)</td>
<td>1 GB to 4 GB</td>
<td>160 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>4 GB to 64 GB</td>
<td>192 MB of RAM.</td>
</tr>
<tr>
<td></td>
<td>64 GB to 1 TB</td>
<td>256 MB of RAM.</td>
</tr>
</tbody>
</table>
On many systems, `kdump` is able to 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.

### IMPORTANT

The automatic configuration of reserved memory based on the total amount of memory in the system is a best effort estimation. The actual required memory may vary due to other factors such as I/O devices. Using not enough of memory might cause that a debug kernel is not able to boot as a capture kernel in case of a kernel panic. To avoid this problem, sufficiently increase the crash kernel memory.

Additional resources

- How has the crashkernel parameter changed between RHEL8 minor releases?
- Technology capabilities and limits tables
- Configuring kdump memory usage
- Configuring kdump memory usage and target location in web console
- Minimum threshold for automatic memory reservation

## 9.2. 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 boot loader 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. The amount differs based on the system’s architecture.

Table 10.2 lists the threshold values for automatic memory allocation. If the system has memory less than the specified threshold value, you must configure the memory manually.

### Table 9.2. Minimum Amount of Memory Required for Automatic Memory Reservation

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Required Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD64 and Intel 64 (x86_64)</td>
<td>2 GB</td>
</tr>
<tr>
<td>IBM Power Systems (ppc64le)</td>
<td>2 GB</td>
</tr>
<tr>
<td>IBM Z (s390x)</td>
<td>4 GB</td>
</tr>
</tbody>
</table>

Additional resources

- Configuring kdump memory usage
9.3. SUPPORTED KDUMP TARGETS

When a kernel crash is captured, the vmcore dump file 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*.

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

**IMPORTANT**

Utilizing firmware assisted dump (*fadump*) to capture a vmcore and store it to a remote machine using SSH or NFS protocol causes renaming of the network interface to *kdump-*<interface-name>*. The renaming happens if the *<interface-name>* is generic, for example *eth*, *net*, and so on. This problem occurs because the vmcore capture scripts in the initial RAM disk (*initrd*) add the *kdump-* prefix to the network interface name to secure persistent naming. Since the same *initrd* is used also for a regular boot, the interface name is changed for the production kernel too.
Additional resources
- Configuring kdump target
- Configuring kdump memory usage and target location in web console

9.4. SUPPORTED KDUMP FILTERING LEVELS

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Zero pages</td>
</tr>
<tr>
<td>2</td>
<td>Cache pages</td>
</tr>
<tr>
<td>4</td>
<td>Cache private</td>
</tr>
<tr>
<td>8</td>
<td>User pages</td>
</tr>
<tr>
<td>16</td>
<td>Free pages</td>
</tr>
</tbody>
</table>

**NOTE**

The makedumpfile command supports removal of transparent huge pages and hugetlbfs pages. Consider both these types of hugepages User Pages and remove them using the -8 level.

Additional resources
- Configuring the core collector

9.5. SUPPORTED DEFAULT FAILURE RESPONSES

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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump_to_rootfs</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reboot</td>
<td>Reboot the system, losing the core dump in the process.</td>
</tr>
<tr>
<td>halt</td>
<td>Halt the system, losing the core dump in the process.</td>
</tr>
<tr>
<td>poweroff</td>
<td>Power off the system, losing the core dump in the process.</td>
</tr>
<tr>
<td>shell</td>
<td>Run a shell session from within the initramfs, allowing the user to record</td>
</tr>
<tr>
<td></td>
<td>the core dump manually.</td>
</tr>
<tr>
<td>final_action</td>
<td>Enable additional operations such as <code>reboot</code>, <code>halt</code>, and <code>poweroff</code> actions</td>
</tr>
<tr>
<td></td>
<td>after a successful <code>kdump</code> or when <code>shell</code> or <code>dump_to_rootfs</code> failure action</td>
</tr>
</tbody>
</table>
CHAPTER 10. 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 core dump created by kdump, netdump, diskdump or xendump as well as a running Linux system. Alternatively, you have the option to use Kernel Oops Analyzer or the Kdump Helper tool.

10.1. INSTALLING THE CRASH UTILITY

The following procedure describes how to install the crash analyzing tool.

Procedure

1. Enable the relevant repositories:
   
   ```
   # subscription-manager repos --enable baseos repository
   # subscription-manager repos --enable appstream repository
   # subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
   ```

2. Install the crash package:
   
   ```
   # yum install crash
   ```

3. Install the kernel-debuginfo package:
   
   ```
   # yum install kernel-debuginfo
   ```

   The package corresponds to your running kernel and provides the data necessary for the dump analysis.

Additional resources

- Configuring basic system settings in RHEL

10.2. RUNNING AND EXITING THE CRASH UTILITY

The following procedure describes how to start the crash utility for analyzing the cause of the system crash.

Prerequisites

- Identify the currently running kernel (for example 5.14.0-1.el9.x86_64).

Procedure

1. To start the crash utility, two necessary parameters need to be passed to the command:
   
   - The debug-info (a decompressed vmlinuz image), for example /usr/lib/debug/lib/modules/5.14.0-1.el9.x86_64/vmlinux provided through a specific kernel-debuginfo package.
The actual vmcore file, for example /var/crash/127.0.0.1-2021-09-13-14:05:33/vmcore

The resulting crash command then looks like this:

```
# crash /usr/lib/debug/lib/modules/5.14.0-1.el9.x86_64/vmlinux /var/crash/127.0.0.1-2021-09-13-14:05:33/vmcore
```

Use the same <kernel> version that was captured by kdump.

Example 10.1. Running the crash utility

The following example shows analyzing a core dump created on September 13 2021 at 14:05 PM, using the 5.14.0-1.el9.x86_64 kernel.

```
... WARNING: kernel relocated [202MB]: patching 90160 gdb minimal_symbol values

KERNEL: /usr/lib/debug/lib/modules/5.14.0-1.el9.x86_64/vmlinux
DUMPFILE: /var/crash/127.0.0.1-2021-09-13-14:05:33/vmcore  [PARTIAL DUMP]
CPUS: 2
DATE: Mon Sep 13 14:05:16 2021
UPTIME: 01:03:57
LOAD AVERAGE: 0.00, 0.00, 0.00
TASKS: 586
NODENAME: localhost.localdomain
RELEASE: 5.14.0-1.el9.x86_64
VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
MACHINE: x86_64 (2904 Mhz)
MEMORY: 2.9 GB
PANIC: "sysrq: SysRq : Trigger a crash"
PID: 10635
COMMAND: "bash"
TASK: ffff8d6c84271800  [THREAD_INFO: ffff8d6c84271800]
CPU: 1
STATE: TASK_RUNNING (SYSRQ)
```

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

Example 10.2. Exiting the crash utility

```
crash> exit
```

NOTE

The crash command can also be used as a powerful tool for debugging a live system. However use it with caution so as not to break your system.

Additional resources

- A Guide to Unexpected System Restarts
10.3. DISPLAYING VARIOUS INDICATORS IN THE CRASH UTILITY

The following procedures describe how to use the crash utility and display various indicators, such as a kernel message buffer, a backtrace, a process status, virtual memory information and open files.

Displaying the message buffer

- To display the kernel message buffer, type the `log` command at the interactive prompt as displayed in the example below:

```
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> ffffffffib c068150f b7776000 f260c40 c0569ec4 ef4dbf9c 00000002 b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560 ef4dfb4
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+0xa0/0xa0
[<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:ef4dfb24
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 into 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.

Displaying a backtrace

- To display the kernel stack trace, use the `bt` command.

```
crash> bt
PID: 5591   TASK: f196d560 CPU: 2   COMMAND: "bash"
#0 [ef4dbdcc] crash_kexec at c0494922
#1 [ef4dbe20] oops_end at c080e402
```
Displaying a process status

To display the status of processes in the system, use the `ps` command.

```
crash> ps
     PID   PPID  CPU TASK ST %MEM VSZ RSS COMM
>     0     0   0 c09dc560 RU 0.0   0  0  [swapper]
>     0     1  f7072030 RU 0.0   0  0  [swapper]
  0     2  f70a3a90 RU 0.0   0  0  [swapper]
>     0     3  f70ac560 RU 0.0   0  0  [swapper]
     1     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
```

Use `ps <pid>` to display the status of a single specific process. Use `help ps` for more information on `ps` usage.

Displaying virtual memory information

To display basic virtual memory information, type the `vm` command at the interactive prompt.

```
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
```

Use `vm <pid>` to display information on a single specific process, or use `help vm` for more information on `vm` usage.

Displaying open files

- To display information about open files, use the `files` command.

```
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
```

Use `files <pid>` to display files opened by only one selected process, or use `help files` for more information on `files` usage.

### 10.4. USING KERNEL OOPS ANALYZER

The Kernel Oops Analyzer tool analyzes the crash dump by comparing the oops messages with known issues in the knowledge base.

**Prerequisites**

- Secure an oops message to feed the Kernel Oops Analyzer.

**Procedure**

1. Access the Kernel Oops Analyzer tool.

2. To diagnose a kernel crash issue, upload a kernel oops log generated in `vmcore`. 
• Alternatively you can also diagnose a kernel crash issue by providing a text message or a `vmcore-dmesg.txt` as an input.

![Detect button and file input option]

3. Click **Detect** to compare the oops message based on information from the `makedumpfile` against known solutions.

Additional resources

- Red Hat Customer Portal Labs
- KernelOops Analyzer
- A Guide to Unexpected System Restarts

### 10.5. THE KDUMP HELPER TOOL

The Kdump Helper tool helps to set up the `kdump` using the provided information. Kdump Helper generates a configuration script based on your preferences. Initiating and running the script on your server sets up the `kdump` service.

Additional resources

- Kdump Helper
You can use the Red Hat Enterprise Linux kernel live patching solution to patch a running kernel without rebooting or restarting any processes.

With this solution, system administrators:

- Can immediately apply critical security patches to the kernel.
- Do not have to wait for long-running tasks to complete, for users to log off, or for scheduled downtime.
- Control the system’s uptime more and do not sacrifice security or stability.

Note that not every critical or important CVE will be resolved using the kernel live patching solution. Our goal is to reduce the required reboots for security-related patches, not to eliminate them entirely. For more details about the scope of live patching, see the Customer Portal Solutions article.

**WARNING**

Some incompatibilities exist between kernel live patching and other kernel subcomponents. Read the Limitations of kpatch carefully before using kernel live patching.

### 11.1. LIMITATIONS OF KPATCH

- The kpatch feature 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 such probes have been removed.

### 11.2. SUPPORT FOR THIRD-PARTY LIVE PATCHING

The kpatch utility is the only kernel live patching utility supported by Red Hat with the RPM modules provided by Red Hat repositories. Red Hat will not support any live patches which were not provided by Red Hat itself.

If you require support for an issue that arises with a third-party live patch, Red Hat recommends that you open a case with the live 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 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?

11.3. ACCESS TO KERNEL LIVE PATCHES

Kernel live patching capability is implemented as a kernel module (\texttt{kmod}) that is delivered as an RPM package.

All customers have access to kernel live patches, which are delivered through the usual channels. However, customers who do not subscribe to an extended support offering will lose access to new patches for the current minor release once the next minor release becomes available. For example, customers with standard subscriptions will only be able to live patch RHEL 9.1 kernel until the RHEL 9.2 kernel is released.

11.4. COMPONENTS OF KERNEL LIVE PATCHING

The components of kernel live patching are as follows:

\textbf{Kernel patch module}

- The delivery mechanism for kernel live patches.
- A kernel module which is built specifically for the kernel being patched.
- The patch module contains the code of the desired fixes for the kernel.
- The patch modules register with the \texttt{livepatch} kernel subsystem and provide information about original functions to be replaced, with corresponding pointers to the replacement functions. Kernel patch modules are delivered as RPMs.
- The naming convention is \texttt{kpatch\_<kernel version>\_<kpatch version>\_<kpatch release>}. The "kernel version" part of the name has \texttt{dots} replaced with \texttt{underscores}.

\textbf{The kpatch utility}

A command-line utility for managing patch modules.

\textbf{The kpatch service}

A \texttt{systemd} service required by \texttt{multiuser.target}. This target loads the kernel patch module at boot time.

\textbf{The kpatch-dnf package}

A DNF plugin delivered in the form of an RPM package. This plugin manages automatic subscription to kernel live patches.

11.5. HOW KERNEL LIVE PATCHING WORKS

The \texttt{kpatch} kernel patching solution uses the \texttt{livepatch} kernel subsystem to redirect old functions to new ones. When a live kernel patch is applied to a system, the following things happen:

1. The kernel patch module is copied to the \texttt{/var/lib/kpatch/} directory and registered for re-application to the kernel by \texttt{systemd} on next boot.

2. The kpatch module is loaded into the running kernel and the new functions are registered to the \texttt{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 by the *ftrace* mechanism which bypasses the original functions and redirects the kernel to patched version of the function.

Figure 11.1. How kernel live patching works

11.6. SUBSCRIBING THE CURRENTLY INSTALLED KERNELS TO THE LIVE PATCHING STREAM

A kernel patch module is delivered in an RPM package, specific to the version of the kernel being patched. Each RPM package will be cumulatively updated over time.

The following procedure explains how to subscribe to all future cumulative live patching updates for a given kernel. Because live patches are cumulative, you cannot select which individual patches are deployed for a given kernel.

**WARNING**

Red Hat does not support any third party live patches applied to a Red Hat supported system.

**Prerequisites**

- Root permissions

**Procedure**

1. Optionally, check your kernel version:

```
# uname -r
5.14.0-1.el9.x86_64
```
2. Search for a live patching package that corresponds to the version of your kernel:

```bash
# yum search $(uname -r)
```

3. Install the live patching package:

```bash
# yum install "kpatch-patch = $(uname -r)"
```

The command above installs and applies the latest cumulative live patches for that specific kernel only.

If the version of a live patching package is 1-1 or higher, the package will contain a patch module. In that case the kernel will be automatically patched during the installation of the live patching package.

The kernel patch module is also installed into the `/var/lib/kpatch/` directory to be loaded by the `systemd` system and service manager during the future reboots.

**NOTE**

An empty live patching package will be installed when there are no live patches available for a given kernel. An empty live patching package will have a `kpatch_version-kpatch_release` of 0-0, for example `kpatch-patch-5_14_0-1-0-0.x86_64.rpm`. The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

4. Optionally, verify that the kernel is patched:

```bash
# kpatch list
Loaded patch modules:
  kpatch_5_14_0_1_0_1 [enabled]

Installed patch modules:
  kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
...
```

The output shows that the kernel patch module has been loaded into the kernel, which is now patched with the latest fixes from the `kpatch-patch-5_14_0-1-0-1.el9.x86_64.rpm` package.

**Additional resources**

- `kpatch(1)` manual page
- `Configuring basic system settings` in RHEL

**11.7. AUTOMATICALLY SUBSCRIBING ANY FUTURE KERNEL TO THE LIVE PATCHING STREAM**

You can use the `kpatch-dnf` DNF plugin to subscribe your system to fixes delivered by the kernel patch module, also known as kernel live patches. The plugin enables automatic subscription for any kernel the system currently uses, and also for kernels **to-be-installed in the future**

**Prerequisites**
• Root permissions.

Procedure

1. Optionally, check all installed kernels and the kernel you are currently running:

   ```
   # yum list installed | grep kernel
   Updating Subscription Management repositories.
   Installed Packages
   ...
   kernel-core.x86_64  5.14.0-1.el9  @beaker-BaseOS
   kernel-core.x86_64  5.14.0-2.el9  @@commandline
   ...
   # uname -r
   5.14.0-2.el9.x86_64
   ```

2. Install the `kpatch-dnf` plugin:

   ```
   # yum install kpatch-dnf
   ```

3. Enable automatic subscription to kernel live patches:

   ```
   # yum kpatch auto
   Updating Subscription Management repositories.
   Last metadata expiration check: 1:38:21 ago on Fri 17 Sep 2021 07:29:53 AM EDT.
   Dependencies resolved.
   Package Architecture
   ===================================================
   Installing:
   kpatch-patch-5_14_0-1  x86_64
   kpatch-patch-5_14_0-2  x86_64
   Transaction Summary
   ==============================================================
   Install 2 Packages
   ```

   This command subscribes all currently installed kernels to receiving kernel live patches. The command also installs and applies the latest cumulative live patches, if any, for all installed kernels.

   In the future, when you update the kernel, live patches will automatically be installed during the new kernel installation process.

   The kernel patch module is also installed into the `/var/lib/kpatch/` directory to be loaded by the `systemd` system and service manager during future reboots.
NOTE

An empty live patching package will be installed when there are no live patches available for a given kernel. An empty live patching package will have a `kpatch_version-kpatch_release` of 0-0, for example `kpatch-patch-5_14_0-1-0-0.el9.x86_64.rpm`. The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

4. Optionally, verify that all installed kernels were patched:

```
# kpatch list
Loaded patch modules:
 kpatch_5_14_0_2_0_1 [enabled]

Installed patch modules:
 kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
 kpatch_5_14_0_2_0_1 (5.14.0-2.el9.x86_64)
```

The output shows that both the kernel you are running, and the other installed kernel have been patched with fixes from `kpatch-patch-5_14_0-1-0-1.el9.x86_64.rpm` and `kpatch-patch-5_14_0-2-1.el9.x86_64.rpm` packages respectively.

Additional resources

- `kpatch(1)` and `dnf-kpatch(8)` manual pages
- `Configuring basic system settings` in RHEL

11.8. UPDATING KERNEL PATCH MODULES

Since kernel patch modules are delivered and applied through RPM packages, updating a cumulative kernel patch module is like updating any other RPM package.

Prerequisites

- The system is subscribed to the live patching stream, as described in `Subscribing the currently installed kernels to the live patching stream`.

Procedure

- Update to a new cumulative version for the current kernel:

  ```
  # yum update "kpatch-patch = $(uname -r)"
  ```

  The command above automatically installs and applies any updates that are available for the currently running kernel. Including any future released cumulative live patches.

- Alternatively, update all installed kernel patch modules:

  ```
  # yum update "kpatch-patch"
  ```
When the system reboots into the same kernel, the kernel is automatically live patched again by the `kpatch.service` systemd service.

### Additional resources

- [Configuring basic system settings](#) in RHEL

### 11.9. REMOVING THE LIVE PATCHING PACKAGE

The following procedure describes how to disable the Red Hat Enterprise Linux kernel live patching solution by removing the live patching package.

**Prerequisites**

- Root permissions
- The live patching package is installed.

**Procedure**

1. Select the live patching package:

   ```
   # yum list installed | grep kpatch-patch
   kpatch-patch-5_14_0-1.x86_64 0-1.el9 @@commandline
   ...
   ```

   The example output above lists live patching packages that you installed.

2. Remove the live patching package:

   ```
   # yum remove kpatch-patch-5_14_0-1.x86_64
   ```

   When a live patching package is removed, the kernel remains patched until the next reboot, but the kernel patch module is removed from disk. On future reboot, the corresponding kernel will no longer be patched.

3. Reboot your system.

4. Verify that the live patching package has been removed:

   ```
   # yum list installed | grep kpatch-patch
   ```

   The command displays no output if the package has been successfully removed.

5. Optionally, verify that the kernel live patching solution is disabled:

   ```
   # kpatch list
   Loaded patch modules:
   ```

   The example output shows that the kernel is not patched and the live patching solution is not active because there are no patch modules that are currently loaded.
IMPORTANT
Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

Additional resources
- `kpatch(1)` manual page
- *Configuring basic system settings* in RHEL

11.10. UNINSTALLING THE KERNEL PATCH MODULE
The following procedure describes how to prevent the Red Hat Enterprise Linux kernel live patching solution from applying a kernel patch module on subsequent boots.

Prerequisites
- Root permissions
- A live patching package is installed.
- A kernel patch module is installed and loaded.

Procedure
1. Select a kernel patch module:

```
# kpatch list
```

```
Loaded patch modules:
kpatch_5_14_0_1_0_1 [enabled]
```

```
Installed patch modules:
kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
...
```

2. Uninstall the selected kernel patch module:

```
# kpatch uninstall kpatch_5_14_0_1_0_1
```

```
uninstalling kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
```

- Note that the uninstalled kernel patch module is still loaded:

```
# kpatch list
```

```
Loaded patch modules:
kpatch_5_14_0_1_0_1 [enabled]
```

```
Installed patch modules:
<NO_RESULT>
```

When the selected module is uninstalled, the kernel remains patched until the next reboot, but the kernel patch module is removed from disk.

3. Reboot your system.
4. Optionally, verify that the kernel patch module has been uninstalled:

```bash
# kpatch list
Loaded patch modules:
...```

The example output above shows no loaded or installed kernel patch modules, therefore the kernel is not patched and the kernel live patching solution is not active.

**IMPORTANT**

Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

**Additional resources**

- `kpatch(1)` manual page

**11.11. DISABLING KPATCHSERVICE**

The following procedure describes how to prevent the Red Hat Enterprise Linux kernel live patching solution from applying all kernel patch modules globally on subsequent boots.

**Prerequisites**

- Root permissions
- A live patching package is installed.
- A kernel patch module is installed and loaded.

**Procedure**

1. Verify `kpatch.service` is enabled:

```bash
# systemctl is-enabled kpatch.service
enabled```

2. Disable `kpatch.service`:

```bash
# systemctl disable kpatch.service
Removed /etc/systemd/system/multi-user.target.wants/kpatch.service.
```

- Note that the applied kernel patch module is still loaded:

```bash
# kpatch list
Loaded patch modules:
  kpatch_5_14_0_1_0_1 [enabled]

Installed patch modules:
  kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
```

3. Reboot your system.
4. Optionally, verify the status of `kpatch.service`:

```bash
# systemctl status kpatch.service
```

```
● kpatch.service - "Apply kpatch kernel patches"
 Loaded: loaded (/usr/lib/systemd/system/kpatch.service; disabled; vendor preset: disabled)
 Active: inactive (dead)
```

The example output testifies that `kpatch.service` has been disabled and is not running. Thereby, the kernel live patching solution is not active.

5. Verify that the kernel patch module has been unloaded:

```bash
# kpatch list
```

```
Loaded patch modules:

Installed patch modules:
kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
```

The example output above shows that a kernel patch module is still installed but the kernel is not patched.

**IMPORTANT**

Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

**Additional resources**

- `kpatch(1)` manual page
- *Configuring basic system settings* in RHEL