



Red Hat Enterprise Linux 8

Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Red Hat Enterprise Linux 8 Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

As a system administrator, you can configure the Linux kernel to optimize the operating system. Changes to the Linux kernel can improve system performance, security, and stability, as well as your ability to audit the system and troubleshoot problems.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	7
CHAPTER 1. THE LINUX KERNEL	8
1.1. WHAT THE KERNEL IS	8
1.2. RPM PACKAGES	8
1.3. THE LINUX KERNEL RPM PACKAGE OVERVIEW	9
1.4. DISPLAYING CONTENTS OF A KERNEL PACKAGE	9
1.5. INSTALLING SPECIFIC KERNEL VERSIONS	10
1.6. UPDATING THE KERNEL	10
1.7. SETTING A KERNEL AS DEFAULT	10
CHAPTER 2. MANAGING KERNEL MODULES	12
2.1. INTRODUCTION TO KERNEL MODULES	12
2.2. KERNEL MODULE DEPENDENCIES	12
2.3. LISTING INSTALLED KERNEL MODULES	13
2.4. LISTING CURRENTLY LOADED KERNEL MODULES	13
2.5. LISTING ALL INSTALLED KERNELS	14
2.6. DISPLAYING INFORMATION ABOUT KERNEL MODULES	14
2.7. LOADING KERNEL MODULES AT SYSTEM RUNTIME	15
2.8. UNLOADING KERNEL MODULES AT SYSTEM RUNTIME	16
2.9. UNLOADING KERNEL MODULES AT EARLY STAGES OF THE BOOT PROCESS	17
2.10. LOADING KERNEL MODULES AUTOMATICALLY AT SYSTEM BOOT TIME	19
2.11. PREVENTING KERNEL MODULES FROM BEING AUTOMATICALLY LOADED AT SYSTEM BOOT TIME	20
2.12. COMPILING CUSTOM KERNEL MODULES	22
CHAPTER 3. SIGNING A KERNEL AND MODULES FOR SECURE BOOT	25
3.1. PREREQUISITES	25
3.2. WHAT IS UEFI SECURE BOOT	26
3.3. UEFI SECURE BOOT SUPPORT	27
3.4. REQUIREMENTS FOR AUTHENTICATING KERNEL MODULES WITH X.509 KEYS	27
3.5. SOURCES FOR PUBLIC KEYS	28
3.6. GENERATING A PUBLIC AND PRIVATE KEY PAIR	29
3.7. EXAMPLE OUTPUT OF SYSTEM KEYRINGS	31
3.8. ENROLLING PUBLIC KEY ON TARGET SYSTEM BY ADDING THE PUBLIC KEY TO THE MOK LIST	32
3.9. SIGNING A KERNEL WITH THE PRIVATE KEY	33
3.10. SIGNING A GRUB BUILD WITH THE PRIVATE KEY	34
3.11. SIGNING KERNEL MODULES WITH THE PRIVATE KEY	35
3.12. LOADING SIGNED KERNEL MODULES	37
CHAPTER 4. CONFIGURING KERNEL COMMAND-LINE PARAMETERS	39
4.1. WHAT ARE KERNEL COMMAND-LINE PARAMETERS	39
4.2. UNDERSTANDING BOOT ENTRIES	39
4.3. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR ALL BOOT ENTRIES	40
4.4. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR A SINGLE BOOT ENTRY	41
4.5. CHANGING KERNEL COMMAND-LINE PARAMETERS TEMPORARILY AT BOOT TIME	42
4.6. CONFIGURING GRUB SETTINGS TO ENABLE SERIAL CONSOLE CONNECTION	43
CHAPTER 5. CONFIGURING KERNEL PARAMETERS AT RUNTIME	44
5.1. WHAT ARE KERNEL PARAMETERS	44
5.2. CONFIGURING KERNEL PARAMETERS TEMPORARILY WITH SYSCTL	45
5.3. CONFIGURING KERNEL PARAMETERS PERMANENTLY WITH SYSCTL	46
5.4. USING CONFIGURATION FILES IN /ETC/SYSCTL.D/ TO ADJUST KERNEL PARAMETERS	46

5.5. CONFIGURING KERNEL PARAMETERS TEMPORARILY THROUGH /PROC/SYS/	47
CHAPTER 6. MAKING TEMPORARY CHANGES TO THE GRUB MENU	48
6.1. INTRODUCTION TO GRUB	48
6.2. INTRODUCTION TO BOOT LOADER SPECIFICATION	49
6.3. BOOTING TO RESCUE MODE	50
6.4. BOOTING TO EMERGENCY MODE	50
6.5. BOOTING TO THE DEBUG SHELL	51
6.6. CONNECTING TO THE DEBUG SHELL	52
6.7. RESETTING THE ROOT PASSWORD USING AN INSTALLATION DISK	53
6.8. RESETTING THE ROOT PASSWORD USING RD.BREAK	55
6.9. ADDITIONAL RESOURCES	56
CHAPTER 7. MAKING PERSISTENT CHANGES TO THE GRUB BOOT LOADER	57
7.1. PREREQUISITES	57
7.2. LISTING THE DEFAULT KERNEL	57
7.3. VIEWING THE GRUB MENU ENTRY FOR A KERNEL	57
7.4. EDITING A KERNEL ARGUMENT	58
7.5. ADDING AND REMOVING ARGUMENTS FROM A GRUB MENU ENTRY	58
7.6. ADDING A NEW BOOT ENTRY	59
7.7. CHANGING THE DEFAULT BOOT ENTRY WITH GRUBBY	60
7.8. UPDATING ALL KERNEL MENUS WITH THE SAME ARGUMENTS	60
7.9. CHANGING DEFAULT KERNEL OPTIONS FOR CURRENT AND FUTURE KERNELS	60
7.10. ADDITIONAL RESOURCES	61
CHAPTER 8. BUILDING A CUSTOMIZED BOOT MENU	62
8.1. THE GRUB CONFIGURATION FILE	62
8.2. HIDING THE LIST OF BOOTABLE KERNELS	62
8.3. CHANGING THE DEFAULT BOOT ENTRY WITH THE GRUB CONFIGURATION FILE	63
CHAPTER 9. REINSTALLING GRUB	65
9.1. REINSTALLING GRUB ON BIOS-BASED MACHINES	65
9.2. REINSTALLING GRUB ON UEFI-BASED MACHINES	65
9.3. REINSTALLING GRUB ON IBM POWER MACHINES	66
9.4. RESETTING GRUB	66
CHAPTER 10. PROTECTING GRUB WITH A PASSWORD	68
10.1. SETTING PASSWORD PROTECTION ONLY FOR MODIFYING MENU ENTRIES	68
10.2. SETTING PASSWORD PROTECTION FOR MODIFYING AND BOOTING MENU ENTRIES	68
CHAPTER 11. KEEPING KERNEL PANIC PARAMETERS DISABLED IN VIRTUALIZED ENVIRONMENTS	70
11.1. WHAT IS A SOFT LOCKUP	70
11.2. PARAMETERS CONTROLLING KERNEL PANIC	70
11.3. SPURIOUS SOFT LOCKUPS IN VIRTUALIZED ENVIRONMENTS	71
CHAPTER 12. ADJUSTING KERNEL PARAMETERS FOR DATABASE SERVERS	72
12.1. INTRODUCTION TO DATABASE SERVERS	72
12.2. PARAMETERS AFFECTING PERFORMANCE OF DATABASE APPLICATIONS	72
CHAPTER 13. GETTING STARTED WITH KERNEL LOGGING	75
13.1. WHAT IS THE KERNEL RING BUFFER	75
13.2. ROLE OF PRINTK ON LOG-LEVELS AND KERNEL LOGGING	75
CHAPTER 14. INSTALLING KDUMP	77
14.1. WHAT IS KDUMP	77
14.2. INSTALLING KDUMP USING ANACONDA	77

14.3. INSTALLING KDUMP ON THE COMMAND LINE	78
CHAPTER 15. CONFIGURING KDUMP ON THE COMMAND LINE	79
15.1. ESTIMATING THE KDUMP SIZE	79
15.2. CONFIGURING KDUMP MEMORY USAGE	79
15.3. CONFIGURING THE KDUMP TARGET	81
15.4. CONFIGURING THE KDUMP CORE COLLECTOR	84
15.5. CONFIGURING THE KDUMP DEFAULT FAILURE RESPONSES	85
15.6. CONFIGURATION FILE FOR KDUMP	86
15.7. TESTING THE KDUMP CONFIGURATION	86
15.8. FILES PRODUCED BY KDUMP AFTER SYSTEM CRASH	88
15.9. ENABLING AND DISABLING THE KDUMP SERVICE	88
15.10. PREVENTING KERNEL DRIVERS FROM LOADING FOR KDUMP	89
15.11. RUNNING KDUMP ON SYSTEMS WITH ENCRYPTED DISK	90
CHAPTER 16. CONFIGURING KDUMP IN THE WEB CONSOLE	92
16.1. CONFIGURING KDUMP MEMORY USAGE AND TARGET LOCATION IN WEB CONSOLE	92
CHAPTER 17. ENABLING KDUMP	94
17.1. ENABLING KDUMP FOR ALL INSTALLED KERNELS	94
17.2. ENABLING KDUMP FOR A SPECIFIC INSTALLED KERNEL	94
17.3. DISABLING THE KDUMP SERVICE	95
CHAPTER 18. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS	97
18.1. MEMORY REQUIREMENTS FOR KDUMP	97
18.2. MINIMUM THRESHOLD FOR AUTOMATIC MEMORY RESERVATION	98
18.3. SUPPORTED KDUMP TARGETS	99
18.4. SUPPORTED KDUMP FILTERING LEVELS	101
18.5. SUPPORTED DEFAULT FAILURE RESPONSES	101
18.6. USING FINAL_ACTION PARAMETER	102
18.7. USING FAILURE_ACTION PARAMETER	102
CHAPTER 19. FIRMWARE ASSISTED DUMP MECHANISMS	104
19.1. FIRMWARE ASSISTED DUMP ON IBM POWERPC HARDWARE	104
19.2. ENABLING FIRMWARE ASSISTED DUMP MECHANISM	104
19.3. FIRMWARE ASSISTED DUMP MECHANISMS ON IBM Z HARDWARE	105
19.4. USING SADUMP ON FUJITSU PRIMEQUEST SYSTEMS	106
CHAPTER 20. ANALYZING A CORE DUMP	107
20.1. INSTALLING THE CRASH UTILITY	107
20.2. RUNNING AND EXITING THE CRASH UTILITY	107
20.3. DISPLAYING VARIOUS INDICATORS IN THE CRASH UTILITY	108
20.4. USING KERNEL OOPS ANALYZER	111
20.5. THE KDUMP HELPER TOOL	112
CHAPTER 21. USING EARLY KDUMP TO CAPTURE BOOT TIME CRASHES	113
21.1. ENABLING EARLY KDUMP	113
CHAPTER 22. APPLYING PATCHES WITH KERNEL LIVE PATCHING	115
22.1. LIMITATIONS OF KPATCH	115
22.2. SUPPORT FOR THIRD-PARTY LIVE PATCHING	115
22.3. ACCESS TO KERNEL LIVE PATCHES	116
22.4. THE PROCESS OF LIVE PATCHING KERNELS	116
22.5. SUBSCRIBING THE CURRENTLY INSTALLED KERNELS TO THE LIVE PATCHING STREAM	117
22.6. AUTOMATICALLY SUBSCRIBING ANY FUTURE KERNEL TO THE LIVE PATCHING STREAM	119

22.7. DISABLING AUTOMATIC SUBSCRIPTION TO THE LIVE PATCHING STREAM	120
22.8. UPDATING KERNEL PATCH MODULES	121
22.9. REMOVING THE LIVE PATCHING PACKAGE	122
22.10. UNINSTALLING THE KERNEL PATCH MODULE	123
22.11. DISABLING KPATCH.SERVICE	124
CHAPTER 23. SETTING SYSTEM RESOURCE LIMITS FOR APPLICATIONS BY USING CONTROL GROUPS ...	126
23.1. INTRODUCING CONTROL GROUPS	126
23.2. INTRODUCING KERNEL RESOURCE CONTROLLERS	127
23.3. INTRODUCING NAMESPACES	128
23.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1	129
CHAPTER 24. USING CGROUPS-V2 TO CONTROL DISTRIBUTION OF CPU TIME FOR APPLICATIONS .	133
24.1. MOUNTING CGROUPS-V2	133
24.2. PREPARING THE CGROUP FOR DISTRIBUTION OF CPU TIME	134
24.3. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU BANDWIDTH	138
24.4. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT	139
CHAPTER 25. USING CONTROL GROUPS VERSION 1 WITH SYSTEMD	142
25.1. ROLE OF SYSTEMD IN CONTROL GROUPS VERSION 1	142
25.2. CREATING TRANSIENT CONTROL GROUPS	143
25.3. CREATING PERSISTENT CONTROL GROUPS	144
25.4. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS ON THE COMMAND-LINE	144
25.5. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS WITH UNIT FILES	145
25.6. REMOVING TRANSIENT CONTROL GROUPS	146
25.7. REMOVING PERSISTENT CONTROL GROUPS	147
25.8. LISTING SYSTEMD UNITS	147
25.9. VIEWING SYSTEMD CGROUPS HIERARCHY	149
25.10. VIEWING RESOURCE CONTROLLERS	150
25.11. MONITORING RESOURCE CONSUMPTION	151
CHAPTER 26. CONFIGURING RESOURCE MANAGEMENT BY USING CGROUPS-V2 AND SYSTEMD ...	152
26.1. PREREQUISITES	152
26.2. INTRODUCTION TO RESOURCE DISTRIBUTION MODELS	152
26.3. ALLOCATING CPU RESOURCES BY USING SYSTEMD	153
26.4. CPU TIME ALLOCATION POLICY OPTIONS FOR SYSTEMD	153
26.5. ALLOCATING MEMORY RESOURCES BY USING SYSTEMD	154
26.6. MEMORY ALLOCATION CONFIGURATION OPTIONS FOR SYSTEMD	155
26.7. CONFIGURING I/O BANDWIDTH BY USING SYSTEMD	155
26.8. I/O BANDWIDTH CONFIGURATION OPTIONS FOR SYSTEMD	156
26.9. CONFIGURING CPUSET CONTROLLER BY USING SYSTEMD	156
CHAPTER 27. CONFIGURING CPU AFFINITY AND NUMA POLICIES USING SYSTEMD	158
27.1. CONFIGURING CPU AFFINITY USING SYSTEMD	158
27.2. CONFIGURING NUMA POLICIES USING SYSTEMD	159
27.3. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD	160
CHAPTER 28. ANALYZING SYSTEM PERFORMANCE WITH BPF COMPILER COLLECTION	161
28.1. INSTALLING THE BCC-TOOLS PACKAGE	161
28.2. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES	161
Using xfs slower to expose unexpectedly slow file system operations	164
CHAPTER 29. ENHANCING SECURITY WITH THE KERNEL INTEGRITY SUBSYSTEM	166

29.1. THE KERNEL INTEGRITY SUBSYSTEM	166
29.2. TRUSTED AND ENCRYPTED KEYS	167
29.3. WORKING WITH TRUSTED KEYS	167
29.4. WORKING WITH ENCRYPTED KEYS	169
29.5. ENABLING IMA AND EVM	170
29.6. COLLECTING FILE HASHES WITH INTEGRITY MEASUREMENT ARCHITECTURE	173
CHAPTER 30. CONFIGURING KERNEL PARAMETERS PERMANENTLY BY USING RHEL SYSTEM ROLES	...
	175
30.1. APPLYING SELECTED KERNEL PARAMETERS BY USING THE KERNEL_SETTINGS RHEL SYSTEM ROLE	175
CHAPTER 31. CONFIGURING THE GRUB BOOT LOADER BY USING RHEL SYSTEM ROLES 177
31.1. UPDATING THE EXISTING BOOT LOADER ENTRIES BY USING THE BOOTLOADER RHEL SYSTEM ROLE	177
31.2. SECURING THE BOOT MENU WITH PASSWORD BY USING THE BOOTLOADER RHEL SYSTEM ROLE	179
31.3. SETTING A TIMEOUT FOR THE BOOT LOADER MENU BY USING THE BOOTLOADER RHEL SYSTEM ROLE	181
31.4. COLLECTING THE BOOT LOADER CONFIGURATION INFORMATION BY USING THE BOOTLOADER RHEL SYSTEM ROLE	184
CHAPTER 32. USING ADVANCED ERROR REPORTING 186
32.1. OVERVIEW OF AER	186
32.2. COLLECTING AND DISPLAYING AER MESSAGES	186

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. THE LINUX KERNEL

Learn about the Linux kernel and the Linux kernel RPM package provided and maintained by Red Hat (Red Hat kernel). Keep the Red Hat kernel updated, which ensures the operating system has all the latest bug fixes, performance enhancements, and patches, and is compatible with new hardware.

1.1. WHAT THE KERNEL IS

The kernel is a core part of a Linux operating system that 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 easily upgraded and verified by the **YUM** package manager.



WARNING

Kernels that are not compiled by Red Hat are **not** supported by Red Hat.

1.2. RPM PACKAGES

An RPM package consists of an archive of files and metadata used to install and erase these files. Specifically, the RPM package contains the following parts:

GPG signature

The GPG signature is used to verify the integrity of the package.

Header (package metadata)

The RPM package manager uses this metadata to determine package dependencies, where to install files, and other information.

Payload

The payload is a **cpio** archive that contains files to install to the system.

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 SRPM can contain patches to source code.
- Binary RPM
A binary RPM contains the binaries built from the sources and patches.

1.3. 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

Provides 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

Provides 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

Provides kernel modules for rare hardware. Loading of the module is disabled by default.

kernel-debug

Provides a kernel with many debugging options enabled for kernel diagnosis, at the expense of reduced performance.

kernel-tools

Provides tools for manipulating the Linux kernel and supporting documentation.

kernel-devel

Provides the kernel headers and makefiles that are enough to build modules against the **kernel** package.

kernel-abi-stablelists

Provides information pertaining to the RHEL kernel ABI, including a list of kernel symbols required by external Linux kernel modules and a **yum** plug-in 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 required for building most standard programs.

Additional resources

- [*What are the kernel-core, kernel-modules, and kernel-modules-extras packages?*](#)

1.4. DISPLAYING CONTENTS OF A KERNEL PACKAGE

By querying the repository, you can see if a kernel package provides a specific file, such as a module. It is not necessary to download or install the package to display the file list.

Use the **dnf** utility to query the file list, for example, of the **kernel-core**, **kernel-modules-core**, or **kernel-modules** package. Note that the **kernel** package is a meta package that does not contain any files.

Procedure

1. List the available versions of a package:

```
$ yum repoquery <package_name>
```

2. Display the list of files in a package:

```
$ yum repoquery -l <package_name>
```

Additional resources

- [Packaging and distributing software](#)

1.5. INSTALLING SPECIFIC KERNEL VERSIONS

Install new kernels using the **yum** package manager.

Procedure

- To install a specific kernel version, enter the following command:

```
# yum install kernel-5.14.0
```

Additional resources

- [Red Hat Enterprise Linux Release Dates](#)

1.6. UPDATING THE KERNEL

Update the kernel using the **yum** package manager.

Procedure

1. To update the kernel, enter the following command:

```
# 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.



NOTE

When upgrading from RHEL 7 to RHEL 8, follow relevant sections of the [Upgrading from RHEL 7 to RHEL 8](#) document.

Additional resources

- [Managing software packages](#)

1.7. SETTING A KERNEL AS DEFAULT

Set a specific kernel as default by using the **grubby** command-line tool and GRUB.

Procedure

- Setting the kernel as default by using the **grubby** tool.
 - Enter 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 by 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 RPMs, self-built kernels, and manually adding the entries to the **/boot/loader/entries/** directory might change the index values.

CHAPTER 2. MANAGING KERNEL MODULES

Learn about kernel modules, how to display their information, and how to perform basic administrative tasks with kernel modules.

2.1. INTRODUCTION TO KERNEL MODULES

The Red Hat Enterprise Linux kernel can be extended with kernel modules, which provide optional additional pieces of functionality, without having to reboot the system. On RHEL 8, kernel modules are extra kernel code 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.

Similarly to the kernel, modules accept parameters that customize their behavior.

You can use the kernel tools to perform the following actions on modules:

- Inspect modules that are currently running.
- Inspect modules that are available to load into the kernel.
- Inspect parameters that a module accepts.
- Enable a mechanism to load and unload kernel modules into the running kernel.

2.2. 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 corresponding kernel version.

depmod

The dependency file is generated by the **depmod** program, included in the **kmod** package. Many utilities provided by **kmod** consider module dependencies when performing operations. Therefore, **manual** dependency-tracking is rarely necessary.



WARNING

The code of kernel modules executes in kernel-space in the unrestricted mode. Be mindful of what modules you are loading.

weak-modules

In addition to **depmod**, Red Hat Enterprise Linux provides the **weak-modules** script, which is a part of the **kmod** package. **weak-modules** determines the modules that are kABI-compatible with installed kernels. While checking modules kernel compatibility, **weak-modules** processes modules symbol dependencies from higher to lower release of kernel for which they were built. It processes each module independently of the kernel release.

Additional resources

- The **modules.dep(5)** manual page
- The **depmod(8)** manual page
- [What is the purpose of weak-modules script shipped with Red Hat Enterprise Linux?](#)
- [What is Kernel Application Binary Interface \(kABI\)?](#) (Red Hat Knowledgebase)

2.3. LISTING INSTALLED KERNEL MODULES

The **grubby --info=ALL** command displays an indexed list of installed kernels on **!BLS** and **BLS** installs.

Procedure

- List the installed kernels using the following command:

```
# grubby --info=ALL | grep title
```

The list of all installed kernels is displayed as follows:

```
title=Red Hat Enterprise Linux (4.18.0-20.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0-19.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0-12.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (0-rescue-2fb13ddde2e24fde9e6a246a942caed1) 8.0 (Ootpa)
```

This is the list of installed kernels of grubby-8.40-17 from the GRUB menu.

2.4. LISTING CURRENTLY LOADED KERNEL MODULES

View the currently loaded kernel modules.

Prerequisites

- The **kmod** package is installed.

Procedure

- To list all currently loaded kernel modules, enter:

```
$ lsmod
```

Module	Size	Used by
fuse	126976	3

```
uinput          20480 1
xt_CHECKSUM      16384 1
ipt_MASQUERADE   16384 1
xt_conntrack     16384 1
ipt_REJECT       16384 1
nft_counter      16384 16
nf_nat_tftp      16384 0
nf_conntrack_tftp 16384 1 nf_nat_tftp
tun              49152 1
bridge           192512 0
stp              16384 1 bridge
llc              16384 2 bridge,stp
nf_tables_set    32768 5
nft_fib_inet     16384 1
...
```

In the example above:

- The **Module** column provides the **names** of currently loaded modules.
- The **Size** column displays the amount of **memory** per module in kilobytes.
- The **Used by** column shows the number, and optionally the names of modules that are **dependent** on a particular module.

Additional resources

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

2.5. LISTING ALL INSTALLED KERNELS

Use the **grubby** utility to list all installed kernels on your system.

Prerequisites

- You have root permissions.

Procedure

- To list all installed kernels, enter:

```
# grubby --info=ALL | grep ^kernel

kernel="/boot/vmlinuz-4.18.0-305.10.2.el8_4.x86_64"
kernel="/boot/vmlinuz-4.18.0-240.el8.x86_64"
kernel="/boot/vmlinuz-0-rescue-41eb2e172d7244698abda79a51778f1b"
```

The output shows the path and versions of all the kernels installed.

2.6. DISPLAYING INFORMATION ABOUT KERNEL MODULES

Use the **modinfo** command to display some detailed information about the specified kernel module.

Prerequisites

- The **kmod** package is installed.

Procedure

- To display information about any kernel module, enter:

```
$ modinfo <KERNEL_MODULE_NAME>
```

For example:

```
$ modinfo virtio_net
```

```
filename:    /lib/modules/4.18.0-94.el8.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:     GPL
description: Virtio network driver
rhelversion: 8.1
srcversion:  2E9345B281A898A91319773
alias:       virtio:d00000001v*
depends:      net_failover
intree:      Y
name:        virtio_net
vermagic:    4.18.0-94.el8.x86_64 SMP mod_unload modversions
...
parm:        napi_weight:int
parm:        csum:bool
parm:        gso:bool
parm:        napi_tx:bool
```

You can query information about all available modules, regardless of whether they are loaded. 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

- The **modinfo(8)** manual page

2.7. LOADING KERNEL MODULES AT SYSTEM RUNTIME

The optimal way to expand the functionality of the Linux kernel is by loading kernel modules. Use the **modprobe** command to find and load a kernel module into the currently running kernel.



IMPORTANT

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

Prerequisites

- Root permissions
- The **kmod** package is installed.
- The respective kernel module is not loaded. To ensure this is the case, list the [Listing currently 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.

Verification

- 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
```

Additional resources

- The **modprobe(8)** manual page

2.8. UNLOADING KERNEL MODULES AT SYSTEM RUNTIME

To unload certain kernel modules from the running kernel, use the **modprobe** command to find and unload a kernel module at system runtime from the currently loaded kernel.



WARNING

You must not unload the kernel modules that are used by the running system because it can lead to an unstable or non-operational system.



IMPORTANT

After finishing the unloading of inactive kernel modules, the modules that are defined to be automatically loaded on boot, will not remain unloaded after rebooting the system. For information about how to prevent this outcome, see [Preventing kernel modules from being automatically loaded at system boot time](#).

Prerequisites

- You have root permissions.
- The **kmod** package is installed.

Procedure

1. List all the loaded kernel modules:

```
# lsmod
```

2. Select the 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](#).

3. 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.

Verification

- Optionally, verify the relevant module was unloaded:

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

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

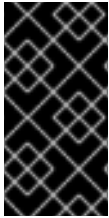
Additional resources

- **modprobe(8)** manual page

2.9. UNLOADING KERNEL MODULES AT EARLY STAGES OF THE BOOT PROCESS

In certain situations, for example, when the kernel module has a code that causes the system to become unresponsive, and the user is not able to reach the stage to permanently disable the rogue kernel module, you might need to unload a kernel module early in the booting process. To temporarily block the loading of the kernel module, you can use a boot loader.

You can edit the relevant boot loader entry to unload the required kernel module before the booting sequence continues.



IMPORTANT

The changes described in this procedure **will not persist** after the next reboot. For information about 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 that you want to prevent from loading for some reason.

Procedure

1. Boot the system into the boot loader.
2. Use the cursor keys to highlight the relevant boot loader entry.
3. Press the **e** key to edit the entry.

Figure 2.1. Kernel boot menu

```
Red Hat Enterprise Linux (4.18.0-305.10.2.el8_4.x86_64) 8.4 (0otpa)
Red Hat Enterprise Linux (4.18.0-305.3.1.el8_4.x86_64) 8.4 (0otpa)
Red Hat Enterprise Linux (4.18.0-240.22.1.el8_3.x86_64) 8.3 (0otpa)
Red Hat Enterprise Linux (0-rescue-bfeb014063334b2da66b978e95804445) 8.3→

Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

4. Use the cursor keys to navigate to the line that starts with **linux**.
5. Append **modprobe.blacklist=module_name** to the end of the line.

Figure 2.2. Kernel boot entry

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-305.10.2.el8_4.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap rhgb quiet modprobe.blacklist=serio_raw
initrd ($root)/initramfs-4.18.0-305.10.2.el8_4.x86_64.img $tuned_initrd

Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
discard edits and return to the menu. Pressing Tab lists
possible completions.
```

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

6. Press **Ctrl+X** to boot using the modified configuration.

Verification

- After the system boots, verify that the relevant kernel module is not loaded:

```
# lsmod | grep serio_raw
```

Additional resources

- [Managing kernel modules](#)

2.10. LOADING KERNEL MODULES AUTOMATICALLY AT SYSTEM BOOT TIME

Configure a kernel module to load it 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
```

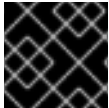
**NOTE**

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.

Verification

1. After reboot, verify the relevant module is loaded:

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

**IMPORTANT**

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

Additional resources

- **modules-load.d(5)** manual page

2.11. PREVENTING KERNEL MODULES FROM BEING AUTOMATICALLY LOADED AT SYSTEM BOOT TIME

You can prevent the system from loading a kernel module automatically during the boot process by listing the module in **modprobe** configuration file with a corresponding command.

Prerequisites

- The commands in this procedure require root privileges. Either use **su -** to switch to the root user or preface the commands with **sudo**.
- The **kmod** package is installed.
- Ensure that your current system configuration does not require a kernel module you plan to deny.

Procedure

1. List modules loaded to the currently running kernel by using the **lsmod** command:

```
$ lsmod
Module                Size  Used by
tls                   131072 0
uinput                 20480 1
snd_seq_dummy         16384 0
snd_hrtimer            16384 1
...
```

In the output, identify the module you want to prevent from getting loaded.

- Alternatively, identify an unloaded kernel module you want to prevent from potentially loading in the **/lib/modules/<KERNEL-VERSION>/kernel/<SUBSYSTEM>/** directory, for example:

■


```
$ ls /lib/modules/4.18.0-477.20.1.el8_8.x86_64/kernel/crypto/
ansi_cprng.ko.xz      chacha20poly1305.ko.xz  md4.ko.xz
serpent_generic.ko.xz
anubis.ko.xz          cmac.ko.xz...
```

2. Create a configuration file serving as a denylist:

```
# touch /etc/modprobe.d/denylist.conf
```

3. In a text editor of your choice, combine the names of modules you want to exclude from automatic loading to the kernel with the **blacklist** configuration command, for example:

```
# Prevents <KERNEL-MODULE-1> from being loaded
blacklist <MODULE-NAME-1>
install <MODULE-NAME-1> /bin/false

# Prevents <KERNEL-MODULE-2> from being loaded
blacklist <MODULE-NAME-2>
install <MODULE-NAME-2> /bin/false

...
```

Because the **blacklist** command does not prevent the module from getting loaded as a dependency for another kernel module that is not in a denylist, you must also define the **install** line. In this case, the system runs **/bin/false** instead of installing the module. The lines starting with a hash sign are comments you can use 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.

4. Create a backup copy of the current initial RAM disk image before rebuilding:

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

- Alternatively, create a backup copy of an initial RAM disk image which corresponds to the kernel version for which you want to prevent kernel modules from automatic loading:

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

5. Generate a new initial RAM disk image to apply the changes:

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

- If you build an initial RAM disk image for a different kernel version than your system currently uses, specify both target **initramfs** and kernel version:

```
# dracut -f -v /boot/initramfs-<TARGET-VERSION>.img <CORRESPONDING-TARGET-KERNEL-VERSION>
```

- Restart the system:

```
$ reboot
```



IMPORTANT

The changes described in this procedure **will take effect and persist** after rebooting the system. If you incorrectly list a key kernel module in the denylist, you can switch the system to an unstable or non-operational state.

Additional resources

- [How do I prevent a kernel module from loading automatically?](#) (Red Hat Knowledgebase)
- modprobe.d(5)** and **dracut(8)** man pages on your system

2.12. 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.

```
# dnf install kernel-devel-$(uname -r) gcc elfutils-libelf-devel
```

- You have root permissions.
- You created the **/root/testmodule/** directory where you compile the custom kernel module.

Procedure

- Create the **/root/testmodule/test.c** file with the following content.

```
#include <linux/module.h>
#include <linux/kernel.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** header file contains function declarations and macro definitions that are shared across multiple C source files.

2. Follow the **init_module()** and **cleanup_module()** functions to start and end the kernel logging function **printk()**, which prints text.
3. Create the **/root/testmodule/Makefile** file with the following content.

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

The Makefile contains instructions for the compiler to produce an object file 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 can instruct to build **test.ko** as a built-in kernel module.

4. Compile the kernel module.

```
# make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
make: Entering directory '/usr/src/kernels/4.18.0-305.el8.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/4.18.0-305.el8.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 152
-rw-r--r--. 1 root root 16 Jul 26 08:19 Makefile
-rw-r--r--. 1 root root 25 Jul 26 08:20 modules.order
-rw-r--r--. 1 root root 0 Jul 26 08:20 Module.symvers
-rw-r--r--. 1 root root 224 Jul 26 08:18 test.c
-rw-r--r--. 1 root root 62176 Jul 26 08:20 test.ko
-rw-r--r--. 1 root root 25 Jul 26 08:20 test.mod
-rw-r--r--. 1 root root 849 Jul 26 08:20 test.mod.c
-rw-r--r--. 1 root root 50936 Jul 26 08:20 test.mod.o
-rw-r--r--. 1 root root 12912 Jul 26 08:20 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/4.18.0-305.el8.x86_64/test.ko
```

5. Verify that the kernel module was successfully loaded:

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

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

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

CHAPTER 3. SIGNING A KERNEL AND MODULES FOR SECURE BOOT

You can enhance the security of your system by using a signed kernel and signed kernel modules. On UEFI-based build systems where Secure Boot is enabled, you can self-sign a privately built kernel or kernel modules. Furthermore, you can import your public key into a target system where you want to deploy your kernel or kernel modules.

If Secure Boot is enabled, all of the following components have to be signed with a private key and authenticated with the corresponding public key:

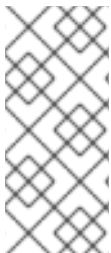
- UEFI operating system boot loader
- The Red Hat Enterprise Linux kernel
- All kernel modules

If any of these components are not signed and authenticated, the system cannot finish the booting process.

RHEL 8 includes:

- Signed boot loaders
- Signed kernels
- Signed kernel modules

In addition, the signed first-stage boot loader and the signed kernel include embedded Red Hat public keys. These signed executable binaries and embedded keys enable RHEL 8 to install, boot, and run with the Microsoft UEFI Secure Boot Certification Authority keys. These keys are provided by the UEFI firmware on systems that support UEFI Secure Boot.



NOTE

- Not all UEFI-based systems include support for Secure Boot.
- The build system, where you build and sign your kernel module, does not need to have UEFI Secure Boot enabled and does not even need to be a UEFI-based system.

3.1. PREREQUISITES

- To be able to sign externally built kernel modules, install the utilities from the following packages:

```
# yum install pesign openssl kernel-devel mokutil keyutils
```

Table 3.1. Required utilities

Utility	Provided by package	Used on	Purpose
efikeygen	pesign	Build system	Generates public and private X.509 key pair

Utility	Provided by package	Used on	Purpose
openssl	openssl	Build system	Exports the unencrypted private key
sign-file	kernel-devel	Build system	Executable file used to sign a kernel module with the private key
mokutil	mokutil	Target system	Optional utility used to manually enroll the public key
keyctl	keyutils	Target system	Optional utility used to display public keys in the system keyring

3.2. WHAT IS UEFI SECURE BOOT

With the *Unified Extensible Firmware Interface* (UEFI) Secure Boot technology, you can prevent the execution of the kernel-space code that is not signed by a trusted key. The system boot loader is signed with a cryptographic key. The database of public keys in the firmware authorizes the process of signing the key. You can subsequently verify a signature in the next-stage boot loader and the kernel.

UEFI Secure Boot establishes a chain of trust from the firmware to the signed drivers and kernel modules as follows:

- An UEFI private key signs, and a public key authenticates the **shim** first-stage boot loader. A *certificate authority* (CA) in turn signs the public key. The CA is stored in the firmware database.
- The **shim** file contains the Red Hat public key **Red Hat Secure Boot (CA key 1)** to authenticate the GRUB boot loader and the kernel.
- The kernel in turn contains public keys to authenticate drivers and modules.

Secure Boot is the boot path validation component of the UEFI specification. The specification defines:

- Programming interface for cryptographically protected UEFI variables in non-volatile storage.
- Storing the trusted X.509 root certificates in UEFI variables.
- Validation of UEFI applications such as boot loaders and drivers.
- Procedures to revoke known-bad certificates and application hashes.

UEFI Secure Boot helps in the detection of unauthorized changes but does **not**:

- Prevent installation or removal of second-stage boot loaders.
- Require explicit user confirmation of such changes.
- Stop boot path manipulations. Signatures are verified during booting but, not when the boot loader is installed or updated.

If the boot loader or the kernel are not signed by a system trusted key, Secure Boot prevents them from starting.

3.3. UEFI SECURE BOOT SUPPORT

You can install and run RHEL 8 on systems with enabled UEFI Secure Boot if the kernel and all the loaded drivers are signed with a trusted key. Red Hat provides kernels and drivers that are signed and authenticated by the relevant Red Hat keys.

If you want to load externally built kernels or drivers, you must sign them as well.

Restrictions imposed by UEFI Secure Boot

- The system only runs the kernel-mode code after its signature has been properly authenticated.
- GRUB module loading is disabled because there is no infrastructure for signing and verification of GRUB modules. Allowing module loading would run untrusted code within the security perimeter defined by Secure Boot.
- Red Hat provides a signed GRUB binary that has all supported modules on RHEL 8.

Additional resources

- [Restrictions Imposed by UEFI Secure Boot](#)

3.4. REQUIREMENTS FOR AUTHENTICATING KERNEL MODULES WITH X.509 KEYS

In RHEL 8, when a kernel module is loaded, the kernel checks the signature of the module against the public X.509 keys from the kernel system keyring (**.builtin_trusted_keys**) and the kernel platform keyring (**.platform**). The **.platform** keyring provides keys from third-party platform providers and custom public keys. The keys from the kernel system **.blacklist** keyring are excluded from verification.

You need to meet certain conditions to load kernel modules on systems with enabled UEFI Secure Boot functionality:

- If UEFI Secure Boot is enabled or if the **module.sig_enforce** kernel parameter has been specified:
 - You can only load those signed kernel modules whose signatures were authenticated against keys from the system keyring (**.builtin_trusted_keys**) and the platform keyring (**.platform**).
 - The public key must not be on the system revoked keys keyring (**.blacklist**).
- If UEFI Secure Boot is disabled and the **module.sig_enforce** kernel parameter has not been specified:
 - You can load unsigned kernel modules and signed kernel modules without a public key.
- If the system is not UEFI-based or if UEFI Secure Boot is disabled:
 - Only the keys embedded in the kernel are loaded onto **.builtin_trusted_keys** and **.platform**.
 - You have no ability to augment that set of keys without rebuilding the kernel.

Table 3.2. Kernel module authentication requirements for loading

Module signed	Public key found and signature valid	UEFI Secure Boot state	sig_enforce	Module load	Kernel tainted
Unsigned	-	Not enabled	Not enabled	Succeeds	Yes
		Not enabled	Enabled	Fails	-
		Enabled	-	Fails	-
Signed	No	Not enabled	Not enabled	Succeeds	Yes
		Not enabled	Enabled	Fails	-
		Enabled	-	Fails	-
Signed	Yes	Not enabled	Not enabled	Succeeds	No
		Not enabled	Enabled	Succeeds	No
		Enabled	-	Succeeds	No

3.5. SOURCES FOR PUBLIC KEYS

During boot, the kernel loads X.509 keys from a set of persistent key stores into the following keyrings:

- The system keyring (**.builtin_trusted_keys**)
- The **.platform** keyring
- The system **.blacklist** keyring

Table 3.3. Sources for system keyrings

Source of X.509 keys	User can add keys	UEFI Secure Boot state	Keys loaded during boot
Embedded in kernel	No	-	.builtin_trusted_keys
UEFI db	Limited	Not enabled	No
		Enabled	.platform
Embedded in the shim boot loader	No	Not enabled	No
		Enabled	.platform

Source of X.509 keys	User can add keys	UEFI Secure Boot state	Keys loaded during boot
Machine Owner Key (MOK) list	Yes	Not enabled	No
		Enabled	.platform

.builtin_trusted_keys

- A keyring that is built on boot.
- Provides trusted public keys.
- **root** privileges are required to view the keys.

.platform

- A keyring that is built on boot.
- Provides keys from third-party platform providers and custom public keys.
- **root** privileges are required to view the keys.

.blacklist

- A keyring with X.509 keys which have been revoked.
- A module signed by a key from **.blacklist** will fail authentication even if your public key is in **.builtin_trusted_keys**.

UEFI Secure Bootdb

- A signature database.
- Stores keys (hashes) of UEFI applications, UEFI drivers, and boot loaders.
- The keys can be loaded on the machine.

UEFI Secure Bootdbx

- A revoked signature database.
- Prevents keys from getting loaded.
- The revoked keys from this database are added to the **.blacklist** keyring.

3.6. GENERATING A PUBLIC AND PRIVATE KEY PAIR

To use a custom kernel or custom kernel modules on a Secure Boot-enabled system, you must generate a public and private X.509 key pair. You can use the generated private key to sign the kernel or the kernel modules. You can also validate the signed kernel or kernel modules by adding the corresponding public key to the Machine Owner Key (MOK) for Secure Boot.

**WARNING**

Apply strong security measures and access policies to guard the contents of your private key. In the wrong hands, the key could be used to compromise any system which is authenticated by the corresponding public key.

Procedure

- Create an X.509 public and private key pair:
 - If you only want to sign custom kernel *modules*:

```
# efkeygen --dbdir /etc/pki/pesign \
--self-sign \
--module \
--common-name 'CN=Organization signing key' \
--nickname 'Custom Secure Boot key'
```

- If you want to sign custom *kernel*:

```
# efkeygen --dbdir /etc/pki/pesign \
--self-sign \
--kernel \
--common-name 'CN=Organization signing key' \
--nickname 'Custom Secure Boot key'
```

- When the RHEL system is running FIPS mode:

```
# efkeygen --dbdir /etc/pki/pesign \
--self-sign \
--kernel \
--common-name 'CN=Organization signing key' \
--nickname 'Custom Secure Boot key'
--token 'NSS FIPS 140-2 Certificate DB'
```

**NOTE**

In FIPS mode, you must use the **--token** option so that **efkeygen** finds the default "NSS Certificate DB" token in the PKI database.

The public and private keys are now stored in the **/etc/pki/pesign/** directory.

**IMPORTANT**

It is a good security practice to sign the kernel and the kernel modules within the validity period of its signing key. However, the **sign-file** utility does not warn you and the key will be usable in RHEL 8 regardless of the validity dates.

Additional resources

- **openssl(1)** manual page
- [RHEL Security Guide](#)
- [Enrolling public key on target system by adding the public key to the MOK list](#)

3.7. EXAMPLE OUTPUT OF SYSTEM KEYRINGS

You can display information about the keys on the system keyrings using the **keyctl** utility from the **keyutils** package.

Prerequisites

- You have root permissions.
- You have installed the **keyctl** utility from the **keyutils** package.

Example 3.1. Keyrings output

The following is a shortened example output of **.builtin_trusted_keys**, **.platform**, and **.blacklist** keyrings from a RHEL 8 system where UEFI Secure Boot is enabled.

```
# keyctl list %:.builtin_trusted_keys
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1): 4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...

# keyctl list %:.platform
4 keys in keyring:
...asymmetric: VMware, Inc.: 4ad8da0472073...
...asymmetric: Red Hat Secure Boot CA 5: cc6fafa72...
...asymmetric: Microsoft Windows Production PCA 2011: a929f298e1...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4e0bd82...

# keyctl list %:.blacklist
4 keys in keyring:
...blacklist: bin:f5ff83a...
...blacklist: bin:0dfdbec...
...blacklist: bin:38f1d22...
...blacklist: bin:51f831f...
```

The **.builtin_trusted_keys** keyring in the example shows the addition of two keys from the UEFI Secure Boot **db** keys as well as the **Red Hat Secure Boot (CA key 1)**, which is embedded in the **shim** boot loader.

Example 3.2. Kernel console output

The following example shows the kernel console output. The messages identify the keys with an UEFI Secure Boot related source. These include UEFI Secure Boot **db**, embedded **shim**, and MOK list.

```
# dmesg | egrep 'integrity.*cert'
```

```
[1.512966] integrity: Loading X.509 certificate: UEFI:db
```

```
[1.513027] integrity: Loaded X.509 cert 'Microsoft Windows Production PCA 2011: a929023...
```

```
[1.513028] integrity: Loading X.509 certificate: UEFI:db
```

```
[1.513057] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309...
```

```
[1.513298] integrity: Loading X.509 certificate: UEFI:MokListRT (MOKvar table)
```

```
[1.513549] integrity: Loaded X.509 cert 'Red Hat Secure Boot CA 5: cc6fa5e72868ba494e93...
```

Additional resources

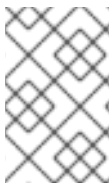
- **keyctl(1)**, **dmesg(1)** manual pages

3.8. ENROLLING PUBLIC KEY ON TARGET SYSTEM BY ADDING THE PUBLIC KEY TO THE MOK LIST

You must authenticate your public key on a system for kernel or kernel module access and enroll it in the platform keyring (**.platform**) of the target system. When RHEL 8 boots on a UEFI-based system with Secure Boot enabled, the kernel imports public keys from the **db** key database and excludes revoked keys from the **dbx** database.

The Machine Owner Key (MOK) facility allows expanding the UEFI Secure Boot key database. When booting RHEL 8 on UEFI-enabled systems with Secure Boot enabled, keys on the MOK list are added to the platform keyring (**.platform**), along with the keys from the Secure Boot database. The list of MOK keys is stored securely and persistently in the same way, but it is a separate facility from the Secure Boot databases.

The MOK facility is supported by **shim**, **MokManager**, **GRUB**, and the **mokutil** utility that enables secure key management and authentication for UEFI-based systems.



NOTE

To get the authentication service of your kernel module on your systems, consider requesting your system vendor to incorporate your public key into the UEFI Secure Boot key database in their factory firmware image.

Prerequisites

- You have generated a public and private key pair and know the validity dates of your public keys. For details, see [Generating a public and private key pair](#).

Procedure

1. Export your public key to the **sb_cert.cer** file:

```
# certutil -d /etc/pki/pesign \
    -n 'Custom Secure Boot key' \
    -Lr \
    > sb_cert.cer
```

2. Import your public key into the MOK list:

```
# mokutil --import sb_cert.cer
```

3. Enter a new password for this MOK enrollment request.
4. Reboot the machine.
The **shim** boot loader notices the pending MOK key enrollment request and it launches **MokManager.efi** to enable you to complete the enrollment from the UEFI console.
5. Choose **Enroll MOK**, enter the password you previously associated with this request when prompted, and confirm the enrollment.
Your public key is added to the MOK list, which is persistent.

Once a key is on the MOK list, it will be automatically propagated to the **.platform** keyring on this and subsequent boots when UEFI Secure Boot is enabled.

3.9. SIGNING A KERNEL WITH THE PRIVATE KEY

You can obtain enhanced security benefits on your system by loading a signed kernel if the UEFI Secure Boot mechanism is enabled.

Prerequisites

- You have generated a public and private key pair and know the validity dates of your public keys. For details, see [Generating a public and private key pair](#).
- You have enrolled your public key on the target system. For details, see [Enrolling public key on target system by adding the public key to the MOK list](#).
- You have a kernel image in the ELF format available for signing.

Procedure

- On the x64 architecture:
 - a. Create a signed image:

```
# pesign --certificate 'Custom Secure Boot key' \
    --in vmlinuz-version \
    --sign \
    --out vmlinuz-version.signed
```

Replace **version** with the version suffix of your **vmlinuz** file, and **Custom Secure Boot key** with the name that you chose earlier.

- b. Optional: Check the signatures:

```
# pesign --show-signature \
    --in vmlinuz-version.signed
```

- c. Overwrite the unsigned image with the signed image:

```
# mv vmlinuz-version.signed vmlinuz-version
```

- On the 64-bit ARM architecture:
 - a. Decompress the **vmlinuz** file:

```
# zcat vmlinuz-version > vmlinux-version
```

- b. Create a signed image:

```
# pesign --certificate 'Custom Secure Boot key' \
  --in vmlinux-version \
  --sign \
  --out vmlinux-version.signed
```

- c. Optional: Check the signatures:

```
# pesign --show-signature \
  --in vmlinux-version.signed
```

- d. Compress the **vmlinux** file:

```
# gzip --to-stdout vmlinux-version.signed > vmlinuz-version
```

- e. Remove the uncompressed **vmlinux** file:

```
# rm vmlinux-version*
```

3.10. SIGNING A GRUB BUILD WITH THE PRIVATE KEY

On a system where the UEFI Secure Boot mechanism is enabled, you can sign a GRUB build with a custom existing private key. You must do this if you are using a custom GRUB build, or if you have removed the Microsoft trust anchor from your system.

Prerequisites

- You have generated a public and private key pair and know the validity dates of your public keys. For details, see [Generating a public and private key pair](#).
- You have enrolled your public key on the target system. For details, see [Enrolling public key on target system by adding the public key to the MOK list](#).
- You have a GRUB EFI binary available for signing.

Procedure

- On the x64 architecture:
 - a. Create a signed GRUB EFI binary:

```
# pesign --in /boot/efi/EFI/redhat/grubx64.efi \
  --out /boot/efi/EFI/redhat/grubx64.efi.signed \
  --certificate 'Custom Secure Boot key' \
  --sign
```

Replace ***Custom Secure Boot key*** with the name that you chose earlier.

- b. Optional: Check the signatures:

```
# pesign --in /boot/efi/EFI/redhat/grubx64.efi.signed \
--show-signature
```

- c. Overwrite the unsigned binary with the signed binary:

```
# mv /boot/efi/EFI/redhat/grubx64.efi.signed \
/boot/efi/EFI/redhat/grubx64.efi
```

- On the 64-bit ARM architecture:

- a. Create a signed GRUB EFI binary:

```
# pesign --in /boot/efi/EFI/redhat/grubaa64.efi \
--out /boot/efi/EFI/redhat/grubaa64.efi.signed \
--certificate 'Custom Secure Boot key' \
--sign
```

Replace **Custom Secure Boot key** with the name that you chose earlier.

- b. Optional: Check the signatures:

```
# pesign --in /boot/efi/EFI/redhat/grubaa64.efi.signed \
--show-signature
```

- c. Overwrite the unsigned binary with the signed binary:

```
# mv /boot/efi/EFI/redhat/grubaa64.efi.signed \
/boot/efi/EFI/redhat/grubaa64.efi
```

3.11. SIGNING KERNEL MODULES WITH THE PRIVATE KEY

You can enhance the security of your system by loading signed kernel modules if the UEFI Secure Boot mechanism is enabled.

Your signed kernel module is also loadable on systems where UEFI Secure Boot is disabled or on a non-UEFI system. As a result, you do not need to provide both, a signed and unsigned version of your kernel module.

Prerequisites

- You have generated a public and private key pair and know the validity dates of your public keys. For details, see [Generating a public and private key pair](#).
- You have enrolled your public key on the target system. For details, see [Enrolling public key on target system by adding the public key to the MOK list](#).
- You have a kernel module in ELF image format available for signing.

Procedure

1. Export your public key to the **sb_cert.cer** file:

```
# certutil -d /etc/pki/pesign \
```

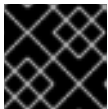
```
-n 'Custom Secure Boot key' \
-Lr \
> sb_cert.cer
```

2. Extract the key from the NSS database as a PKCS #12 file:

```
# pk12util -o sb_cert.p12 \
-n 'Custom Secure Boot key' \
-d /etc/pki/pesign
```

3. When the previous command prompts, enter a new password that encrypts the private key.
4. Export the unencrypted private key:

```
# openssl pkcs12 \
-in sb_cert.p12 \
-out sb_cert.priv \
-nocerts \
-nodes
```



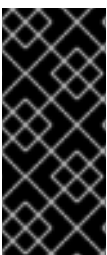
IMPORTANT

Keep the unencrypted private key secure.

5. Sign your kernel module. The following command appends the signature directly to the ELF image in your kernel module file:

```
# /usr/src/kernels/$(uname -r)/scripts/sign-file \
sha256 \
sb_cert.priv \
sb_cert.cer \
my_module.ko
```

Your kernel module is now ready for loading.



IMPORTANT

In RHEL 8, the validity dates of the key pair matter. The key does not expire, but the kernel module must be signed within the validity period of its signing key. The **sign-file** utility will not warn you of this. For example, a key that is only valid in 2019 can be used to authenticate a kernel module signed in 2019 with that key. However, users cannot use that key to sign a kernel module in 2020.

Verification

1. Display information about the kernel module's signature:

```
# modinfo my_module.ko | grep signer
signer:      Your Name Key
```

Check that the signature lists your name as entered during generation.



NOTE

The appended signature is not contained in an ELF image section and is not a formal part of the ELF image. Therefore, utilities such as **readelf** cannot display the signature on your kernel module.

2. Load the module:

```
# insmod my_module.ko
```

3. Remove (unload) the module:

```
# modprobe -r my_module.ko
```

Additional resources

- [Displaying information about kernel modules](#)

3.12. LOADING SIGNED KERNEL MODULES

After enrolling your public key in the system keyring (**.builtin_trusted_keys**) and the MOK list, and signing kernel modules with your private key, you can load them using the **modprobe** command.

Prerequisites

- You have generated the public and private key pair. For details, see [Generating a public and private key pair](#).
- You have enrolled the public key into the system keyring. For details, see [Enrolling public key on target system by adding the public key to the MOK list](#).
- You have signed a kernel module with the private key. For details, see [Signing kernel modules with the private key](#).
- Install the **kernel-modules-extra** package, which creates the **/lib/modules/\$(uname -r)/extra/** directory:

```
# yum -y install kernel-modules-extra
```

Procedure

1. Verify that your public keys are on the system keyring:

```
# keyctl list %:.platform
```

2. Copy the kernel module into the **extra/** directory of the kernel that you want:

```
# cp my_module.ko /lib/modules/$(uname -r)/extra/
```

3. Update the modular dependency list:

```
# depmod -a
```

4. Load the kernel module:

```
# modprobe -v my_module
```

5. Optional: To load the module on boot, add it to the `/etc/modules-load.d/my_module.conf` file:

```
# echo "my_module" > /etc/modules-load.d/my_module.conf
```

Verification

- Verify that the module was successfully loaded:

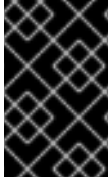
```
# lsmod | grep my_module
```

Additional resources

- [Managing kernel modules](#)

CHAPTER 4. CONFIGURING KERNEL COMMAND-LINE PARAMETERS

With kernel command-line parameters, you can change the behavior of certain aspects of the Red Hat Enterprise Linux kernel at boot time. As a system administrator, you control which options get set at boot. Note that certain kernel behaviors can only be set at boot time.



IMPORTANT

Changing the behavior of the system by modifying kernel command-line parameters can have negative effects on your system. Always test changes before deploying them in production. For further guidance, contact Red Hat Support.

4.1. WHAT ARE KERNEL COMMAND-LINE PARAMETERS

With kernel command-line parameters, you can overwrite default values and set specific hardware settings. At boot time, you can configure the following features:

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

By default, the kernel command-line parameters for systems using the GRUB boot loader are defined in the **kernelopts** variable of the **/boot/grub2/grubenv** file for each kernel boot entry.



NOTE

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

You can manipulate boot loader configuration files by using the **grubby** utility. With **grubby**, you can perform these actions:

- Change the default boot entry.
- Add or remove arguments from a GRUB menu entry.

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](#)
- The **grubby(8)** manual page

4.2. UNDERSTANDING BOOT ENTRIES

A boot entry is a collection of options 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:

```
6f9cc9cb7d7845d49698c9537337cedc-4.18.0-5.el8.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 that contains the kernel command-line parameters. The configuration file can have the following contents:

```
title Red Hat Enterprise Linux (4.18.0-74.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-74.el8.x86_64
linux /vmlinuz-4.18.0-74.el8.x86_64
initrd /initramfs-4.18.0-74.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20190227183418-4.18.0-74.el8.x86_64
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](#)

4.3. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR ALL BOOT ENTRIES

Change kernel command-line parameters for all boot entries on your system.

Prerequisites

- **grubby** utility is installed on your system.
- **zipl** utility is installed on your IBM Z system.

Procedure

- To add a parameter:

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

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

- On IBM Z, update the boot menu:

```
# zipl
```

- To remove a parameter:

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

- On IBM Z, update the boot menu:

```
# zipl
```

**NOTE**

Newly installed kernels inherit the kernel command-line parameters from your previously configured kernels.

Additional resources

- [What are kernel command-line parameters](#)
- **grubby(8)** and **zipl(8)** manual pages

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

Make changes in kernel command-line parameters for a single boot entry on your system.

Prerequisites

- **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, update the boot menu:

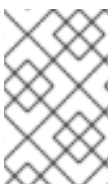
```
# zipl
```

- To remove a parameter:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --remove-args="
<PARAMETER_TO_REMOVE>"
```

- On IBM Z, update the boot menu:

```
# zipl
```

**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 GRUB 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/<RELEVANT_KERNEL_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.

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

Make temporary changes to a Kernel Menu Entry by changing the kernel parameters only during a single boot process.



NOTE

This procedure applies only for a single boot and does not persistently make the changes.

Procedure

1. Boot into the GRUB boot menu.
2. Select the kernel you want to start.
3. Press the **e** key to edit the kernel parameters.
4. 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.
5. 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.

6. Edit the kernel parameters as required. For example, to run the system in emergency mode, add the **emergency** parameter at the end of the **linux** line:

```
linux ($root)/vmlinuz-4.18.0-348.12.2.el8_5.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet emergency
```

To enable the system messages, remove the **rhgb** and **quiet** parameters.

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



IMPORTANT

If you press the **Esc** key to leave command line editing, it will drop all the user made changes.

4.6. CONFIGURING GRUB SETTINGS TO ENABLE SERIAL CONSOLE CONNECTION

The serial console is beneficial when you need to connect to a headless server or an embedded system and the network is down. Or when you need to avoid security rules and obtain login access on a different system.

You need to configure some default GRUB settings to use the serial console connection.

Prerequisites

- You have root permissions.

Procedure

1. Add the following two lines to the **/etc/default/grub** file:

```
GRUB_TERMINAL="serial"
GRUB_SERIAL_COMMAND="serial --speed=9600 --unit=0 --word=8 --parity=no --stop=1"
```

The first line disables the graphical terminal. The **GRUB_TERMINAL** key overrides values of **GRUB_TERMINAL_INPUT** and **GRUB_TERMINAL_OUTPUT** keys.

The second line adjusts the baud rate (**--speed**), parity and other values to fit your environment and hardware. Note that a much higher baud rate, for example 115200, is preferable for tasks such as following log files.

2. Update the GRUB configuration file.

- On BIOS-based machines:

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

- On UEFI-based machines:

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

3. Reboot the system for the changes to take effect.

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. 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.



IMPORTANT

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

For more information about tuning kernel on IBM DB2, see [Tuning Red Hat Enterprise Linux for IBM DB2](#).

5.1. WHAT ARE KERNEL PARAMETERS

Kernel parameters are tunable values that you can adjust while the system is running. Note that for changes to take effect, you do not need to reboot the system or recompile the kernel.

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

Tunable class	Subsystem
abi	Execution domains and personalities
crypto	Cryptographic interfaces
debug	Kernel debugging interfaces
dev	Device-specific information
fs	Global and specific file system tunables
kernel	Global kernel tunables
net	Network tunables
sunrpc	Sun Remote Procedure Call (NFS)

Tunable class	Subsystem
user	User Namespace limits
vm	Tuning and management of memory, buffers, and cache

Additional resources

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

5.2. CONFIGURING KERNEL PARAMETERS TEMPORARILY WITH SYSCTL

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. List all parameters and their values.

```
# 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, enter:

```
# 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

- The **sysctl(8)** manual page
- [Using configuration files in /etc/sysctl.d/ to adjust kernel parameters](#)

5.3. CONFIGURING KERNEL PARAMETERS PERMANENTLY WITH SYSCTL

Use the **sysctl** command to permanently set kernel parameters.

Prerequisites

- Root permissions

Procedure

1. List all parameters.

```
# sysctl -a
```

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

2. Configure a parameter permanently:

```
# 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

- The **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

You must modify the configuration files in the **/etc/sysctl.d/** directory manually to permanently set kernel parameters.

Prerequisites

- You have root permissions.

Procedure

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

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

2. Include kernel parameters, one per line:

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

3. Save the configuration file.
4. Reboot the machine for the changes to take effect.
 - Alternatively, apply changes without rebooting:

```
# 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/

Set kernel parameters temporarily through the files in the **/proc/sys/** virtual file system directory.

Prerequisites

- Root permissions

Procedure

1. Identify a kernel parameter you want to configure.

```
# 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.

```
# echo <TARGET_VALUE> > /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

The configuration changes applied by using a command are not permanent and will disappear once the system is restarted.

Verification

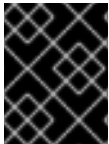
1. Verify the value of the newly set kernel parameter.

```
# cat /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

CHAPTER 6. MAKING TEMPORARY CHANGES TO THE GRUB MENU

You can modify GRUB menu entries or pass arguments to the kernel, which applies only to the current boot. On a selected menu entry in the boot loader menu, you can:

- Display the menu entry editor interface by pressing the **e** key.
- Discard any changes and reload the standard menu interface by pressing the **Esc** key.
- Load the command line by pressing the **c** key.
- Type any relevant GRUB commands and enter them by pressing the **Enter** key.
- Complete a command based on context by pressing the **Tab** key.
- Move to the beginning of a line by pressing the **Ctrl+a** key combination.
- Move to the end of a line by pressing the **Ctrl+e** key combination.



IMPORTANT

The following procedures provide instruction on making changes to a GRUB Menu during a single boot process.

6.1. INTRODUCTION TO GRUB

You can use the GNU GRand Unified Bootloader (GRUB) to perform the following actions:

- Select an operating system or kernel to load at system boot time.
- Pass arguments to the kernel.

When booting with GRUB, you can use either a menu interface or a command-line interface, the **GRUB command shell**. When you start the system, the menu interface is displayed.

```
Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)
Red Hat Enterprise Linux (0-rescue-67db13ba8cbb420794ef3ee0a8313205) 8.6→
```

```
Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

You can switch to the command line by pressing the **c** key.

```
Minimal BASH-like line editing is supported. For the first word,
TAB lists possible command completions. Anywhere else TAB lists
possible device or file completions. ESC at any time exits.
```

```
grub>
```

You can return to the menu interface by typing `exit` and pressing the Enter key.

GRUB BLS files

The boot loader menu entries are defined as Boot Loader Specification (**BLS**) files. This file format manages boot loader configuration for each boot option in a drop-in directory, without manipulating boot loader configuration files. The **grubby** utility can edit these **BLS** files.

GRUB configuration file

The `/boot/grub2/grub.cfg` configuration file does not define the menu entries.

Additional resources

- [Boot menu](#)
- [Introduction to boot loader specification](#)

6.2. INTRODUCTION TO BOOT LOADER SPECIFICATION

The BootLoader Specification (BLS) defines a scheme and the file format to manage the boot loader configuration for each boot option in the drop-in directory without the need to manipulate the boot loader configuration files. Unlike earlier approaches, each boot entry is now represented by a separate configuration file in the drop-in directory. The drop-in directory extends its configuration without having the need to edit or regenerate the configuration files. The BLS extends this concept for the boot menu entries.

Using BLS, you can manage the boot loader menu options by adding, removing, or editing individual boot entry files in a directory. This makes the kernel installation process significantly simpler and consistent across the different architectures.

The **grubby** tool is a thin wrapper script around the BLS and it supports the same **grubby** arguments and options. It runs the **dracut** utility to create an initial ramdisk image. With this setup, the core boot loader configuration files are static and are not modified after kernel installation.

This premise is particularly relevant in RHEL 8 because not all architectures use the same boot loader:

- **x86_64**, **aarch64**, and **ppc64le** with open firmware use **GRUB2**.
- **ppc64le** with Open Power Abstraction Layer (OPAL) uses **Petitboot**.
- **s390x** uses **zipl**.

Additional Resources

- [Section 4.2, “Understanding boot entries”](#)
- the **grubby(8)** manual page

6.3. BOOTING TO RESCUE MODE

Rescue mode provides a convenient single-user environment in which you can repair your system in situations when it is unable to complete a normal booting process. In rescue mode, the system attempts to mount all local file systems and start some important system services. However, it does not activate network interfaces or allow more users to be logged into the system at the same time.

Procedure

1. On the GRUB boot screen, press the **e** key for edit.
2. Add the following parameter at the end of the **linux** line:

```
systemd.unit=rescue.target
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet zswap.enabled=0 systemd.unit=rescue.target
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl+x** to boot to rescue mode.

```
You are in rescue mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue):
```

6.4. BOOTING TO EMERGENCY MODE

Emergency mode provides the most minimal environment possible in which you can repair your system even in situations when the system is unable to enter rescue mode.

In emergency mode, the system:

- mounts the **root** file system only for reading
- starts a few essential services

However, the system **does not**:

- attempt to mount any other local file systems
- activate network interfaces

Procedure

1. On the GRUB boot screen, press the **e** key for edit.
2. Add the following parameter at the end of the **linux** line:

```
systemd.unit=emergency.target
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet zswap.enabled=0 systemd.unit=emergency.target
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl+x** to boot to emergency mode.

```
You are in emergency mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue): _
```

6.5. BOOTING TO THE DEBUG SHELL

The **systemd** debug shell provides a shell prompt very early in the start-up process. Once in the debug shell, you can use the **systemctl** commands, such as **systemctl list-jobs** and **systemctl list-units**, to search for the cause of **systemd** related boot-up problems.

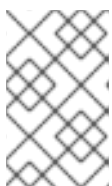
Procedure

1. On the GRUB boot screen, press the **e** key for edit.
2. Add the following parameter at the end of the **linux** line:

```
systemd.debug-shell
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet systemd.debug-shell
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Optional: Add the **debug** option.



NOTE

Adding the **debug** option to the kernel command line increases the number of log messages. For **systemd**, the kernel command-line option **debug** is now a shortcut for **systemd.log_level=debug**.

4. Press **Ctrl+x** to boot to the debug shell.

**WARNING**

Permanently enabling the debug shell is a security risk because there is no authentication required to use it. It is recommended to disable it when the debugging session has ended.

6.6. CONNECTING TO THE DEBUG SHELL

During the boot process, the **systemd-debug-generator** configures the debug shell on TTY9.

Prerequisites

- You have booted to the debug shell successfully. See [Booting to the debug shell](#).

Procedure

- Press **Ctrl+Alt+F9** to connect to the debug shell.
If you work with a virtual machine, sending this key combination requires support from the virtualization application. For example, if you use **Virtual Machine Manager**, select **Send Key** → **Ctrl+Alt+F9** from the menu.
- The debug shell does not require authentication, therefore you can see a prompt similar to the following on TTY9:

```
sh-4.4#
```

Verification

- Enter a command as follows:

```
sh-4.4# systemctl status $$
```

```
sh-4.4# systemctl status $$
■ debug-shell.service - Early root shell on /dev/tty9 FOR DEBUGGING ONLY
   Loaded: loaded (/usr/lib/systemd/system/debug-shell.service; enabled-runtime; vendor preset: disabled)
   Active: active (running) since Tue 2022-08-02 08:40:09 EDT; 45s ago
     Docs: man:systemd-debug-generator(8)
    Main PID: 735 (sh)
      Tasks: 3 (limit: 17257)
     Memory: 2.3M
    CGroup: /system.slice/debug-shell.service
            └─ 735 /bin/sh
               └─ 2092 systemctl status 735
                  └─ 2093 less

sh-4.4#
```

- To return to the default shell, if the boot succeeded, press **Ctrl+Alt+F1**.

Additional resources

- The **systemd-debug-generator(8)** manual page

6.7. RESETTING THE ROOT PASSWORD USING AN INSTALLATION DISK

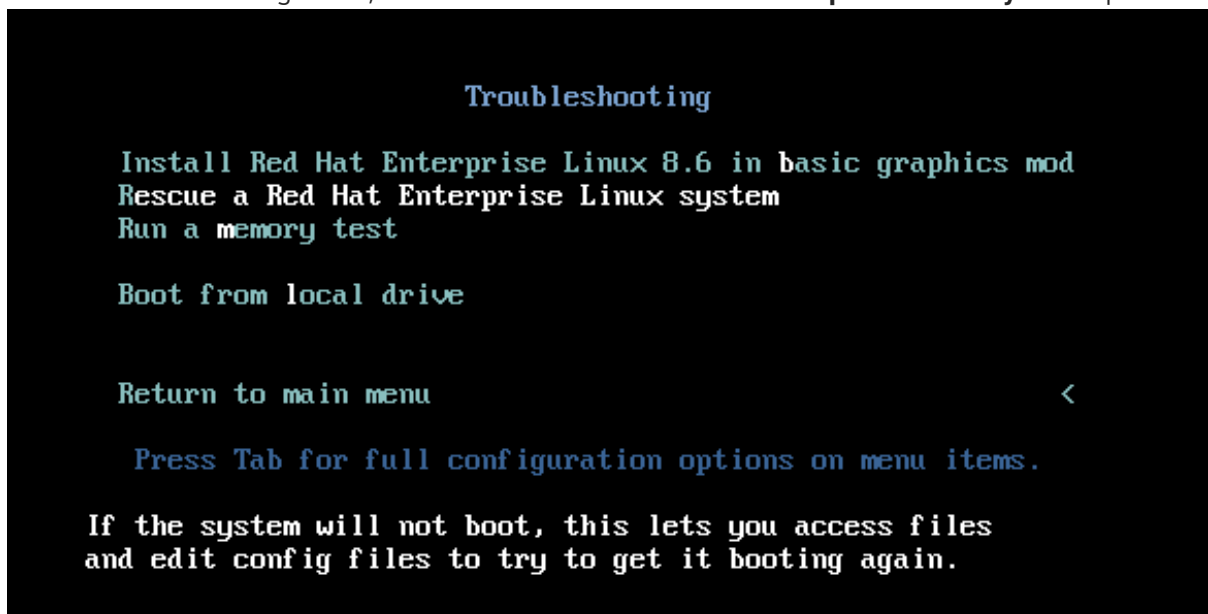
In case you forget or lose the **root** password, you can reset it.

Procedure

1. Boot the host from an installation source.
2. In the boot menu for the installation media, select the **Troubleshooting** option.



3. In the Troubleshooting menu, select the **Rescue a Red Hat Enterprise Linux system** option.



4. At the Rescue menu, select **1** and press the **Enter** key to continue.

```
=====
Rescue

The rescue environment will now attempt to find your Linux installation and
mount it under the directory : /mnt/sysroot. You can then make any changes
required to your system. Choose '1' to proceed with this step.
You can choose to mount your file systems read-only instead of read-write by
choosing '2'.
If for some reason this process does not work choose '3' to skip directly to a
shell.

1) Continue
2) Read-only mount
3) Skip to shell
4) Quit (Reboot)

Please make a selection from the above: 1_
```

5. Change the file system **root** as follows:

```
sh-4.4# chroot /mnt/sysimage
```

```
When finished, please exit from the shell and your system will reboot.
Please press ENTER to get a shell:
sh-4.4# chroot /mnt/sysimage
bash-4.4#

[anaconda11:main* 2:shell 3:log 4:storage-log 5:program-log]
```

6. Enter the **passwd** command and follow the instructions displayed on the command line to change the **root** password.

```
bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#

[anaconda11:main* 2:shell 3:log 4:storage-log 5:program-log]
```

7. Remove the **autorelabel** file to prevent a time consuming SELinux relabel of the disk:

```
sh-4.4# rm -f /.autorelabel
```

8. Enter the **exit** command to exit the **chroot** environment.
9. Enter the **exit** command again to resume the initialization and finish the system boot.

6.8. RESETTING THE ROOT PASSWORD USING RD.BREAK

In case you forget or lose the **root** password, you can reset it.

Procedure

1. Start the system and, on the GRUB boot screen, press the **e** key for edit.
2. Add the **rd.break** parameter at the end of the **linux** line:

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet rd.break
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl+x** to boot the system with the changed parameters.

```
Generating "/run/initramfs/rdsosreport.txt"

Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.

switch_root:/# _
```

4. Remount the file system as writable.

```
switch_root:/# mount -o remount,rw /sysroot
```

5. Change the file system's **root**.

```
switch_root:/# chroot /sysroot
```

6. Enter the **passwd** command and follow the instructions displayed on the command line.

```
bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#

[anaconda11:main* 2:shell 3:log 4:storage-log 5:program-log
```

7. Relabel all files on the next system boot.

```
sh-4.4# touch /.autorelabel
```

8. Remount the file system as **read only**:

```
sh-4.4# mount -o remount,ro /
```

9. Enter the **exit** command to exit the **chroot** environment.
10. Enter the **exit** command again to resume the initialization and finish the system boot.



NOTE

The SELinux relabeling process can take a long time. A system reboot occurs automatically when the process is complete.

TIP

You can omit the time consuming SELinux relabeling process by adding the **enforcing=0** option.

Procedure

1. When adding the **rd.break** parameter at the end of the **linux** line, append **enforcing=0** as well.

```
rd.break enforcing=0
```

2. Restore the **/etc/shadow** file's SELinux security context.

```
# restorecon /etc/shadow
```

3. Turn SELinux policy enforcement back on and verify that it is on.

```
# setenforce 1
# getenforce
Enforcing
```

Note that if you added the **enforcing=0** option in step 3 you can omit entering the **touch /.autorelabel** command in step 8.

6.9. ADDITIONAL RESOURCES

- The **/usr/share/doc/grub2-common** directory.
- The **info grub2** command.

CHAPTER 7. MAKING PERSISTENT CHANGES TO THE GRUB BOOT LOADER

Use the **grubby** tool to make persistent changes in GRUB.

7.1. PREREQUISITES

- You have successfully installed RHEL on your system.
- You have root permission.

7.2. LISTING THE DEFAULT KERNEL

By listing the default kernel, you can find the file name and the index number of the default kernel to make permanent changes to the GRUB boot loader.

Procedure

- To get the file name of the default kernel, enter:

```
# grubby --default-kernel
/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

- To get the index number of the default kernel, enter:

```
# grubby --default-index
0
```

7.3. VIEWING THE GRUB MENU ENTRY FOR A KERNEL

You can list all the kernel menu entries or view the GRUB menu entry for a specific kernel.

Procedure

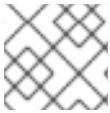
- To list all kernel menu entries, enter:

```
# grubby --info=ALL
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
index=1
kernel="/boot/vmlinuz-0-rescue-67db13ba8cdb420794ef3ee0a8313205"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-0-rescue-67db13ba8cdb420794ef3ee0a8313205.img"
```

```
title="Red Hat Enterprise Linux (0-rescue-67db13ba8cdb420794ef3ee0a8313205) 8.6
(Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-0-rescue"
```

- To view the GRUB menu entry for a specific kernel, enter:

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```



NOTE

Try tab completion to see available kernels within the **/boot** directory.

7.4. EDITING A KERNEL ARGUMENT

You can change a value in an existing kernel argument. For example, you can change the virtual console (screen) font and size.

Procedure

- Change the virtual console font to **latarcyrheb-sun** with the size of **32**:

```
# grubby --args=vconsole.font=latarcyrheb-sun32 --update-kernel /boot/vmlinuz-
4.18.0-372.9.1.el8.x86_64
```

7.5. ADDING AND REMOVING ARGUMENTS FROM A GRUB MENU ENTRY

You can add, remove, or simultaneously add and remove arguments from the GRUB Menu.

Procedure

- To add arguments to a GRUB menu entry, use the **--update-kernel** option in combination with **-args**. For example, following command adds a serial console:

```
# grubby --args=console=ttyS0,115200 --update-kernel /boot/vmlinuz-4.18.0-
372.9.1.el8.x86_64
```

The console arguments are attached to the end of the line, the new console will take precedence over any other configured consoles.

- To remove arguments from a GRUB menu entry, use the **--update-kernel** option in combination with **--remove-args**. For example:

```
# grubby --remove-args="rhgb quiet" --update-kernel /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

This command removes the Red Hat graphical boot argument and enables log messages, that is verbose mode.

- To add and remove arguments simultaneously, enter:

```
# grubby --remove-args="rhgb quiet" --args=console=ttyS0,115200 --update-kernel /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

Verification

- To review the permanent changes you have made, enter:

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params zswap.enabled=1 console=ttyS0,115200"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```

7.6. ADDING A NEW BOOT ENTRY

You can add a new boot entry to the boot loader menu entries.

Procedure

1. Copy all the kernel arguments from your default kernel to this new kernel entry:

```
# grubby --add-kernel=new_kernel --title="entry_title" --initrd="new_initrd" --copy-default
```

2. Get the list of available boot entries:

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r-- 1 root root 336 Aug 15 15:12
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

3. Create a new boot entry. For example, for the *4.18.0-193.el8.x86_64* kernel, issue the command as follows:

```
# grubby --grub2 --add-kernel=/boot/vmlinuz-4.18.0-193.el8.x86_64 --title="Red Hat Enterprise 8 Test" --initrd=/boot/initramfs-4.18.0-193.el8.x86_64.img --copy-default
```

Verification

verification

- Verify that the newly added boot entry is listed among the available boot entries:

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r-- 1 root root 287 Aug 16 15:17
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-
193.el8.x86_64.0~custom.conf
-rw-r--r-- 1 root root 287 Aug 16 15:29
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

7.7. CHANGING THE DEFAULT BOOT ENTRY WITH GRUBBY

With the **grubby** tool, you can change the default boot entry.

Procedure

- To make a persistent change in the kernel designated as the default kernel, enter:

```
# grubby --set-default /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
The default is /boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-
372.9.1.el8.x86_64.conf with index 0 and kernel /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

7.8. UPDATING ALL KERNEL MENUS WITH THE SAME ARGUMENTS

You can add the same kernel boot arguments to all the kernel menu entries.

Procedure

- To add the same kernel boot arguments to all the kernel menu entries, attach the **--update-kernel=ALL** parameter. For example, this command adds a serial console to all kernels:

```
# grubby --update-kernel=ALL --args=console=ttyS0,115200
```

**NOTE**

The **--update-kernel** parameter also accepts **DEFAULT** or a comma-separated list of kernel index numbers.

7.9. CHANGING DEFAULT KERNEL OPTIONS FOR CURRENT AND FUTURE KERNELS

By using the **kernelopts** variable, you can change the default kernel options for both current and future kernels.

Procedure

1. List the kernel parameters from the **kernelopts** variable:


```
# grub2-editenv - list | grep kernelopts
```

```
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap  
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
```

2. Make the changes to the kernel command-line parameters. You can add, remove or modify a parameter. For example, to add the **debug** parameter, enter:

```
# grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) <debug>"
```

3. Optional: Verify the parameter newly added to **kernelopts**:

```
# grub2-editenv - list | grep kernelopts
```

```
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap  
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet debug
```

4. Reboot the system for the changes to take effect.



NOTE

As an alternative, you can use the **grubby** command to pass the arguments to current and future kernels:

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

7.10. ADDITIONAL RESOURCES

- The **/usr/share/doc/grub2-common** directory.
- The **info grub2** command.

CHAPTER 8. BUILDING A CUSTOMIZED BOOT MENU

You can build a boot menu containing specific entries or change the order of the entries. For such a task, you can use GRUB, **grubby**, and Boot Loader Specification (**BLS**) files.

The following sections provide information about using GRUB and **grubby** to do basic customization of the boot menu.

8.1. THE GRUB CONFIGURATION FILE

Learn about the boot loader configuration file that is **/boot/grub2/grub.cfg** on BIOS-based machines and **/boot/efi/EFI/redhat/grub.cfg** on UEFI-based machines.

GRUB scripts search the user's computer and build a boot menu based on what operating systems the scripts find. To reflect the latest system boot options, the boot menu is rebuilt automatically when the kernel is updated or a new kernel is added.

GRUB uses a series of scripts, located in the **/etc/grub.d/** directory, to build the menu. The scripts include the following files:

- **00_header**, which loads GRUB settings from the **/etc/default/grub** file.
- **01_users**, which reads the root password from the **user.cfg** file.
- **10_linux**, which locates kernels in the default partition of Red Hat Enterprise Linux.
- **30_os-prober**, which builds entries for operating systems found on other partitions.
- **40_custom**, a template used to create additional menu entries.

GRUB reads scripts from the **/etc/grub.d/** directory in alphabetical order and therefore you can rename them to change the boot order of specific menu entries.

8.2. HIDING THE LIST OF BOOTABLE KERNELS

You can prevent GRUB from displaying the list of bootable kernels when the system starts up.

Procedure

1. Set the **GRUB_TIMEOUT_STYLE** option in the **/etc/default/grub** file as follows:

```
GRUB_TIMEOUT_STYLE=hidden
```

2. Rebuild the **grub.cfg** file for the changes to take effect.

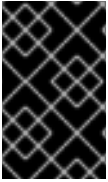
- On BIOS-based machines, enter:

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

- On UEFI-based machines, enter:

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

3. Press the **Esc** key to display the list of bootable kernels when booting.



IMPORTANT

Do not set **GRUB_TIMEOUT** to 0 in the `/etc/default/grub` file to hide the list of bootable kernels. With such a setting, the system always boots immediately on the default menu entry, and if the default kernel fails to boot, it is not possible to boot any previous kernel.

8.3. CHANGING THE DEFAULT BOOT ENTRY WITH THE GRUB CONFIGURATION FILE

You can specify the default kernel package type, and change the default boot entry.

Procedure

1. Specify which operating system or kernel must be loaded by default by passing its index to the **grub2-set-default** command, for example:

```
# grubby --set-default-index=1
The default is /boot/loader/entries/d5151aa93c444ac89e78347a1504d6c6-4.18.0-348.el8.x86_64.conf with index 1 and kernel /boot/vmlinuz-4.18.0-348.el8.x86_64
```

GRUB supports using a numeric value as the key for the **saved_entry** directive in `/boot/grub2/grubenv` to change the default order in which the operating systems are loaded.



NOTE

Index counting starts with zero. Therefore, GRUB loads the second entry. With the next installed kernel, the index value will be overwritten.



NOTE

You can also use **grubby** to find indices for kernels. For more information, see [Viewing the GRUB Menu Entry for a Kernel](#).

2. Optional: Force the system to always use a particular menu entry:

- a. List the available menu entries:

```
# grubby --info=ALL
```

- b. Use the menu entry name or the number of the position of a menu entry in the list as the key to the **GRUB_DEFAULT** directive in the `/etc/default/grub` file. For example:

```
GRUB_DEFAULT=example-gnu-linux
```

3. Rebuild the **grub.cfg** file for the changes to take effect.

- On BIOS-based machines, enter:

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

- On UEFI-based machines:

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

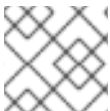
■

CHAPTER 9. REINSTALLING GRUB

You can reinstall the GRUB boot loader to fix certain problems, usually caused by an incorrect installation of GRUB, missing files, or a broken system. You can resolve these issues by restoring the missing files and updating the boot information.

Reasons to reinstall GRUB:

- Upgrading the GRUB boot loader packages.
- Adding the boot information to another drive.
- The user requires the GRUB boot loader to control installed operating systems. However, some operating systems are installed with their own boot loaders and reinstalling GRUB returns control to the desired operating system.

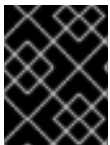


NOTE

GRUB restores files only if they are not corrupted.

9.1. REINSTALLING GRUB ON BIOS-BASED MACHINES

You can reinstall the GRUB boot loader on your BIOS-based system. Always reinstall GRUB after updating the GRUB packages.



IMPORTANT

This overwrites the existing GRUB to install the new GRUB. Ensure that the system does not cause data corruption or boot crash during the installation.

Procedure

1. Reinstall GRUB on the device where it is installed. For example, if **sda** is your device:

```
# grub2-install /dev/sda
```

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

```
# reboot
```

Additional resources

- **grub-install(1)** man page on your system

9.2. REINSTALLING GRUB ON UEFI-BASED MACHINES

You can reinstall the GRUB boot loader on your UEFI-based system.



IMPORTANT

Ensure that the system does not cause data corruption or boot crash during the installation.

Procedure

1. Reinstall the **grub2-efi** and **shim** boot loader files:

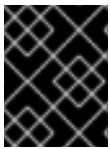
```
# yum reinstall grub2-efi shim
```

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

```
# reboot
```

9.3. REINSTALLING GRUB ON IBM POWER MACHINES

You can reinstall the GRUB boot loader on the Power PC Reference Platform (PReP) boot partition of your IBM Power system. Always reinstall GRUB after updating the GRUB packages.



IMPORTANT

This overwrites the existing GRUB to install the new GRUB. Ensure that the system does not cause data corruption or boot crash during the installation.

Procedure

1. Determine the disk partition that stores GRUB:

```
# bootlist -m normal -o  
sda1
```

2. Reinstall GRUB on the disk partition:

```
# grub2-install partition
```

Replace ***partition*** with the identified GRUB partition, such as **/dev/sda1**.

3. Reboot your system for the changes to take effect:

```
# reboot
```

Additional resources

- **grub-install(1)** man page on your system

9.4. RESETTING GRUB

Resetting GRUB completely removes all GRUB configuration files and system settings, and reinstalls the boot loader. You can reset all the configuration settings to their default values, and therefore fix failures caused by corrupted files and invalid configuration.



IMPORTANT

The following procedure will remove all the customization made by the user.

Procedure

1. Remove the configuration files:

```
# rm /etc/grub.d/*  
# rm /etc/sysconfig/grub
```

2. Reinstall packages.

- On BIOS-based machines:

```
# yum reinstall grub2-tools
```

- On UEFI-based machines:

```
# yum reinstall grub2-efi shim grub2-tools grub2-common
```

3. Rebuild the **grub.cfg** file for the changes to take effect.

- On BIOS-based machines:

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

- On UEFI-based machines:

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

4. Follow [Reinstalling GRUB](#) procedure to restore GRUB on the **/boot/** partition.

CHAPTER 10. PROTECTING GRUB WITH A PASSWORD

You can protect GRUB with a password in two ways:

- Password is required for modifying menu entries but not for booting existing menu entries.
- Password is required for modifying menu entries and for booting existing menu entries.

10.1. SETTING PASSWORD PROTECTION ONLY FOR MODIFYING MENU ENTRIES

You can configure GRUB to support password authentication for modifying GRUB menu entries. This procedure creates a **/boot/grub2/user.cfg** file that provides the password in the hash format.



IMPORTANT

Setting a password using the **grub2-setpassword** command prevents menu entries from unauthorized modification but not from unauthorized booting.

Procedure

1. Issue the **grub2-setpassword** command as root:

```
# grub2-setpassword
```

2. Enter the password for the user and press the **Enter** key to confirm the password:

```
Enter password:  
Confirm the password:
```



NOTE

The root user is defined in the **/boot/grub2/grub.cfg** file with the password changes. Therefore, modifying a boot entry during booting requires the name and password of the root user.

10.2. SETTING PASSWORD PROTECTION FOR MODIFYING AND BOOTING MENU ENTRIES

You can configure GRUB to prevent menu entries from unauthorized modification and booting.



WARNING

If you forget the GRUB password, you will not be able to boot the entries you have reconfigured.

Procedure

1. Open the Boot Loader Specification (**BLS**) file for boot entry you want to modify from the **/boot/loader/entries/** directory.
2. Find the line beginning with **grub_users**. This parameter passes extra arguments to **menuentry**.
3. Set the **grub_users** attribute to the user name that is allowed to boot the entry besides the superusers, by default this user is **root**.

Here is a sample configuration file:

+

```
title Red Hat Enterprise Linux (4.18.0-221.el8.x86_64) 8.3
(Ootpa)
version 4.18.0-221.el8.x86_64
linux /vmlinuz-4.18.0-221.el8.x86_64
initrd /initramfs-4.18.0-221.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20200625210904-4.18.0-221.el8.x86_64
grub_users root
grub_arg --unrestricted
grub_class kernel
```

1. Save and close the **BLS** file:



NOTE

If you want to protect all the menu entries from booting, you can directly set the **grub_users** attribute. For example, if root is the user:

```
# grub2-editenv - set grub_users="root"
```

CHAPTER 11. KEEPING KERNEL PANIC PARAMETERS DISABLED IN VIRTUALIZED ENVIRONMENTS

When configuring a Virtual Machine in RHEL 8, do not enable the **softlockup_panic** and **nmi_watchdog** kernel parameters, because the Virtual Machine might suffer from a spurious soft lockup. And that should not require a kernel panic.

Find the reasons behind this advice in the following sections.

11.1. WHAT IS A SOFT LOCKUP

A soft lockup is a situation usually caused by a bug, when a task is executing in kernel space on a CPU without rescheduling. The task also does not allow any other task to execute on that particular CPU. As a result, a warning is displayed to a user through the system console. This problem is also referred to as the soft lockup firing.

Additional resources

- [What is a CPU soft lockup?](#)

11.2. PARAMETERS CONTROLLING KERNEL PANIC

The following kernel parameters can be set to control a system's behavior when a soft lockup is detected.

softlockup_panic

Controls whether or not the kernel will panic when a soft lockup is detected.

Type	Value	Effect
Integer	0	kernel does not panic on soft lockup
Integer	1	kernel panics on soft lockup

By default, on RHEL 8, this value is 0.

The system needs to detect a hard lockup first to be able to panic. The detection is controlled by the **nmi_watchdog** parameter.

nmi_watchdog

Controls whether lockup detection mechanisms (**watchdogs**) are active or not. This parameter is of integer type.

Value	Effect
0	disables lockup detector
1	enables lockup detector

The hard lockup detector monitors each CPU for its ability to respond to interrupts.

watchdog_thresh

Controls frequency of watchdog **hrtimer**, NMI events, and soft or hard lockup thresholds.

Default threshold	Soft lockup threshold
10 seconds	2 * watchdog_thresh

Setting this parameter to zero disables lockup detection altogether.

Additional resources

- [Softlockup detector and hardlockup detector](#)
- [Kernel sysctl](#)

11.3. SPURIOUS SOFT LOCKUPS IN VIRTUALIZED ENVIRONMENTS

The [soft lockup](#) firing on physical hosts usually represents a kernel or a hardware bug. The same phenomenon happening on guest operating systems in virtualized environments might represent a false warning.

Heavy workload on a host or high contention over some specific resource, such as memory, can cause a spurious soft lockup firing because the host might schedule out the guest CPU for a period longer than 20 seconds. When the guest CPU is again scheduled to run on the host, it experiences a *time jump* that triggers the due timers. The timers also include the **hrtimer** watchdog that can report a soft lockup on the guest CPU.

Soft lockup in a virtualized environment can be false. You must not enable the kernel parameters that trigger a system panic when a soft lockup reports to a guest CPU.

**IMPORTANT**

To understand soft lockups in guests, it is essential to know that the host schedules the guest as a task, and the guest then schedules its own tasks.

Additional resources

- [Virtual machine components and their interaction](#)

CHAPTER 12. ADJUSTING KERNEL PARAMETERS FOR DATABASE SERVERS

To ensure efficient operation of database servers and databases, you must configure the required sets of kernel parameters.

12.1. INTRODUCTION TO DATABASE SERVERS

A database server is a service that provides features of a database management system (DBMS). DBMS provides utilities for database administration and interacts with end users, applications, and databases.

Red Hat Enterprise Linux 8 provides the following database management systems:

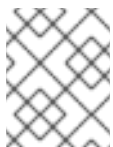
- **MariaDB 10.3**
- **MariaDB 10.5** - available since RHEL 8.4
- **MariaDB 10.11** - available since RHEL 8.10
- **MySQL 8.0**
- **PostgreSQL 10**
- **PostgreSQL 9.6**
- **PostgreSQL 12** - available since RHEL 8.1.1
- **PostgreSQL 13** - available since RHEL 8.4
- **PostgreSQL 15** - available since RHEL 8.8
- **PostgreSQL 16** - available since RHEL 8.10

12.2. PARAMETERS AFFECTING PERFORMANCE OF DATABASE APPLICATIONS

The following kernel parameters affect performance of database applications.

fs.aio-max-nr

Defines the maximum number of asynchronous I/O operations the system can handle on the server.



NOTE

Raising the **fs.aio-max-nr** parameter produces no additional changes beyond increasing the aio limit.

fs.file-max

Defines the maximum number of file handles (temporary file names or IDs assigned to open files) the system supports at any instance.

The kernel dynamically allocates file handles whenever a file handle is requested by an application. However, the kernel does not free these file handles when they are released by the application. It recycles these file handles instead. The total number of allocated file handles will increase over time even though the number of currently used file handles might be low.

kernel.shmall

Defines the total number of shared memory pages that can be used system-wide. To use the entire main memory, the value of the **kernel.shmall** parameter should be \leq total main memory size.

kernel.shmmax

Defines the maximum size in bytes of a single shared memory segment that a Linux process can allocate in its virtual address space.

kernel.shmmni

Defines the maximum number of shared memory segments the database server is able to handle.

net.ipv4.ip_local_port_range

The system uses this port range for programs that connect to a database server without specifying a port number.

net.core.rmem_default

Defines the default receive socket memory through Transmission Control Protocol (TCP).

net.core.rmem_max

Defines the maximum receive socket memory through Transmission Control Protocol (TCP).

net.core.wmem_default

Defines the default send socket memory through Transmission Control Protocol (TCP).

net.core.wmem_max

Defines the maximum send socket memory through Transmission Control Protocol (TCP).

vm.dirty_bytes / vm.dirty_ratio

Defines a threshold in bytes / in percentage of dirty-able memory at which a process generating dirty data is started in the **write()** function.

**NOTE**

Either **vm.dirty_bytes** or **vm.dirty_ratio** can be specified at a time.

vm.dirty_background_bytes / vm.dirty_background_ratio

Defines a threshold in bytes / in percentage of dirty-able memory at which the kernel tries to actively write dirty data to hard-disk.

**NOTE**

Either **vm.dirty_background_bytes** or **vm.dirty_background_ratio** can be specified at a time.

vm.dirty_writeback_centisecs

Defines a time interval between periodic wake-ups of the kernel threads responsible for writing dirty data to hard-disk.

This kernel parameters measures in 100th's of a second.

vm.dirty_expire_centisecs

Defines the time of dirty data that becomes old to be written to hard-disk.

This kernel parameters measures in 100th's of a second.

Additional resources

- [*Dirty pagecache writeback and vm.dirty parameters*](#)

CHAPTER 13. GETTING STARTED WITH KERNEL LOGGING

Log files provide messages about the system, including the kernel, services, and applications running on it. The logging system in Red Hat Enterprise Linux is provided by the built-in **syslog** protocol. Various utilities use this system to record events and organize them into log files. These files are useful when auditing the operating system or troubleshooting problems.

13.1. WHAT IS THE KERNEL RING BUFFER

During the boot process, the console provides important information about the initial phase of the system startup. To avoid loss of the early messages the kernel utilizes a ring buffer. This buffer stores all messages, including boot messages, generated by the **printk()** function within the kernel code. The messages from the kernel ring buffer are then read and stored in log files on permanent storage, for example, by the **syslog** service.

The ring buffer is a cyclic data structure that has a fixed size, and is hard-coded into the kernel. Users can display data stored in the kernel ring buffer through the **dmesg** command or the **/var/log/boot.log** file. When the ring buffer is full, the new data overwrites the old.

Additional resources

- **syslog(2)** and **dmesg(1)** manual page

13.2. ROLE OF PRINTK ON LOG-LEVELS AND KERNEL LOGGING

Each message the kernel reports has a log-level associated with it that defines the importance of the message. The kernel ring buffer, as described in [What is the kernel ring buffer](#), collects kernel messages of all log-levels. It is the **kernel.printk** parameter that defines what messages from the buffer are printed to the console.

The log-level values break down in this order:

- | | |
|---|--|
| 0 | Kernel emergency. The system is unusable. |
| 1 | Kernel alert. Action must be taken immediately. |
| 2 | Condition of the kernel is considered critical. |
| 3 | General kernel error condition. |
| 4 | General kernel warning condition. |
| 5 | Kernel notice of a normal but significant condition. |
| 6 | Kernel informational message. |
| 7 | Kernel debug-level messages. |

By default, **kernel.printk** in RHEL 8 has the following values:

```
# sysctl kernel.printk
kernel.printk = 7 4 1 7
```

The four values define the following, in order:

1. Console log-level, defines the lowest priority of messages printed to the console.
2. Default log-level for messages without an explicit log-level attached to them.
3. Sets the lowest possible log-level configuration for the console log-level.
4. Sets default value for the console log-level at boot time.

Each of these values defines a different rule for handling error messages.



IMPORTANT

The default **7 4 1 7 printk** value allows for better debugging of kernel activity. However, when coupled with a serial console, this **printk** setting might cause intense I/O bursts that might lead to a RHEL system becoming temporarily unresponsive. To avoid these situations, setting a **printk** value of **4 4 1 7** typically works, but at the expense of losing the extra debugging information.

Also note that certain kernel command line parameters, such as **quiet** or **debug**, change the default **kernel.printk** values.

Additional resources

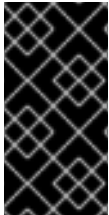
- **syslog(2)** manual page

CHAPTER 14. INSTALLING KDUMP

The **kdump** service is installed and activated by default on the new versions of RHEL 8 installations.

14.1. WHAT IS KDUMP

kdump is a service that provides a crash dumping mechanism and generates a crash dump or a **vmcore** dump file. **vmcore** includes the contents of the system memory for analysis and troubleshooting. **kdump** uses the **kexec** system call to boot into the second kernel, *capture kernel*, without a reboot. This kernel captures the contents of the crashed kernel's memory and saves it into a file. The second kernel is available in a reserved part of the system memory.



IMPORTANT

A kernel crash dump can be the only information available if a system failure occur. Therefore, operational **kdump** is important in mission-critical environments. Red Hat advises to regularly update and test **kexec-tools** in your normal kernel update cycle. This is important when you install new kernel features.

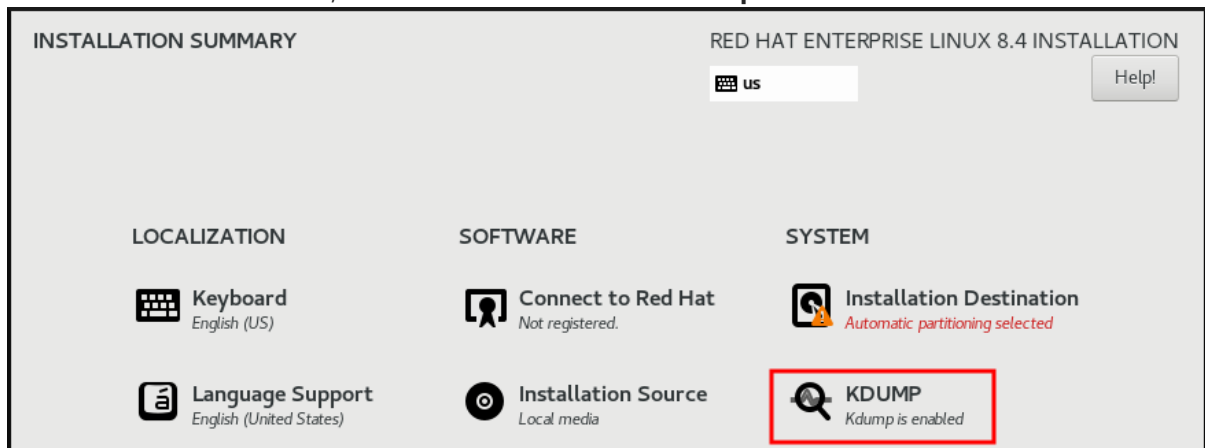
If you have multiple kernels on a machine, you can enable **kdump** for all installed kernels or for specified kernels only. When you install **kdump**, the system creates a default **/etc/kdump.conf** file. **/etc/kdump.conf** includes the default minimum **kdump** configuration, which you can edit to customize the **kdump** configuration.

14.2. INSTALLING KDUMP USING ANACONDA

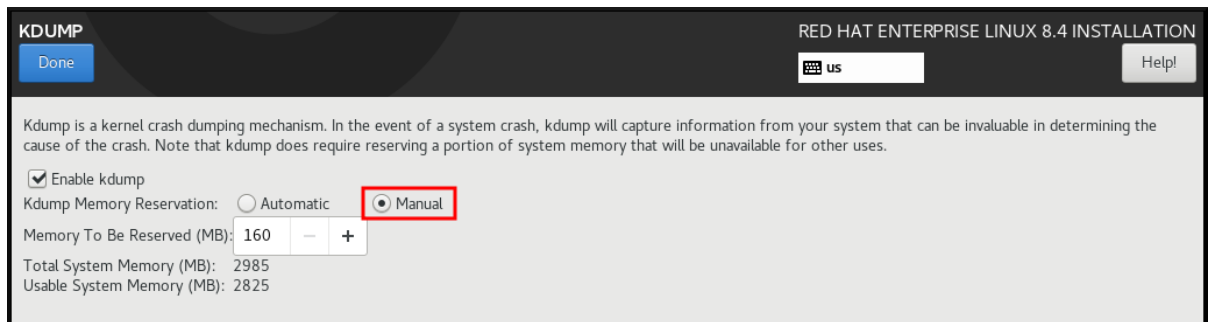
The **Anaconda** installer provides a graphical interface screen for **kdump** configuration during an interactive installation. You can enable **kdump** and reserve the required amount of memory.

Procedure

1. On the Anaconda installer, click **KDUMP** and enable **kdump**:



2. In **Kdump Memory Reservation**, select **Manual** if you must customize the memory reserve.
3. In **KDUMP > Memory To Be Reserved (MB)**, set the required memory reserve for **kdump**.



14.3. INSTALLING KDUMP ON THE COMMAND LINE

Installation options such as custom **Kickstart** installations, in some cases does **not** install or enable **kdump** by default. The following procedure helps you enable **kdump** in this case.

Prerequisites

- An active RHEL subscription.
- A repository containing the **kexec-tools** package for your system CPU architecture.
- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).

Procedure

1. Check if **kdump** is installed on your system:

```
# rpm -q kexec-tools
```

Output if the package is installed:

```
kexec-tools-2.0.17-11.el8.x86_64
```

Output if the package is not installed:

```
package kexec-tools is not installed
```

2. Install **kdump** and other necessary packages:

```
# dnf install kexec-tools
```



IMPORTANT

From **kernel-3.10.0-693.el7** onwards, the **Intel IOMMU** driver is supported for **kdump**. For **kernel-3.10.0-514[.XYZ].el7** and early versions, you must ensure that **Intel IOMMU** is disabled to prevent an unresponsive capture kernel.

CHAPTER 15. CONFIGURING KDUMP ON THE COMMAND LINE

The memory for **kdump** is reserved during the system boot. You can configure the memory size in the system's Grand Unified Bootloader (GRUB) configuration file. The memory size depends on the **crashkernel=** value specified in the configuration file and the size of the physical memory of system.

15.1. ESTIMATING THE KDUMP SIZE

When planning and building your **kdump** environment, it is important to know the space required by the crash dump file.

The **makedumpfile --mem-usage** command estimates the space required by the crash dump file. It generates a memory usage report. The report helps you decide the dump level and the pages that are safe to exclude.

Procedure

- Enter the following command to generate a memory usage report:

```
# makedumpfile --mem-usage /proc/kcore
```

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

IMPORTANT

The **makedumpfile --mem-usage** command reports required memory in pages. This means that you must calculate the size of memory in use against the kernel page size.

By default the RHEL kernel uses 4 KB sized pages on AMD64 and Intel 64 CPU architectures, and 64 KB sized pages on IBM POWER architectures.

15.2. CONFIGURING KDUMP MEMORY USAGE

The memory reservation for **kdump** occurs during the system boot. The memory size is set in the system's Grand Unified Bootloader (GRUB) configuration. The memory size depends on the value of the **crashkernel=** option specified in the configuration file and the size of the system physical memory.

You can define the **crashkernel=** option in many ways. You can specify the **crashkernel=** value or configure the **auto** option. The **crashkernel=auto** parameter reserves memory automatically, based on the total amount of physical memory in the system. When configured, the kernel automatically reserves an appropriate amount of required memory for the capture kernel. This helps to prevent Out-of-Memory (OOM) errors.



NOTE

The automatic memory allocation for **kdump** varies based on system hardware architecture and available memory size.

For example, on AMD64 and Intel 64, the **crashkernel=auto** parameter works only when the available memory is more than 1GB. The 64-bit ARM architecture and IBM Power Systems needs more than 2 GB of available memory.

If the system has less than the minimum memory threshold for automatic allocation, you can configure the amount of reserved memory manually.

Prerequisites

- You have root permissions on the system.
- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).

Procedure

1. Prepare the **crashkernel=** option.

- For example, to reserve 128 MB of memory, use the following:

```
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>**. For example:

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

The command reserves 64 MB of memory if the total amount of system memory is in the range of 512 MB and 2 GB. If the total amount of memory is more than 2 GB, the memory reserve is 128 MB.

- Offset the reserved memory.
Some systems require to reserve memory with a certain fixed offset because the **crashkernel** reservation happens early, and you may need to reserve more memory for special usage. When you define an offset, the reserved memory begins there. To offset the reserved memory, use the following syntax:

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

In this example, **kdump** reserves 128 MB of memory starting at 16 MB (physical address **0x01000000**). If you set the offset parameter to 0 or omit entirely, **kdump** offsets the reserved memory automatically. You can also use this syntax when setting a variable memory reservation. In that case, the offset is always specified last. For example:

```
crashkernel=512M-2G:64M,2G-:128M@16M
```

2. Apply the **crashkernel=** option to your boot loader configuration:

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

Replace **<value>** with the value of the **crashkernel=** option that you prepared in the previous step.

Additional resources

- [Memory requirements for kdump](#)
- [Configuring kernel command-line parameters](#)
- [How to manually modify the boot parameter in grub before the system boots](#) (Red Hat Knowledgebase)
- [How to install and boot custom kernels in Red Hat Enterprise Linux 8](#)
- **grubby(8)** manual page

15.3. CONFIGURING THE KDUMP TARGET

The crash dump is usually stored as a file in a local file system, written directly to a device. Optionally, you can send crash dump over a network by using the **NFS** or **SSH** protocols. Only one of these options to preserve a crash dump file can be set at a time. The default behavior is to store it in the **/var/crash/** directory of the local file system.

Prerequisites

- You have root permissions on the system.
- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).

Procedure

- To store the crash dump 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 crash dump file.



NOTE

- When you specify a dump target in the **/etc/kdump.conf** file, then the path is **relative** to the specified dump target.
- When 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 the file system mounted in the current system, the dump target and the adjusted dump path are configured automatically.

- To secure the crash dump file and the accompanying files produced by **kdump**, you should set

up proper attributes for the target destination directory, such as user permissions and SELinux contexts. Additionally, you can define a script, for example **kdump_post.sh** in the **kdump.conf** file as follows:

```
kdump_post <path_to_kdump_post.sh>
```

The **kdump_post** directive specifies a shell script or a command that executes **after kdump** has completed capturing and saving a crash dump to the specified destination. You can use this mechanism to extend the functionality of **kdump** to perform actions including the adjustments in file permissions.

- The **kdump** target configuration

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

The dump target is specified (**ext4 /dev/mapper/vg00-varcrashvol**), and, therefore, it is mounted at **/var/crash**. The **path** option is also set to **/var/crash**. Therefore, the **kdump** saves the **vmcore** file in the **/var/crash/var/crash** directory.

- To change the local directory for saving the crash dump, edit the **/etc/kdump.conf** configuration file as a **root** user:
 - a. Remove the hash sign (#) from the beginning of the **#path /var/crash** line.
 - b. Replace the value with the intended directory path. For example:

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



IMPORTANT

In RHEL 8, the directory defined as the **kdump** target using the **path** directive must exist when the **kdump systemd** service starts to avoid failures. Unlike in earlier versions of RHEL, the directory is no longer created automatically if it does not exist when the service starts.

- To write the file to a different partition, edit the **/etc/kdump.conf** configuration file:
 - a. 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)
 - b. Change the file system type and the device name, label or UUID, to the required values. The correct syntax for specifying UUID values is both **UUID="correct-uuid"** and **UUID=correct-uuid**. For example:

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



IMPORTANT

It is recommended to specify storage devices by using a **LABEL=** or **UUID=**. Disk device names such as **/dev/sda3** are not guaranteed to be consistent across reboot.

When you use Direct Access Storage Device (DASD) on IBM Z hardware, ensure the dump devices are correctly specified in **/etc/dasd.conf** before proceeding with **kdump**.

- To write the crash dump directly to a device, edit the **/etc/kdump.conf** configuration file:
 - a. Remove the hash sign (**#**) from the beginning of the **#raw /dev/vg/lv_kdump** line.
 - b. Replace the value with the intended device name. For example:

```
raw /dev/sdb1
```

- To store the crash dump to a remote machine by using the **NFS** protocol:
 - a. Remove the hash sign (**#**) from the beginning of the **#nfs my.server.com:/export/tmp** line.
 - b. Replace the value with a valid hostname and directory path. For example:

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

- c. Restart the **kdump** service for the changes to take effect:

```
sudo systemctl restart kdump.service
```



NOTE

While using the NFS directive to specify the NFS target, **kdump.service** automatically attempts to mount the NFS target to check the disk space. There is no need to mount the NFS target in advance. To prevent **kdump.service** from mounting the target, use the **dracut_args --mount** directive in **kdump.conf**. This will enable **kdump.service** to call the **dracut** utility with the **--mount** argument to specify the NFS target.

- To store the crash dump to a remote machine by using the SSH protocol:
 - a. Remove the hash sign (**#**) from the beginning of the **#ssh user@my.server.com** line.
 - b. Replace the value with a valid username and hostname.
 - c. Include your SSH key in the configuration.
 - i. Remove the hash sign from the beginning of the **#sshkey /root/.ssh/kdump_id_rsa** line.
 - ii. 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
```

Additional resources

[Files produced by kdump after system crash](#) .

15.4. CONFIGURING THE KDUMP CORE COLLECTOR

The **kdump** service uses a **core_collector** program to capture the crash dump image. In RHEL, the **makedumpfile** utility is the default core collector. It helps shrink the dump file by:

- Compressing the size of a crash dump file and copying only necessary pages by using various dump levels.
- Excluding unnecessary crash dump pages.
- Filtering the page types to be included in the crash dump.



NOTE

Crash dump file compression is enabled by default in the RHEL 7 and above.

If you need to customize the crash dump file compression, follow this procedure.

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

- You have root permissions on the system.
- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).

Procedure

1. As a **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. Enter the following command to enable crash dump file compression:

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


The **-l** option specifies the **dump** compressed file format. The **-d** option specifies dump level as 31. The **--message-level** option specifies message level as 1.

Also, consider following examples with the **-c** and **-p** options:

- To compress a crash dump file by using **-c**:

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

- To compress a crash dump file by using **-p**:

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

Additional resources

- **makedumpfile(8)** man page on your system
- [Configuration file for kdump](#)

15.5. CONFIGURING THE KDUMP DEFAULT FAILURE RESPONSES

By default, when **kdump** fails to create a crash dump file at the configured target location, the system reboots and the dump is lost in the process. You can change the default failure response and configure **kdump** to perform a different operation when it fails to save the core dump to the primary target. The additional actions are:

dump_to_rootfs

Saves the core dump to the **root** file system.

reboot

Reboots the system, losing the core dump in the process.

halt

Stops the system, losing the core dump in the process.

poweroff

Power the system off, losing the core dump in the process.

shell

Runs a shell session from within the **initramfs**, you can record the core dump manually.

final_action

Enables additional operations such as **reboot**, **halt**, and **poweroff** after a successful **kdump** or when shell or **dump_to_rootfs** failure action completes. The default is **reboot**.

failure_action

Specifies the action to perform when a dump might fail in a kernel crash. The default is **reboot**.

Prerequisites

- Root permissions.
- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).

Procedure

1. As a **root** user, 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 required action.

```
failure_action poweroff
```

Additional resources

- [Configuring the kdump target](#)

15.6. 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 the **KDUMP_COMMANDLINE_APPEND** option to append the **kdump** kernel command-line to obtain a detailed debugging output or the **KDUMP_COMMANDLINE_REMOVE** option to remove arguments from the **kdump** command line.

KDUMP_COMMANDLINE_REMOVE

This option removes arguments from the current **kdump** command line. It removes parameters that can cause **kdump** errors or **kdump** kernel boot failures. These parameters might 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.

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

```
# KDUMP_COMMANDLINE_REMOVE="hugepages hugepagesz slub_debug quiet log_buf_len swiotlb"
```

KDUMP_COMMANDLINE_APPEND

This option appends arguments to the current command line. These arguments might 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 can consume a significant part of the kernel memory reserved for **kdump** or cause **kdump** kernel boot failures.

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

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

Additional resources

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

15.7. TESTING THE KDUMP CONFIGURATION

After configuring **kdump**, you must manually test a system crash and ensure that the **vmcore** file is generated in the defined **kdump** target. The **vmcore** file is captured from the context of the freshly booted kernel. Therefore, **vmcore** has critical information for debugging a kernel crash.



WARNING

Do not test **kdump** on active production systems. The commands to test **kdump** will cause the kernel to crash with loss of data. Depending on your system architecture, ensure that you schedule significant maintenance time because **kdump** testing might require several reboots with a long boot time.

If the **vmcore** file is not generated during the **kdump** test, identify and fix issues before you run the test again for a successful **kdump** testing.

If you make any manual system modifications, you must test the **kdump** configuration at the end of any system modification. For example, if you make any of the following changes, ensure that you test the **kdump** configuration for an optimal **kdump** performances for:

- Package upgrades.
- Hardware level changes, for example, storage or networking changes.
- Firmware upgrades.
- New installation and application upgrades that include third party modules.
- If you use the hot-plugging mechanism to add more memory on hardware that support this mechanism.
- After you make changes in the **/etc/kdump.conf** or **/etc/sysconfig/kdump** file.

Prerequisites

- You have root permissions on the system.
- You have saved all important data. The commands to test **kdump** cause the kernel to crash with loss of data.
- You have scheduled significant machine maintenance time depending on the system architecture.

Procedure

1. Enable the **kdump** service:

```
# kdumpctl restart
```

2. Check the status of the **kdump** service with the **kdumpctl**:

```
# kdumpctl status
kdump:Kdump is operational
```

Optionally, if you use the **systemctl** command, the output prints in the **systemd** journal.

3. Start a kernel crash to test the **kdump** configuration. The **sysrq-trigger** key combination causes the kernel to crash and might reboot the system if required.

```
# echo c > /proc/sysrq-trigger
```

On a kernel reboot, the **address-YYYY-MM-DD-HH:MM:SS/vmcore** file is created at the location you have specified in the **/etc/kdump.conf** file. The default is **/var/crash/**.

Additional resources

- [Configuring the kdump target](#)

15.8. FILES PRODUCED BY KDUMP AFTER SYSTEM CRASH

After your system crashes, the **kdump** service captures the kernel memory in a dump file (**vmcore**) and it also generates additional diagnostic files to aid in troubleshooting and postmortem analysis.

Files produced by **kdump**:

- **vmcore** - main kernel memory dump file containing system memory at the time of the crash. It includes data as per the configuration of the **core_collector** program specified in **kdump** configuration. By default the kernel data structures, process information, stack traces, and other diagnostic information.
- **vmcore-dmesg.txt** - contents of the kernel ring buffer log (**dmesg**) from the primary kernel that panicked.
- **kexec-dmesg.log** - has kernel and system log messages from the execution of the secondary **kexec** kernel that collects the **vmcore** data.

Additional resources

- [What is the kernel ring buffer](#)

15.9. ENABLING AND DISABLING THE KDUMP SERVICE

You can configure to enable or disable the **kdump** functionality on a specific kernel or on all installed kernels. You must routinely test the **kdump** functionality and validate its operates correctly.

Prerequisites

- You have root permissions on the system.
- You have completed **kdump** requirements for configurations and targets. See [Supported kdump configurations and targets](#).
- All configurations for installing **kdump** are set up as required.

Procedure

1. Enable the **kdump** service for **multi-user.target**:

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

2. Start the service in the current session:

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

3. Stop the **kdump** service:

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

4. Disable the **kdump** service:

```
# 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 regardless of whether the Kernel Address Space Layout (**KASLR**) is enabled.

If **kptr_restrict** is not set to **1** and KASLR is enabled, the contents of **/proc/kcore** file are generated as all zeros. The **kdumpctl** service fails to access the **/proc/kcore** file and load the crash kernel. The **kexec-kdump-howto.txt** file displays a warning message, which recommends you to set **kptr_restrict=1**. Verify for the following in the **sysctl.conf** file to ensure that **kdumpctl** service loads the crash kernel:

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

15.10. PREVENTING KERNEL DRIVERS FROM LOADING FOR KDUMP

You can control the capture kernel from loading certain kernel drivers by adding the **KDUMP_COMMANDLINE_APPEND=** variable in the **/etc/sysconfig/kdump** configuration file. By using this method, you can prevent the **kdump** initial RAM disk image **initramfs** from loading the specified kernel module. This helps to prevent the out-of-memory (OOM) killer errors or other crash kernel failures.

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

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

Prerequisites

- You have root permissions on the system.

Procedure

1. Display the list of modules that are loaded to the currently running kernel. Select the 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_contrack           16384   1
```

2. Update the **KDUMP_COMMANDLINE_APPEND=** variable in the **/etc/sysconfig/kdump** file. For example:

```
KDUMP_COMMANDLINE_APPEND="rd.driver.blacklist=hv_vmbus,hv_storvsc,hv_utils,hv_netvsc,hid-hyperv"
```

Also, consider the following example by 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** man page on your system.

15.11. RUNNING KDUMP ON SYSTEMS WITH ENCRYPTED DISK

When you run a LUKS encrypted partition, systems require certain amount of available memory. If the system has less than the required amount of available memory, the **cryptsetup** utility fails to mount the partition. As a result, capturing the **vmcore** file to an encrypted target location fails in the second kernel (capture kernel).

The **kdumpectl estimate** command helps you estimate the amount of memory you need for **kdump**. **kdumpectl estimate** prints the recommended **crashkernel** value, which is the most suitable memory size required for **kdump**.

The recommended **crashkernel** value is calculated based on the current kernel size, kernel module, initramfs, and the LUKS encrypted target memory requirement.

If you are using the custom **crashkernel=** option, **kdumpectl estimate** prints the **LUKS required size** value. The value is the memory size required for LUKS encrypted target.

Procedure

1. Print the estimate **crashkernel=** value:

```
# *kdumpctl estimate*
```

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

Reserved crashkernel: 256M

Recommended crashkernel: 652M

Kernel image size: 47M

Kernel modules size: 8M

Initramfs size: 20M

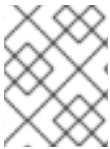
Runtime reservation: 64M

LUKS required size: 512M

Large modules: <none>

WARNING: Current crashkernel size is lower than recommended size 652M.

2. Configure the amount of required memory by increasing the **crashkernel=** value.
3. Reboot the system.



NOTE

If the **kdump** service still fails to save the dump file to the encrypted target, increase the **crashkernel=** value as required.

CHAPTER 16. CONFIGURING KDUMP IN THE WEB CONSOLE

You can set up and test the **kdump** configuration by using the RHEL 8 web console. The web console can enable the **kdump** service at boot time. With the web console, you can configure the reserved memory for **kdump** and to select the **vmcore** saving location in an uncompressed or compressed format.

16.1. CONFIGURING KDUMP MEMORY USAGE AND TARGET LOCATION IN WEB CONSOLE

You can configure the memory reserve for the **kdump** kernel and also specify the target location to capture the **vmcore** dump file with the RHEL web console interface.

Prerequisites

- The web console must be installed and accessible. For details, see [Installing the web console](#).

Procedure

1. In the web console, open the **Kernel dump** tab and start the **kdump** service by setting the **Kernel crash dump** switch to on.
2. Configure the **kdump** memory usage in the terminal, for example:

```
$ sudo grubby --update-kernel ALL --args crashkernel=512M
```

Restart the system to apply the changes.

3. In the **Kernel dump** tab, click **Edit** at the end of the **Crash dump location** field.
4. Specify the target directory for saving the **vmcore** dump file:
 - For a local filesystem, select **Local Filesystem** from the drop-down menu.
 - For a remote system by using the SSH protocol, select **Remote over SSH** from the drop-down menu and specify the following fields:
 - In the **Server** field, enter the remote server address.
 - In the **SSH key** field, enter the SSH key location.
 - In the **Directory** field, enter the target directory.
 - For a remote system by using the NFS protocol, select **Remote over NFS** from the drop-down menu and specify the following fields:
 - In the **Server** field, enter the remote server address.
 - In the **Export** field, enter the location of the shared folder of an NFS server.
 - In the **Directory** field, enter the target directory.

**NOTE**

You can reduce the size of the **vmcore** file by selecting the **Compression** checkbox.

5. Optional: Display the automation script by clicking **View automation script**
A window with the generated script opens. You can browse a shell script and an Ansible playbook generation options tab.
6. Optional: Copy the script by clicking **Copy to clipboard**
You can use this script to apply the same configuration on multiple machines.

Verification

1. Click **Test configuration**.
2. Click **Crash system** under **Test kdump settings**.

**WARNING**

When you start the system crash, the kernel operation stops and results in a system crash with data loss.

Additional resources

- [Supported kdump targets](#)

CHAPTER 17. ENABLING KDUMP

For your RHEL 8 systems, you can configure enabling or disabling the **kdump** functionality on a specific kernel or on all installed kernels. However, you must routinely test the **kdump** functionality and validate its working status.

17.1. ENABLING KDUMP FOR ALL INSTALLED KERNELS

The **kdump** service starts by enabling **kdump.service** after the **kexec** tool is installed. You can enable and start the **kdump** service for all kernels installed on the machine.

Prerequisites

- You have administrator privileges.

Procedure

1. Add the **crashkernel=** command-line parameter to all installed kernels:

```
# grubby --update-kernel=ALL --args="crashkernel=xxM"
```

xxM is the required memory in megabytes.

2. Reboot the system:

```
# reboot
```

3. Enable the **kdump** service:

```
# systemctl enable --now kdump.service
```

Verification

- Check that the **kdump** service is running:

```
# systemctl status kdump.service
```

```
○ kdump.service - Crash recovery kernel arming
```

```
Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset: disabled)
```

```
Active: active (live)
```

17.2. ENABLING KDUMP FOR A SPECIFIC INSTALLED KERNEL

You can enable the **kdump** service for a specific kernel on the machine.

Prerequisites

- You have administrator privileges.

Procedure

1. List the kernels installed on the machine.

```
# ls -a /boot/vmlinuz-*
/boot/vmlinuz-0-rescue-2930657cd0dc43c2b75db480e5e5b4a9
/boot/vmlinuz-4.18.0-330.el8.x86_64
/boot/vmlinuz-4.18.0-330.rt7.111.el8.x86_64
```

2. Add a specific **kdump** kernel to the system's Grand Unified Bootloader (GRUB) configuration. For example:

```
# grubby --update-kernel=vmlinuz-4.18.0-330.el8.x86_64 --args="crashkernel=xxM"
```

xxM is the required memory reserve in megabytes.

3. Enable the **kdump** service.

```
# systemctl enable --now kdump.service
```

Verification

- Check that the **kdump** service is running.

```
# systemctl status kdump.service
○ kdump.service - Crash recovery kernel arming
  Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset: disabled)
  Active: active (live)
```

17.3. DISABLING THE KDUMP SERVICE

You can stop the **kdump.service** and disable the service from starting on your RHEL 8 systems.

Prerequisites

- Fulfilled requirements for **kdump** configurations and targets. For details, see [Supported kdump configurations and targets](#).
- All configurations for installing **kdump** are set up according to your needs. For details, see [Installing kdump](#).

Procedure

1. To stop the **kdump** service in the current session:

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

2. To disable the **kdump** service:

```
# 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 regardless of whether the Kernel Address Space Layout (**KASLR**) is enabled.

If **kptr_restrict** is not set to **1** and **KASLR** is enabled, the contents of **/proc/kcore** file are generated as all zeros. The **kdumpctl** service fails to access the **/proc/kcore** file and load the crash kernel. The **kexec-kdump-howto.txt** file displays a warning message, which recommends you to set **kptr_restrict=1**. Verify for the following in the **sysctl.conf** file to ensure that **kdumpctl** service loads the crash kernel:

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

Additional resources

- [Managing systemd](#)

CHAPTER 18. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS

The **kdump** mechanism is a feature of the Linux kernel that generates a crash dump file when a kernel crash occurs. The kernel dump file has critical information that helps to analyze and determine the root cause of a kernel crash. The crash can be because of various factors, hardware issues or third-party kernel modules problems, to name a few.

By using the provided information and procedures, you can perform the following actions:

- Identify the supported configurations and targets for your RHEL 8 systems.
- Configure **kdump**.
- Verify **kdump** operation.

18.1. MEMORY REQUIREMENTS FOR KDUMP

For **kdump** to capture a kernel crash dump and save it for further analysis, a part of the system memory should 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 identify 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
```

With the stated list of minimum memory requirements, you can set the appropriate memory size to automatically reserve a memory for **kdump** on the latest available versions. The memory size depends on the system's architecture and total available physical memory.

Table 18.1. Minimum amount of reserved memory required for **kdump**

Architecture	Available Memory	Minimum Reserved Memory
AMD64 and Intel 64 (x86_64)	1 GB to 4 GB	192 MB of RAM
	4 GB to 64 GB	256 MB of RAM
	64 GB and more	512 MB of RAM
64-bit ARM architecture (arm64)	2 GB and more	480 MB of RAM
IBM Power Systems (ppc64le)	2 GB to 4 GB	384 MB of RAM
	4 GB to 16 GB	512 MB of RAM
	16 GB to 64 GB	1 GB of RAM
	64 GB to 128 GB	2 GB of RAM

Architecture	Available Memory	Minimum Reserved Memory
	128 GB and more	4 GB of RAM
IBM Z (s390x)	1 GB to 4 GB	192 MB of RAM
	4 GB to 64 GB	256 MB of RAM
	64 GB and more	512 MB of RAM

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 might vary due to other factors such as I/O devices. Using not enough of memory might cause debug kernel unable to boot as a capture kernel in the case of kernel panic. To avoid this problem, increase the crash kernel memory sufficiently.

Additional resources

- [How has the crashkernel parameter changed between RHEL8 minor releases?](#) (Red Hat Knowledgebase)
- [Technology capabilities and limits tables](#)
- [Minimum threshold for automatic memory reservation](#)

18.2. MINIMUM THRESHOLD FOR AUTOMATIC MEMORY RESERVATION

By default, the **kexec-tools** utility configures the **crashkernel** command line parameter and reserves a certain amount of memory for **kdump**. On some systems however, it is still possible to assign memory for **kdump** 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, a certain amount of total memory needs to be available in the system. The memory requirement varies based on the system's architecture. If the system memory is less than the specified threshold value, you must configure the memory manually.

Table 18.2. Minimum amount of memory required for automatic memory reservation

Architecture	Required Memory
AMD64 and Intel 64 (x86_64)	2 GB
IBM Power Systems (ppc64le)	2 GB

Architecture	Required Memory
IBM Z (s390x)	4 GB

**NOTE**

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

18.3. SUPPORTED KDUMP TARGETS

When a kernel crash occurs, the operating system saves the dump file on the configured or default target location. You can save the dump file either directly to a device, store as a file on a local file system, or send the dump file over a network. With the following list of dump targets, you can know the targets that are currently supported or not supported by **kdump**.

Table 18.3. kdump targets on RHEL 8

Target type	Supported Targets	Unsupported Targets
Physical storage	<ul style="list-style-type: none"> Logical Volume Manager (LVM). Thin provisioning volume. Fibre Channel (FC) disks such as qla2xxx, lpfc, bnx2fc, and bfa. An iSCSI software-configured logical device on a networked storage server. The mdraid subsystem as a software RAID solution. Hardware RAID such as cciss, hpsa, megaraid_sas, mpt2sas, and aacraid. SCSI and SATA disks. iSCSI and HBA offloads. Hardware FCoE such as qla2xxx and lpfc. 	<ul style="list-style-type: none"> BIOS RAID. Software iSCSI with iBFT. Currently supported transports are bnx2i, cxgb3i, and cxgb4i. Software iSCSI with a hybrid device driver such as be2iscsi. Fibre Channel over Ethernet (FCoE). Legacy IDE. GlusterFS servers. GFS2 file system. Clustered Logical Volume Manager (CLVM). High availability LVM volumes (HA-LVM).

Target type	Supported Targets	Unsupported Targets
Network	<ul style="list-style-type: none"> Hardware using kernel modules: tg3, igb, ixgbe, sfc, e1000e, bna, cnic, netxen_nic, qlge, bnx2x, bnx, qlcnict, be2net, enic, virtio-net, ixgbev, igbvf. IPv4 protocol. Network bonding on different devices, such as Ethernet devices or VLAN. VLAN network. Network Bridge. Network Teaming. Tagged VLAN and VLAN over a bond. Bridge network over bond, team, and VLAN. 	<ul style="list-style-type: none"> IPv6 protocol. Wireless connections. InfiniBand networks. VLAN network over bridge and team.
Hypervisor	<ul style="list-style-type: none"> Kernel-based virtual machines (KVM). Xen hypervisor in certain configurations only. VMware ESXi 4.1 and 5.1. Hyper-V 2012 R2 on RHEL Gen1 UP Guest only. 	
File systems	The ext[234], XFS, and NFS file systems.	The Btrfs file system.
Firmware	<ul style="list-style-type: none"> BIOS-based systems. UEFI Secure Boot. 	

Additional resources

- [Configuring the kdump target](#)

18.4. SUPPORTED KDUMP FILTERING LEVELS

To reduce the size of the dump file, **kdump** uses the **makedumpfile** core collector to compress the data and also exclude unwanted information, for example, you can remove **hugepages** and **hugetlbfs** pages by using the **-8** level. The levels that **makedumpfile** currently supports can be seen in the table for *Filtering levels for `kdump`*.

Table 18.4. Filtering levels forkdump

Option	Description
1	Zero pages
2	Cache pages
4	Cache private
8	User pages
16	Free pages

Additional resources

- [Configuring the kdump core collector](#)

18.5. SUPPORTED DEFAULT FAILURE RESPONSES

By default, when **kdump** fails to create a core dump, the operating system reboots. However, you can configure **kdump** to perform a different operation in case it fails to save the core dump to the primary target.

Table 18.5. Failure responses forkdump

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

Option	Description
shell	Run a shell session from within the initramfs, allowing the user to record the core dump manually.
final_action	Enable additional operations such as reboot , halt , and poweroff actions after a successful kdump or when shell or dump_to_rootfs failure action completes. The default final_action option is reboot .

Additional resources

- [Configuring the kdump default failure responses](#)

18.6. USING FINAL_ACTION PARAMETER

When **kdump** succeeds or if **kdump** fails to save the **vmcore** file at the configured target, you can perform additional operations like **reboot**, **halt**, and **poweroff** by using the **final_action** parameter. If the **final_action** parameter is not specified, **reboot** is the default response.

Procedure

1. To configure **final_action**, edit the **/etc/kdump.conf** file and add one of the following options:
 - **final_action reboot**
 - **final_action halt**
 - **final_action poweroff**
2. Restart the **kdump** service for the changes to take effect.

```
# kdumpctl restart
```

18.7. USING FAILURE_ACTION PARAMETER

The **failure_action** parameter specifies the action to perform when a dump fails in the event of a kernel crash. The default action for **failure_action** is **reboot** that reboots the system.

The parameter recognizes the following actions to take:

reboot

Reboots the system after a dump failure.

dump_to_rootfs

Saves the dump file on a root file system when a non-root dump target is configured.

halt

Halts the system.

poweroff

Stops the running operations on the system.

shell

Starts a shell session inside **initramfs**, from which you can manually perform additional recovery actions.

Procedure

1. To configure an action to take if the dump fails, edit the **/etc/kdump.conf** file and specify one of the **failure_action** options:
 - **failure_action reboot**
 - **failure_action halt**
 - **failure_action poweroff**
 - **failure_action shell**
 - **failure_action dump_to_rootfs**
2. Restart the **kdump** service for the changes to take effect.

```
# kdumpctl restart
```

CHAPTER 19. FIRMWARE ASSISTED DUMP MECHANISMS

Firmware assisted dump (fadump) is a dump capturing mechanism, provided as an alternative to the **kdump** mechanism on IBM POWER systems. The **kexec** and **kdump** mechanisms are useful for capturing core dumps on AMD64 and Intel 64 systems. However, some hardware, such as mini systems and mainframe computers, uses the onboard firmware to isolate regions of memory and prevent any accidental overwriting of data that is important to the crash analysis. The **fadump** utility is optimized for the **fadump** mechanisms and their integration with RHEL on IBM POWER systems.

19.1. FIRMWARE ASSISTED DUMP ON IBM POWERPC HARDWARE

The **fadump** utility captures the **vmcore** file from a fully-reset system with PCI and I/O devices. This mechanism uses firmware to preserve memory regions during a crash and then reuses the **kdump** userspace scripts to save the **vmcore** file. The memory regions consist of all system memory contents, except the boot memory, system registers, and hardware Page Table Entries (PTEs).

The **fadump** mechanism offers improved reliability over the traditional dump type, by rebooting the partition and using a new kernel to dump the data from the previous kernel crash. The **fadump** requires an IBM POWER6 processor-based or later version hardware platform.

For further details about the **fadump** mechanism, including PowerPC specific methods of resetting hardware, see the `/usr/share/doc/kexec-tools/fadump-howto.txt` file.



NOTE

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

Unlike **kexec-initiated** event, the **fadump** mechanism uses the production kernel to recover a crash dump. When booting after a crash, PowerPC hardware makes the device node `/proc/device-tree/rtas/ibm.kernel-dump` available to the **proc** filesystem (**procfs**). The **fadump-aware kdump** scripts, check for the stored **vmcore**, and then complete the system reboot cleanly.

19.2. ENABLING FIRMWARE ASSISTED DUMP MECHANISM

You can enhance the crash dumping capabilities of IBM POWER systems by enabling the firmware assisted dump (**fadump**) mechanism.

In the Secure Boot environment, the GRUB boot loader allocates a boot memory region, known as the Real Mode Area (RMA). The RMA has a size of 512 MB, divided among the boot components. If a component exceeds its size allocation, **GRUB** fails with an out-of-memory (**OOM**) error.



WARNING

Do not enable firmware assisted dump (**fadump**) mechanism in the Secure Boot environment on RHEL 8.7 and 8.6 versions. The **GRUB2** boot loader fails with the following error:

```
error: ../../grub-core/kern/mm.c:376:out of memory.
Press any key to continue...
```

The system is recoverable only if you increase the default **initramfs** size due to the **fadump** configuration.

For information about workaround methods to recover the system, see the [System boot ends in GRUB Out of Memory \(OOM\)](#) article.

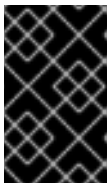
Procedure

1. Install and configure **kdump**.
2. Enable the **fadump=on** kernel option:

```
# grubby --update-kernel=ALL --args="fadump=on"
```

3. Optional: If you want to specify reserved boot memory instead of using the defaults, enable the **crashkernel=xxM** option, where **xx** is the amount of the memory required in megabytes:

```
# grubby --update-kernel=ALL --args="crashkernel=xxM fadump=on"
```



IMPORTANT

When specifying boot configuration options, test all boot configuration options before you run them. If the **kdump** kernel fails to boot, increase the value specified in **crashkernel=** argument gradually to set an appropriate value.

19.3. FIRMWARE ASSISTED DUMP MECHANISMS ON IBM Z HARDWARE

IBM Z systems support the following firmware assisted dump mechanisms:

- **Stand-alone dump (sadump)**
- **VMDUMP**

The **kdump** infrastructure is supported and utilized on IBM Z systems. However, using one of the firmware assisted dump (fadump) methods for IBM Z has the following benefits:

- The system console initiates and controls the **sadump** mechanism, and stores it on an **IPL** bootable device.

- The **VMDUMP** mechanism is similar to **sadump**. This tool is also initiated from the system console, but retrieves the resulting dump from hardware and copies it to the system for analysis.
- These methods (similarly to other hardware based dump mechanisms) have the ability to capture the state of a machine in the early boot phase, before the **kdump** service starts.
- Although **VMDUMP** contains a mechanism to receive the dump file into a Red Hat Enterprise Linux system, the configuration and control of **VMDUMP** is managed from the IBM Z Hardware console.

Additional resources

- [Stand-alone dump program](#)
- [Creating dumps on z/VM with VMDUMP](#)
- [Using the Dump Tools on Red Hat Enterprise Linux 7.4](#)

19.4. USING SADUMP ON FUJITSU PRIMEQUEST SYSTEMS

The Fujitsu **sadump** mechanism provides a **fallback** dump capture when **kdump** is unable to complete successfully. You can manually invoke **sadump** from the system Management Board (MMB) interface. Using MMB, configure **kdump** like for an Intel 64 or AMD64 server and then proceed to enable **sadump**.

Procedure

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

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



WARNING

In particular, ensure that after **kdump**, the system does not reboot. If the system reboots after **kdump** has failed to save the **vmcore** file, then it is not possible to invoke the **sadump**.

2. Set the **failure_action** parameter in **/etc/kdump.conf** appropriately as **halt** or **shell**.

```
failure_action shell
```

Additional resources

- The FUJITSU Server PRIMEQUEST 2000 Series Installation Manual

CHAPTER 20. ANALYZING A CORE DUMP

To identify the cause of the system crash, you can use the **crash** utility, which provides an interactive prompt similar to the GNU Debugger (GDB). By using **crash**, you can analyze a core dump created by **kdump**, **netdump**, **diskdump**, or **xendump** and a running Linux system. Alternatively, you can use the Kernel Oops Analyzer or the Kdump Helper tool.

20.1. INSTALLING THE CRASH UTILITY

With the provided information, understand the required packages and the procedure to install the **crash** utility. The **crash** utility might not be installed by default on your RHEL 8 systems. **crash** is a tool to interactively analyze a system's state while it is running or after a kernel crash occurs and a core dump file is created. The core dump file is also known as the **vmcore** file.

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 **kernel-debuginfo** will correspond to the running kernel and provides the data necessary for the dump analysis.

20.2. RUNNING AND EXITING THE CRASH UTILITY

The **crash** utility is a powerful tool for analyzing **kdump**. By running **crash** on a crash dump file, you can gain insights into the system's state at the time of the crash, identify the root cause of the issue, and troubleshoot kernel-related problems.

Prerequisites

- Identify the currently running kernel (for example **4.18.0-5.el8.x86_64**).

Procedure

1. To start the **crash** utility, pass the following two necessary parameters:
 - The debug-info (a decompressed vmlinuz image), for example **/usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinuz** provided through a specific **kernel-debuginfo** package.

- The actual vmcore file, for example **/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore**. The resulting **crash** command will be as follows:

```
# crash /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
```

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

Example 20.1. Running the crash utility

The following example shows analyzing a core dump created on October 6 2018 at 14:05 PM, using the 4.18.0-5.el8.x86_64 kernel.

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

    KERNEL: /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore [PARTIAL DUMP]
    CPUS: 2
    DATE: Sat Oct 6 14:05:16 2018
    UPTIME: 01:03:57
    LOAD AVERAGE: 0.00, 0.00, 0.00
    TASKS: 586
    NODENAME: localhost.localdomain
    RELEASE: 4.18.0-5.el8.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)

crash>
```

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

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



NOTE

The **crash** command is also utilized as a powerful tool for debugging a live system. However, you must use it with caution to avoid system-level issues.

Additional resources

- [A Guide to Unexpected System Restarts](#)

20.3. DISPLAYING VARIOUS INDICATORS IN THE CRASH UTILITY

Use the **crash** utility to 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:

```
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> ffffffff c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002 b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560 ef4dbfb4
Call Trace:
[<c068146b>] ? __handle_sysrq+0xfb/0x160
[<c06814d0>] ? write_sysrq_trigger+0x0/0x50
[<c068150f>] ? write_sysrq_trigger+0x3f/0x50
[<c0569ec4>] ? proc_reg_write+0x64/0xa0
[<c0569e60>] ? proc_reg_write+0x0/0xa0
[<c051de50>] ? vfs_write+0xa0/0x190
[<c051e8d1>] ? sys_write+0x41/0x70
[<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04 09 d0 88 41 03 f3 c3 90 c7
05 c8 1b 9e c0 01 00 00 00 0f ae f8 89 f6 <c6> 05 00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00
00 8d 50 d0 83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000
```

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



NOTE

The kernel message buffer includes the most essential information about the system crash. It is always dumped first in to the **vmcore-dmesg.txt** file. If you fail to obtain the full **vmcore** file, for example, due to insufficient space on the target location, you can obtain the required information from the kernel message buffer. By default, **vmcore-dmesg.txt** is placed 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
#2 [ef4dbe34] no_context at c043089d
#3 [ef4dbe58] bad_area at c0430b26
#4 [ef4dbe6c] do_page_fault at c080fb9b
```

```
#5 [ef4dbee4] error_code (via page_fault) at c080d809
  EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000 EBP: 00000000
  DS: 007b   ESI: c0a09ca0 ES: 007b   EDI: 00000286 GS: 00e0
  CS: 0060   EIP: c068124f ERR: ffffffff EFLAGS: 00010096
#6 [ef4dbf18] sysrq_handle_crash at c068124f
#7 [ef4dbf24] __handle_sysrq at c0681469
#8 [ef4dbf48] write_sysrq_trigger at c068150a
#9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
  EAX: ffffffff EBX: 00000001 ECX: b7776000 EDX: 00000002
  DS: 007b   ESI: 00000002 ES: 007b   EDI: b7776000
  SS: 007b   ESP: bfc2088 EBP: bfc20b4 GS: 0033
  CS: 0073   EIP: 00edc416 ERR: 00000004 EFLAGS: 00000246
```

Type **bt <pid>** to display the backtrace of a specific process or type **help bt** for more information about **bt** usage.

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    0  1 f7072030 RU  0.0   0    0 [swapper]
    0    0  2 f70a3a90 RU  0.0   0    0 [swapper]
>  0    0  3 f70ac560 RU  0.0   0    0 [swapper]
    1    0  1 f705ba90 IN  0.0 2828 1424 init
... several lines omitted ...
 5566   1  1 f2592560 IN  0.0 12876  784 auditd
 5567   1  2 ef427560 IN  0.0 12876  784 auditd
 5587 5132  0 f196d030 IN  0.0 11064 3184 sshd
> 5591 5587  2 f196d560 RU  0.0  5084 1648 bash
```

Use **ps <pid>** to display the status of a single specific process. Use **help ps** for more information about **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
efbc2a18 268000 3ed000 8000075 /lib/libc-2.12.so
efbc23d8 3ed000 3ee000 8000070 /lib/libc-2.12.so
efbc2888 3ee000 3f0000 8100071 /lib/libc-2.12.so
efbc2cd4 3f0000 3f1000 8100073 /lib/libc-2.12.so
```

```
efbc243c 3f1000 3f4000 100073
efbc28ec 3f6000 3f9000 8000075 /lib/libdl-2.12.so
efbc2568 3f9000 3fa000 8100071 /lib/libdl-2.12.so
efbc2f2c 3fa000 3fb000 8100073 /lib/libdl-2.12.so
f26af888 7e6000 7fc000 8000075 /lib/libtinfo.so.5.7
f26aff2c 7fc000 7ff000 8100073 /lib/libtinfo.so.5.7
efbc211c d83000 d8f000 8000075 /lib/libnss_files-2.12.so
efbc2504 d8f000 d90000 8100071 /lib/libnss_files-2.12.so
efbc2950 d90000 d91000 8100073 /lib/libnss_files-2.12.so
f26afe00 edc000 edd000 4040075
f1bb0a18 8047000 8118000 8001875 /bin/bash
f1bb01e4 8118000 811d000 8101873 /bin/bash
f1bb0c70 811d000 8122000 100073
f26afae0 9fd9000 9ffa000 100073
... several lines omitted ...
```

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

20.4. 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

- An **oops** message is secured 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 diagnose a kernel crash issue by providing a text message or a **vmcore-dmesg.txt** as an input.

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

Additional resources

- [The Kernel Oops Analyzer](#) article

20.5. 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](#)

CHAPTER 21. USING EARLY KDUMP TO CAPTURE BOOT TIME CRASHES

Early kdump is a feature of the **kdump** mechanism that captures the **vmcore** file if a system or kernel crash occurs during the early phases of the boot process before the system services start. Early kdump loads the crash kernel and the **initramfs** of crash kernel in the memory much earlier.

A kernel crash can sometimes occur during the early boot phase before the **kdump** service starts and is able to capture and save the contents of the crashed kernel memory. Therefore, crucial information related to the crash that is important for troubleshooting is lost. To address this problem, you can use the **early kdump** feature, which is a part of the **kdump** service.

21.1. ENABLING EARLY KDUMP

The **early kdump** feature sets up the crash kernel and the initial RAM disk image (**initramfs**) to load early enough to capture the **vmcore** information for an early crash. This helps to eliminate the risk of losing information about the early boot kernel crashes.

Prerequisites

- An active RHEL subscription.
- A repository containing the **kexec-tools** package for your system CPU architecture.
- Fulfilled **kdump** configuration and targets requirements. For more information see, [Supported kdump configurations and targets](#).

Procedure

1. Verify that the **kdump** service is enabled and active:

```
# systemctl is-enabled kdump.service && systemctl is-active kdump.service
enabled
active
```

If **kdump** is not enabled and running, set all required configurations and verify that **kdump** service is enabled.

2. Rebuild the **initramfs** image of the booting kernel with the **early kdump** functionality:

```
# dracut -f --add earlykdump
```

3. Add the **rd.earlykdump** kernel command line parameter:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="rd.earlykdump"
```

4. Reboot the system to reflect the changes:

```
# reboot
```

Verification

- Verify that **rd.earlykdump** is successfully added and **early kdump** feature is enabled:

cat /proc/cmdline

```
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-187.el8.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet rd.earlykdump
```

journalctl -x | grep early-kdump

```
Mar 20 15:44:41 redhat dracut-cmdline[304]: early-kdump is enabled.
```

```
Mar 20 15:44:42 redhat dracut-cmdline[304]: kexec: loaded early-kdump kernel
```

Additional resources

- The **/usr/share/doc/kexec-tools/early-kdump-howto.txt** file
- [What is early kdump support and how do I configure it?](#) (Red Hat Knowledgebase)

CHAPTER 22. APPLYING PATCHES WITH KERNEL LIVE PATCHING

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:

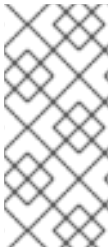
- Can immediately apply critical security patches to the kernel.
- Do not have to wait for long-running tasks to complete, for users to log off, or for scheduled downtime.
- Control the system's uptime more and do not sacrifice security or stability.

By using the kernel live patching, you can reduce the number of reboots required for security patches. However, note that you cannot address all critical or important CVEs. For more details about the scope of live patching, see the Red Hat Knowledgebase solution [Is live kernel patch \(kpatch\) supported in Red Hat Enterprise Linux?](#)



WARNING

Some incompatibilities exist between kernel live patching and other kernel subcomponents. Read the [Limitations of kpatch](#) carefully before using kernel live patching.



NOTE

For details about the support cadence of kernel live patching updates, see:

- [Kernel Live Patch Support Cadence Update](#)
- [Kernel Live Patch life cycles](#)

22.1. LIMITATIONS OF KPATCH

- By using the **kpatch** feature, you can apply simple security and bug fix updates that do not require an immediate system reboot.
- You must not use the **SystemTap** or **kprobe** tool during or after loading a patch. The patch might not take effect until the probes are removed.

22.2. 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 does not support live patches provided by a third party.

For more information about third-party software support policies, see [As a customer how does Red Hat support me when I use third party components?](#)

22.3. ACCESS TO KERNEL LIVE PATCHES

A kernel module (**kmod**) implements kernel live patching capability and is provided 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 8.2 kernel until the RHEL 8.3 kernel is released.

The components of kernel live patching are as follows:

Kernel patch module

- The delivery mechanism for kernel live patches.
- A kernel module built specifically for the kernel being patched.
- The patch module contains the code of the required fixes for the kernel.
- Patch modules register with the **livepatch** kernel subsystem and specify the original functions to replace, along with pointers to the replacement functions. Kernel patch modules are delivered as RPMs.
- The naming convention is **kpatch_<kernel version>_<kpatch version>_<kpatch release>**. The "kernel version" part of the name has *dots* replaced with *underscores*.

The kpatch utility

A command-line utility for managing patch modules.

The kpatch service

A **systemd** service required by **multiuser.target**. This target loads the kernel patch module at boot time.

The kpatch-dnf package

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

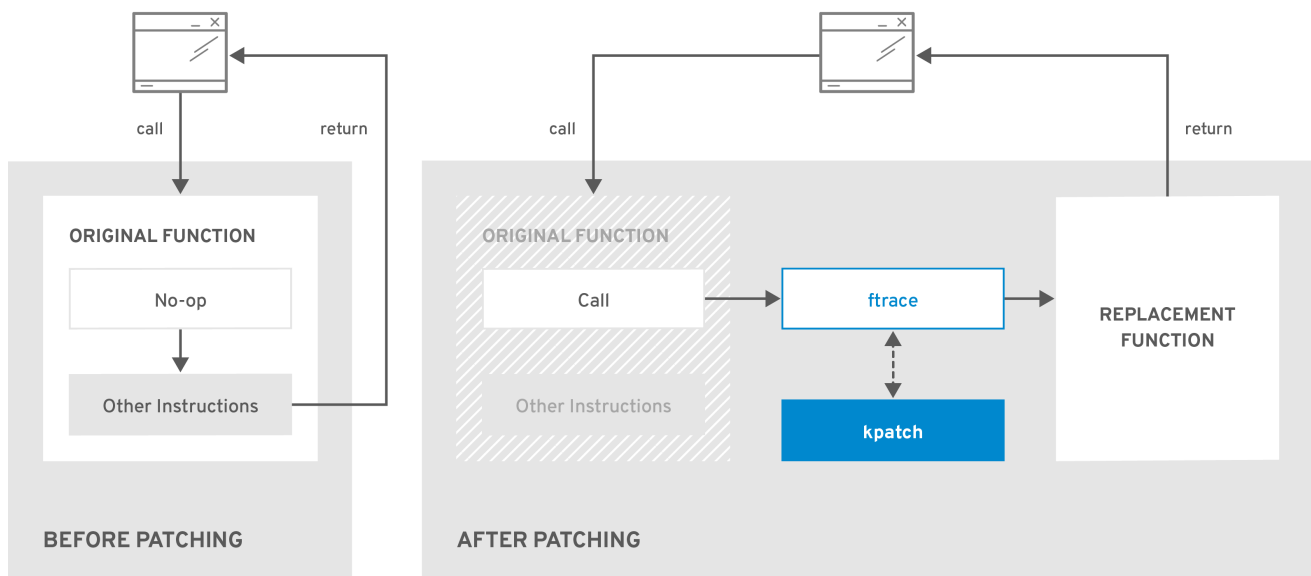
22.4. THE PROCESS OF LIVE PATCHING KERNELS

The **kpatch** kernel patching solution uses the **livepatch** kernel subsystem to redirect outdated functions to updated ones. Applying a live kernel patch to a system triggers the following processes:

1. The kernel patch module is copied to the **/var/lib/kpatch/** directory and registered for re-application to the kernel by **systemd** on next boot.
2. The **kpatch** module loads into the running kernel and the new functions are registered to the **ftrace** mechanism with a pointer to the location in memory of the new code.

When the kernel accesses the patched function, the **ftrace** mechanism redirects it, bypassing the original functions and leading the kernel to the patched version of the function.

Figure 22.1. How kernel live patching works



RHEL_424549_0119

22.5. 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

- You have root permissions.

Procedure

1. Optional: Check your kernel version:

```
# uname -r
4.18.0-94.el8.x86_64
```

2. Search for a live patching package that corresponds to the version of your kernel:

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

3. Install the live patching package:

```
# 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-4_18_0-94-0-0.el8.x86_64.rpm**. The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

Verification

- Verify that all installed kernels have been patched:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
...
```

The output shows that the kernel patch module has been loaded into the kernel that is now patched with the latest fixes from the **kpatch-patch-4_18_0-94-1-1.el8.x86_64.rpm** package.



NOTE

Entering the **kpatch list** command does not return an empty live patching package. Use the **rpm -qa | grep kpatch** command instead.

```
# rpm -qa | grep kpatch
kpatch-patch-4_18_0-477_21_1-0-0.el8_8.x86_64
kpatch-dnf-0.9.7_0.4-2.el8.noarch
kpatch-0.9.7-2.el8.noarch
```

Additional resources

- **kpatch(1)** manual page

22.6. AUTOMATICALLY SUBSCRIBING ANY FUTURE KERNEL TO THE LIVE PATCHING STREAM

You can use the **kpatch-dnf** YUM 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

- You have root permissions.

Procedure

1. Optional: 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      4.18.0-240.10.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.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: 19:10:26 ago on Wed 10 Mar 2021 04:08:06 PM CET.
Dependencies resolved.
=====
Package                               Architecture
=====
Installing:
kpatch-patch-4_18_0-240_10_1          x86_64
kpatch-patch-4_18_0-240_15_1          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.

When you update the kernel, live patches are installed automatically 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-4_18_0-240-0-0.el8.x86_64.rpm**. The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

Verification

- Verify that all installed kernels are patched:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_240_10_1_0_1 [enabled]

Installed patch modules:
kpatch_4_18_0_240_10_1_0_1 (4.18.0-240.10.1.el8_3.x86_64)
kpatch_4_18_0_240_15_1_0_2 (4.18.0-240.15.1.el8_3.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-4_18_0-240_10_1-0-1.rpm** and **kpatch-patch-4_18_0-240_15_1-0-1.rpm** packages respectively.



NOTE

Entering the **kpatch list** command does not return an empty live patching package. Use the **rpm -qa | grep kpatch** command instead.

```
# rpm -qa | grep kpatch
kpatch-patch-4_18_0-477_21_1-0-0.el8_8.x86_64
kpatch-dnf-0.9.7_0.4-2.el8.noarch
kpatch-0.9.7-2.el8.noarch
```

Additional resources

- **kpatch(1)** and **dnf-kpatch(8)** manual pages

22.7. DISABLING AUTOMATIC SUBSCRIPTION TO THE LIVE PATCHING STREAM

When you subscribe your system to fixes delivered by the kernel patch module, your subscription is **automatic**. You can disable this feature, to disable automatic installation of **kpatch-patch** packages.

Prerequisites

- You have root permissions.

Procedure

1. Optional: 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      4.18.0-240.10.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. Disable automatic subscription to kernel live patches:

```
# yum kpatch manual
Updating Subscription Management repositories.
```

Verification

- You can check for the successful outcome:

```
# yum kpatch status
...
Updating Subscription Management repositories.
Last metadata expiration check: 0:30:41 ago on Tue Jun 14 15:59:26 2022.
Kpatch update setting: manual
```

Additional resources

- **kpatch(1)** and **dnf-kpatch(8)** manual pages

22.8. UPDATING KERNEL PATCH MODULES

The kernel patch modules are delivered and applied through RPM packages. The process of updating a cumulative kernel patch module is similar to 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"
```



NOTE

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

22.9. REMOVING THE LIVE PATCHING PACKAGE

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-4_18_0-94.x86_64      1-1.el8      @@commandline
...
```

The example output lists live patching packages that you installed.

2. Remove the live patching package.

```
# yum remove kpatch-patch-4_18_0-94.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 the live patching package is removed:

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

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

Verification

1. Verify 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

- The **kpatch(1)** manual page
- [Uninstalling software packages](#) in RHEL

22.10. UNINSTALLING THE KERNEL PATCH MODULE

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_4_18_0_94_1_1 [enabled]

Installed patch modules:

kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)

...

2. Uninstall the selected kernel patch module.

kpatch uninstall kpatch_4_18_0_94_1_1

uninstalling kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)

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

kpatch list

Loaded patch modules:

kpatch_4_18_0_94_1_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.

Verification

1. Verify that the kernel patch module is uninstalled:

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

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

Additional resources

- The **kpatch(1)** manual page

22.11. DISABLING KPATCH.SERVICE

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.

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

2. Disable **kpatch.service**.

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

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

```
# kpatch list  
Loaded patch modules:  
kpatch_4_18_0_94_1_1 [enabled]
```



```

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)

```

3. Reboot your system.
4. Optional: Verify the status of **kpatch.service**.

```
# 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** is disabled. Thereby, the kernel live patching solution is not active.

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

```
# kpatch list
```

```

Loaded patch modules:
<NO_RESULT>

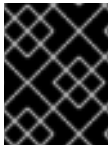
```

```

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.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

- The **kpatch(1)** manual page.
- [Managing systemd](#)

CHAPTER 23. SETTING SYSTEM RESOURCE LIMITS FOR APPLICATIONS BY USING CONTROL GROUPS

Using the control groups (**cgroups**) kernel functionality, you can control resource usage of applications to use them more efficiently.

You can use **cgroups** for the following tasks:

- Setting limits for system resource allocation.
- Prioritizing the allocation of hardware resources to specific processes.
- Isolating certain processes from obtaining hardware resources.

23.1. INTRODUCING CONTROL GROUPS

Using the *control groups* Linux kernel feature, you can organize processes into hierarchically ordered groups - **cgroups**. You define the hierarchy (control groups tree) by providing structure to **cgroups** virtual file system, mounted by default on the **/sys/fs/cgroup/** directory.

The **systemd** service manager uses **cgroups** to organize all units and services that it governs. Manually, you can manage the hierarchies of **cgroups** by creating and removing sub-directories in the **/sys/fs/cgroup/** directory.

The resource controllers in the kernel then modify the behavior of processes in **cgroups** by limiting, prioritizing or allocating system resources, of those processes. These resources include the following:

- CPU time
- Memory
- Network bandwidth
- Combinations of these resources

The primary use case of **cgroups** is aggregating system processes and dividing hardware resources among applications and users. This makes it possible to increase the efficiency, stability, and security of your environment.

Control groups version 1

Control groups version 1 (**cgroups-v1**) provide a per-resource controller hierarchy. Each resource, such as CPU, memory, or I/O, has its own control group hierarchy. You can combine different control group hierarchies in a way that one controller can coordinate with another in managing their respective resources. However, when the two controllers belong to different process hierarchies, the coordination is limited.

The **cgroups-v1** controllers were developed across a large time span, resulting in inconsistent behavior and naming of their control files.

Control groups version 2

Control groups version 2 (**cgroups-v2**) provide a single control group hierarchy against which all resource controllers are mounted.

The control file behavior and naming is consistent among different controllers.

**NOTE**

cgroups-v2 is fully supported in RHEL 8.2 and later versions. For more information, see [Control Group v2 is now fully supported in RHEL 8](#).

Additional resources

- [Introducing kernel resource controllers](#)
- The **cgroups(7)** manual page
- [Role of systemd in control groups](#)

23.2. INTRODUCING KERNEL RESOURCE CONTROLLERS

Kernel resource controllers enable the functionality of control groups. RHEL 8 supports various controllers for *control groups version 1* (**cgroups-v1**) and *control groups version 2* (**cgroups-v2**).

A resource controller, also called a control group subsystem, is a kernel subsystem that represents a single resource, such as CPU time, memory, network bandwidth or disk I/O. The Linux kernel provides a range of resource controllers that are mounted automatically by the **systemd** service manager. You can find a list of the currently mounted resource controllers in the **/proc/cgroups** file.

Controllers available for cgroups-v1:**blkio**

Sets limits on input/output access to and from block devices.

cpu

Adjusts the parameters of the Completely Fair Scheduler (CFS) for a control group's tasks. The **cpu** controller is mounted together with the **cpuacct** controller on the same mount.

cpuacct

Creates automatic reports on CPU resources used by tasks in a control group. The **cpuacct** controller is mounted together with the **cpu** controller on the same mount.

cpuset

Restricts control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes.

devices

Controls access to devices for tasks in a control group.

freezer

Suspends or resumes tasks in a control group.

memory

Sets limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.

net_cls

Tags network packets with a class identifier (**classid**) that enables the Linux traffic controller (the **tc** command) to identify packets that originate from a particular control group task. A subsystem of **net_cls**, the **net_filter** (iptables), can also use this tag to perform actions on such packets. The **net_filter** tags network sockets with a firewall identifier (**fwid**) that allows the Linux firewall to identify packets that originate from a particular control group task (by using the **iptables** command).

net_prio

Sets the priority of network traffic.

pids

Sets limits for multiple processes and their children in a control group.

perf_event

Groups tasks for monitoring by the **perf** performance monitoring and reporting utility.

rdma

Sets limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.

hugetlb

Limits the usage of large size virtual memory pages by tasks in a control group.

Controllers available for cgroups-v2:**io**

Sets limits on input/output access to and from block devices.

memory

Sets limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.

pids

Sets limits for multiple processes and their children in a control group.

rdma

Sets limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.

cpu

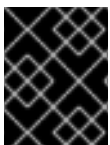
Adjusts the parameters of the Completely Fair Scheduler (CFS) for a control group's tasks and creates automatic reports on CPU resources used by tasks in a control group.

cpuset

Restricts control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes. Supports only the core functionality (**cpus{,.effective}**, **mems{,.effective}**) with a new partition feature.

perf_event

Groups tasks for monitoring by the **perf** performance monitoring and reporting utility. **perf_event** is enabled automatically on the v2 hierarchy.

**IMPORTANT**

A resource controller can be used either in a **cgroups-v1** hierarchy or a **cgroups-v2** hierarchy, not simultaneously in both.

Additional resources

- The **cgroups(7)** manual page
- Documentation in **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** directory (after installing the **kernel-doc** package).

23.3. INTRODUCING NAMESPACES

Namespaces create separate spaces for organizing and identifying software objects. This keeps them from affecting each other. As a result, each software object contains its own set of resources, for example, a mount point, a network device, or a hostname, even though they are sharing the same system.

One of the most common technologies that use namespaces are containers.

Changes to a particular global resource are visible only to processes in that namespace and do not affect the rest of the system or other namespaces.

To inspect which namespaces a process is a member of, you can check the symbolic links in the `/proc/<PID>/ns/` directory.

Table 23.1. Supported namespaces and resources which they isolate:

Namespace	Isolates
Mount	Mount points
UTS	Hostname and NIS domain name
IPC	System V IPC, POSIX message queues
PID	Process IDs
Network	Network devices, stacks, ports, etc
User	User and group IDs
Control groups	Control group root directory

Additional resources

- The **namespaces(7)** and **cgroup_namespaces(7)** manual pages

23.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1

To configure CPU limits to an application by using *control groups version 1* (**cgroups-v1**), use the **/sys/fs/** virtual file system.

Prerequisites

- You have root permissions.
- You have an application to restrict its CPU consumption installed on your system.
- You verified that the **cgroups-v1** controllers are mounted:

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/systemd-
```

```

cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,pids)
...

```

Procedure

1. Identify the process ID (PID) of the application that you want to restrict in CPU consumption:

```

# top
top - 11:34:09 up 11 min, 1 user, load average: 0.51, 0.27, 0.22
Tasks: 267 total, 3 running, 264 sleeping, 0 stopped, 0 zombie
%Cpu(s): 49.0 us, 3.3 sy, 0.0 ni, 47.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.0 st
MiB Mem : 1826.8 total, 303.4 free, 1046.8 used, 476.5 buff/cache
MiB Swap: 1536.0 total, 1396.0 free, 140.0 used. 616.4 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 6955 root        20   0 228440  1752  1472 R   99.3   0.1   0:32.71 sha1sum
 5760 jdoe        20   0 3603868 205188 64196 S    3.7  11.0   0:17.19 gnome-shell
 6448 jdoe        20   0 743648  30640 19488 S    0.7   1.6   0:02.73 gnome-terminal-
    1 root        20   0 245300   6568  4116 S    0.3   0.4   0:01.87 systemd
 505 root        20   0     0     0     0 I   0.3   0.0   0:00.75 kworker/u4:4-events_unbound
...

```

The **sha1sum** example application with **PID 6955** consumes a large amount of CPU resources.

2. Create a sub-directory in the **cpu** resource controller directory:

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

This directory represents a control group, where you can place specific processes and apply certain CPU limits to the processes. At the same time, a number of **cgroups-v1** interface files and **cpu** controller-specific files will be created in the directory.

3. Optional: Inspect the newly created control group:

```

# ll /sys/fs/cgroup/cpu/Example/
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.shares

```

```
-r--r--r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r--r--. 1 root root 0 Mar 11 11:42 tasks
```

Files, such as **cpuacct.usage**, **cpu.cfs._period_us** represent specific configurations and/or limits, which can be set for processes in the **Example** control group. Note that the file names are prefixed with the name of the control group controller they belong to.

By default, the newly created control group inherits access to the system's entire CPU resources without a limit.

4. Configure CPU limits for the control group:

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

- The **cpu.cfs_period_us** file represents how frequently a control group's access to CPU resources must be reallocated. The time period is in microseconds (μ s, "us"). The upper limit is 1 000 000 microseconds and the lower limit is 1000 microseconds.
- The **cpu.cfs_quota_us** file represents the total amount of time in microseconds for which all processes in a control group can collectively run during one period, as defined by **cpu.cfs_period_us**. When processes in a control group use up all the time specified by the quota during a single period, they are throttled for the remainder of the period and not allowed to run until the next period. The lower limit is 1000 microseconds.
The example commands above set the CPU time limits so that all processes collectively in the **Example** control group will be able to run only for 0.2 seconds (defined by **cpu.cfs_quota_us**) out of every 1 second (defined by **cpu.cfs_period_us**).

5. Optional: Verify the limits:

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. Add the application's PID to the **Example** control group:

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

This command ensures that a specific application becomes a member of the **Example** control group and does not exceed the CPU limits configured for the **Example** control group. The PID must represent an existing process in the system. The **PID 6955** here was assigned to the **sha1sum /dev/zero &** process, used to illustrate the use case of the **cpu** controller.

Verification

1. Verify that the application runs in the specified control group:

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
```

```

7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-
server.service

```

The process of an application runs in the **Example** control group applying CPU limits to the application's process.

2. Identify the current CPU consumption of your throttled application:

```

# top
top - 12:28:42 up 1:06, 1 user, load average: 1.02, 1.02, 1.00
Tasks: 266 total, 6 running, 260 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.0 us, 1.2 sy, 0.0 ni, 87.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.2 st
MiB Mem : 1826.8 total, 287.1 free, 1054.4 used, 485.3 buff/cache
MiB Swap: 1536.0 total, 1396.7 free, 139.2 used. 608.3 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R  20.6  0.1 47:11.43 sha1sum
 5760 jdoe    20  0 3604956 208832 65316 R  2.3 11.2  0:43.50 gnome-shell
 6448 jdoe    20  0 743836 31736 19488 S  0.7  1.7  0:08.25 gnome-terminal-
 505 root    20  0    0    0    0 I  0.3  0.0  0:03.39 kworker/u4:4-events_unbound
 4217 root    20  0 74192 1612 1320 S  0.3  0.1  0:01.19 spice-vdagentd
...

```

Note that the CPU consumption of the **PID 6955** has decreased from 99% to 20%.



NOTE

The **cgroups-v2** counterpart for **cpu.cfs_period_us** and **cpu.cfs_quota_us** is the **cpu.max** file. The **cpu.max** file is available through the **cpu** controller.

Additional resources

- [Introducing kernel resource controllers](#)
- **cgroups(7)**, **sysfs(5)** manual pages

CHAPTER 24. USING CGROUPS-V2 TO CONTROL DISTRIBUTION OF CPU TIME FOR APPLICATIONS

Some applications use too much CPU time, which can negatively impact the overall health of your environment. You can put your applications into *control groups version 2* (**cgroups-v2**) and configure CPU limits for those control groups. As a result, you can regulate your applications in CPU consumption.

The user has two methods how to regulate distribution of CPU time allocated to a control group:

- Setting **CPU bandwidth** (editing the **cpu.max** controller file)
- Setting **CPU weight** (editing the **cpu.weight** controller file)

24.1. MOUNTING CGROUPS-V2

During the boot process, RHEL 8 mounts the **cgroup-v1** virtual filesystem by default. To utilize **cgroup-v2** functionality in limiting resources for your applications, manually configure the system. You use **systemd** to control the usage of system resources. You should manually configure the cgroups virtual file system only in special cases. For example, when you need to use cgroup-v1 controllers that have no equivalents in cgroup-v2 hierarchy.

Prerequisites

- You have root permissions.

Procedure

1. Configure the system to mount **cgroups-v2** by default during system boot by the **systemd** system and service manager:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=1"
```

This adds the necessary kernel command-line parameter to the current boot entry.

To add the **systemd.unified_cgroup_hierarchy=1** parameter to all kernel boot entries:

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=1"
```

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

Verification

1. Verify the **cgroups-v2** filesystem is mounted:

```
# mount -l | grep cgroup
cgroup2 on /sys/fs/cgroup type cgroup2
(rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate)
```

The **cgroups-v2** filesystem was successfully mounted on the **/sys/fs/cgroup/** directory.

2. Inspect the contents of the **/sys/fs/cgroup/** directory:

```
# ll /sys/fs/cgroup/
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.controllers
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.depth
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.procs
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.stat
-rw-r--r--. 1 root root 0 Apr 29 12:18 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.threads
-rw-r--r--. 1 root root 0 Apr 29 12:03 cpu.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.cpus.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.mems.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpu.stat
drwxr-xr-x. 2 root root 0 Apr 29 12:03 init.scope
-rw-r--r--. 1 root root 0 Apr 29 12:03 io.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 io.stat
-rw-r--r--. 1 root root 0 Apr 29 12:03 memory.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 memory.stat
drwxr-xr-x. 69 root root 0 Apr 29 12:03 system.slice
drwxr-xr-x. 3 root root 0 Apr 29 12:18 user.slice
```

The `/sys/fs/cgroup/` directory, also called the *root control group*, by default, provides interface files (starting with **cgroup**) and controller-specific files such as **cpuset.cpus.effective**. In addition, there are some directories related to **systemd**, such as, `/sys/fs/cgroup/init.scope`, `/sys/fs/cgroup/system.slice`, and `/sys/fs/cgroup/user.slice`.

Additional resources

- **cgroups(7)**, **sysfs(5)** manual pages

24.2. PREPARING THE CGROUP FOR DISTRIBUTION OF CPU TIME

To control CPU consumption of your applications, you need to enable specific CPU controllers and create a dedicated control groups. It is recommended to create at least two levels of child control groups inside the `/sys/fs/cgroup/` root control group to keep organizational clarity of **cgroup** files.

Prerequisites

- You have root permissions.
- You have identified PIDs of processes that you want to control.
- You have mounted the **cgroups-v2** file system. For more information, see [Mounting cgroups-v2](#).

Procedure

1. Identify the process IDs (PIDs) of applications whose CPU consumption you want to constrict:

```
# top
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 17.6 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.8 hi, 0.0 si, 0.0 st
MiB Mem : 3737.4 total, 3312.7 free, 133.3 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.1 avail Mem
```

```

PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
34578 root        20   0 18720 1756 1468 R 99.0   0.0   0:31.09 sha1sum
34579 root        20   0 18720 1772 1480 R 99.0   0.0   0:30.54 sha1sum
   1 root        20   0 186192 13940 9500 S  0.0   0.4   0:01.60 systemd
   2 root        20   0     0     0     0 S  0.0   0.0   0:00.01 kthreadd
   3 root         0 -20     0     0     0 I  0.0   0.0   0:00.00 rcu_gp
   4 root         0 -20     0     0     0 I  0.0   0.0   0:00.00 rcu_par_gp
...

```

The example output reveals that **PID 34578** and **34579** (two illustrative applications of **sha1sum**) consume a huge amount of resources, namely CPU. Both are the example applications used to demonstrate managing the **cgroups-v2** functionality.

2. Verify that the **cpu** and **cpuset** controllers are available in the **/sys/fs/cgroup/cgroup.controllers** file:

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

3. Enable CPU-related controllers:

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

These commands enable the **cpu** and **cpuset** controllers for the immediate children groups of the **/sys/fs/cgroup/** root control group. A *child group* is where you can specify processes and apply control checks to each of the processes based on your criteria.

You can review the **cgroup.subtree_control** file at any level to identify the controllers that can be enabled in the immediate child group.



NOTE

By default, the **/sys/fs/cgroup/cgroup.subtree_control** file in the root control group contains **memory** and **pids** controllers.

4. Create the **/sys/fs/cgroup/Example/** directory:

```
# mkdir /sys/fs/cgroup/Example/
```

The **/sys/fs/cgroup/Example/** directory defines a child group. Also, the previous step enabled the **cpu** and **cpuset** controllers for this child group.

When you create the **/sys/fs/cgroup/Example/** directory, some **cgroups-v2** interface files and **cpu** and **cpuset** controller-specific files are automatically created in the directory. The **/sys/fs/cgroup/Example/** directory also provides controller-specific files for the **memory** and **pids** controllers.

5. Optional: Inspect the newly created child control group:

```
# ll /sys/fs/cgroup/Example/
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
```

```
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.procs
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
...
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun  1 10:33 cpu.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight.nice
...
-r--r--r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.low
...
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.current
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 pids.max
```

The example output shows files such as **cpuset.cpus** and **cpu.max**. These files are specific to the **cpuset** and **cpu** controllers. The **cpuset** and **cpu** controllers are manually enabled for the root's (**/sys/fs/cgroup/**) *direct child control groups* using the **/sys/fs/cgroup/cgroup.subtree_control** file.

The directory also includes general **cgroup** control interface files such as **cgroup.procs** or **cgroup.controllers**, which are common to all control groups, regardless of enabled controllers.

The files such as **memory.high** and **pids.max** relate to the **memory** and **pids** controllers, which are in the root control group (**/sys/fs/cgroup/**), and are always enabled by default.

By default, the newly created child group inherits access to all of the system's CPU and memory resources, without any limits.

6. Enable the CPU-related controllers in **/sys/fs/cgroup/Example/** to obtain controllers that are relevant only to CPU:

```
# echo "+cpu" >> /sys/fs/cgroup/Example/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

These commands ensure that the immediate child control group will *only* have controllers relevant to regulate the CPU time distribution - not to **memory** or **pids** controllers.

7. Create the **/sys/fs/cgroup/Example/tasks/** directory:

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

The **/sys/fs/cgroup/Example/tasks/** directory defines a child group with files that relate purely to **cpu** and **cpuset** controllers.

8. Optional: Inspect another child control group:

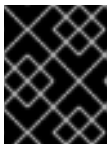
```
# ll /sys/fs/cgroup/Example/tasks
```

```
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.procs
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.stat
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.threads
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.type
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.max
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.pressure
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus
-r--r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.mems
-r--r--r--. 1 root root 0 Jun  1 11:45 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun  1 11:45 cpu.stat
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.weight
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.weight.nice
-rw-r--r--. 1 root root 0 Jun  1 11:45 io.pressure
-rw-r--r--. 1 root root 0 Jun  1 11:45 memory.pressure
```

9. Ensure the processes that you want to control for CPU time compete on the same CPU:

```
# echo "1" > /sys/fs/cgroup/Example/tasks/cpuset.cpus
```

This ensures the processes you will place in the **Example/tasks** child control group, compete on the same CPU. This setting is important for the **cpu** controller to activate.



IMPORTANT

The **cpu** controller is only activated if the relevant child control group has at least 2 processes to compete for time on a single CPU.

Verification

1. Optional: Ensure the CPU-related controllers are enabled for the immediate children cgroups:

```
# cat /sys/fs/cgroup/cgroup.subtree_control
/sys/fs/cgroup/Example/cgroup.subtree_control
cpuset cpu memory pids
cpuset cpu
```

2. Optional: Ensure the processes that you want to control for CPU time compete on the same CPU:

```
# cat /sys/fs/cgroup/Example/tasks/cpuset.cpus
1
```

Additional resources

- [Introducing control groups](#)

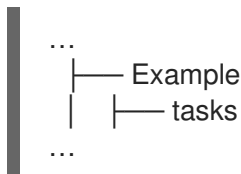
- [Introducing kernel resource controllers](#)
- [Mounting cgroups-v2](#)
- **cgroups(7), sysfs(5)** manual pages

24.3. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU BANDWIDTH

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites

- You have root permissions.
- You have at least two applications for which you want to control distribution of CPU time.
- You ensured the relevant applications compete for CPU time on the same CPU as described in [Preparing the cgroup for distribution of CPU time](#).
- You mounted **cgroups-v2** filesystem as described in [Mounting cgroups-v2](#).
- You enabled **cpu** and **cpuset** controllers both in the parent control group and in child control group similarly as described in [Preparing the cgroup for distribution of CPU time](#).
- You created two levels of *child control groups* inside the **/sys/fs/cgroup/** root control group as in the example below:



Procedure

1. Configure CPU bandwidth to achieve resource restrictions within the control group:

```
# echo "200000 1000000" > /sys/fs/cgroup/Example/tasks/cpu.max
```

The first value is the allowed time quota in microseconds for which all processes collectively in a child group can run during one period. The second value specifies the length of the period.

During a single period, when processes in a control group collectively exhaust the time specified by this quota, they are throttled for the remainder of the period and not allowed to run until the next period.

This command sets CPU time distribution controls so that all processes collectively in the **/sys/fs/cgroup/Example/tasks** child group can run on the CPU for only 0.2 seconds of every 1 second. That is, one fifth of each second.

2. Optional: Verify the time quotas:

```
# cat /sys/fs/cgroup/Example/tasks/cpu.max
200000 1000000
```

3. Add the applications' PIDs to the **Example/tasks** child group:

```
# echo "34578" > /sys/fs/cgroup/Example/tasks/cgroup.procs
# echo "34579" > /sys/fs/cgroup/Example/tasks/cgroup.procs
```

The example commands ensure that required applications become members of the **Example/tasks** child group and do not exceed the CPU time distribution configured for this child group.

Verification

1. Verify that the applications run in the specified control group:

```
# cat /proc/34578/cgroup /proc/34579/cgroup
0::/Example/tasks
0::/Example/tasks
```

The output above shows the processes of the specified applications that run in the **Example/tasks** child group.

2. Inspect the current CPU consumption of the throttled applications:

```
# top
top - 11:13:53 up 23:10, 1 user, load average: 0.26, 1.33, 1.66
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.0 us, 7.0 sy, 0.0 ni, 89.5 id, 0.0 wa, 0.2 hi, 0.2 si, 0.2 st
MiB Mem : 3737.4 total, 3312.6 free, 133.4 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.0 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 34578 root    20  0 18720 1756 1468 R  10.0  0.0 37:36.13 sha1sum
 34579 root    20  0 18720 1772 1480 R  10.0  0.0 37:41.22 sha1sum
    1 root    20  0 186192 13940 9500 S   0.0  0.4 0:01.60 systemd
    2 root    20  0     0     0     0 S   0.0  0.0 0:00.01 kthreadd
    3 root     0 -20     0     0     0 I   0.0  0.0 0:00.00 rcu_gp
    4 root     0 -20     0     0     0 I   0.0  0.0 0:00.00 rcu_par_gp
...
```

Notice that the CPU consumption for the **PID 34578** and **PID 34579** has decreased to 10%. The **Example/tasks** child group regulates its processes to 20% of the CPU time collectively. Since there are 2 processes in the control group, each can utilize 10% of the CPU time.

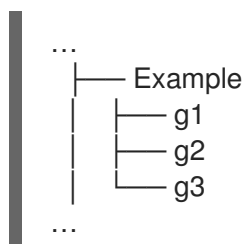
24.4. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites

- You have root permissions.
- You have applications for which you want to control distribution of CPU time.

- You ensured the relevant applications compete for CPU time on the same CPU as described in [Preparing the cgroup for distribution of CPU time](#).
- You mounted **cgroups-v2** filesystem as described in [Mounting cgroups-v2](#).
- You created a two level hierarchy of *child control groups* inside the **/sys/fs/cgroup/** root control group as in the following example:



- You enabled **cpu** and **cpuset** controllers in the parent control group and in child control groups similarly as described in [Preparing the cgroup for distribution of CPU time](#).

Procedure

1. Configure desired CPU weights to achieve resource restrictions within the control groups:

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. Add the applications' PIDs to the **g1**, **g2**, and **g3** child groups:

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

The example commands ensure that desired applications become members of the **Example/g*** child cgroups and will get their CPU time distributed as per the configuration of those cgroups.

The weights of the children cgroups (**g1**, **g2**, **g3**) that have running processes are summed up at the level of the parent cgroup (**Example**). The CPU resource is then distributed proportionally based on the respective weights.

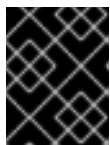
As a result, when all processes run at the same time, the kernel allocates to each of them the proportionate CPU time based on their respective cgroup's **cpu.weight** file:

Child cgroup	cpu.weight file	CPU time allocation
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

The value of the **cpu.weight** controller file is not a percentage.

If one process stopped running, leaving cgroup **g2** with no running processes, the calculation would omit the cgroup **g2** and only account weights of cgroups **g1** and **g3**:

Child cgroup	cpu.weight file	CPU time allocation
g1	150	~75% (150/200)
g3	50	~25% (50/200)



IMPORTANT

If a child cgroup has multiple running processes, the CPU time allocated to the cgroup is distributed equally among its member processes.

Verification

1. Verify that the applications run in the specified control groups:

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

The command output shows the processes of the specified applications that run in the **Example/g*** child cgroups.

2. Inspect the current CPU consumption of the throttled applications:

```
# top
top - 05:17:18 up 1 day, 18:25, 1 user, load average: 3.03, 3.03, 3.00
Tasks: 95 total, 4 running, 91 sleeping, 0 stopped, 0 zombie
%Cpu(s): 18.1 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st
MiB Mem : 3737.0 total, 3233.7 free, 132.8 used, 370.5 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3373.1 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 33373 root    20  0 18720 1748 1460 R  49.5  0.0 415:05.87 sha1sum
 33374 root    20  0 18720 1756 1464 R  32.9  0.0 412:58.33 sha1sum
 33377 root    20  0 18720 1860 1568 R  16.3  0.0 411:03.12 sha1sum
   760 root    20  0 416620 28540 15296 S   0.3  0.7  0:10.23 tuned
     1 root    20  0 186328 14108 9484 S   0.0  0.4  0:02.00 systemd
     2 root    20  0   0     0   0 S   0.0  0.0  0:00.01 kthread
...
```



NOTE

All processes run on a single CPU for clear illustration. The CPU weight applies the same principles when used on multiple CPUs.

Notice that the CPU resource for the **PID 33373**, **PID 33374**, and **PID 33377** was allocated based on the 150, 100, and 50 weights you assigned to the respective child cgroups. The weights correspond to around 50%, 33%, and 16% allocation of CPU time for each application.

CHAPTER 25. USING CONTROL GROUPS VERSION 1 WITH SYSTEMD

You can manage **cgroups** with the **systemd** system and service manager and the utilities they provide. This is also the preferred way of the **cgroups** management.

25.1. ROLE OF SYSTEMD IN CONTROL GROUPS VERSION 1

RHEL 8 moves the resource management settings from the process level to the application level by binding the system of **cgroup** hierarchies with the **systemd** unit tree. Therefore, you can manage the system resources with the **systemctl** command, or by modifying the **systemd** unit files.

By default, the **systemd** system and service manager use the **slice**, **scope** and **service** units to organize and structure processes in the control groups. The **systemctl** command can further modify this structure by creating custom **slices**. **systemd** also automatically mounts hierarchies for important kernel resource controllers in the **/sys/fs/cgroup/** directory.

Three **systemd** unit types are used for resource control:

- **Service** – A process or a group of processes, which **systemd** started according to a unit configuration file. Services encapsulate the specified processes to be started and stopped as one set. Services are named in the following way:

```
<name>.service
```

- **Scope** – A group of externally created processes. Scopes encapsulate processes that are started and stopped by the arbitrary processes through the **fork()** function and then registered by **systemd** at runtime. For example, user sessions, containers, and virtual machines are treated as scopes. Scopes are named as follows:

```
<name>.scope
```

- **Slice** – A group of hierarchically organized units. Slices organize a hierarchy in which scopes and services are placed. The actual processes are included in scopes or in services. Every name of a slice unit corresponds to the path to a location in the hierarchy. The dash ("-") character acts as a separator of the path components to a slice from the **-.slice** root slice. In the following example, services and scopes that contain processes are placed in slices that do not have processes of their own:

```
<parent-name>.slice
```

parent-name.slice is a sub-slice of **parent.slice**, which is a sub-slice of the **-.slice** root slice. **parent-name.slice** can have its own sub-slice named **parent-name-name2.slice**, and so on.

The **service**, the **scope**, and the **slice** units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths built from the unit names.

Example of a control group hierarchy

The services and scopes containing processes are placed in slices that do not have processes of their own.

```
Control group /:
-.slice
```

```

└─user.slice
   └─user-42.slice
      └─session-c1.scope
         └─967 gdm-session-worker [pam/gdm-launch-environment]
            └─1035 /usr/libexec/gdm-x-session gnome-session --autostart
               /usr/share/gdm/greeter/autostart
                  └─1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -background none
                     -noreset -keeptty -verbose 3
                        └─1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
                           └─1369 /usr/bin/gnome-shell
                              └─1732 ibus-daemon --xim --panel disable
                                 └─1752 /usr/libexec/ibus-dconf
                                    └─1762 /usr/libexec/ibus-x11 --kill-daemon
                                       └─1912 /usr/libexec/gsd-xsettings
                                          └─1917 /usr/libexec/gsd-a11y-settings
                                             └─1920 /usr/libexec/gsd-clipboard
...
└─init.scope
   └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
   └─rngd.service
      └─800 /sbin/rngd -f
   └─systemd-udevd.service
      └─659 /usr/lib/systemd/systemd-udevd
   └─chronyd.service
      └─823 /usr/sbin/chronyd
   └─auditd.service
      └─761 /sbin/auditd
         └─763 /usr/sbin/sedispatch
   └─accounts-daemon.service
      └─876 /usr/libexec/accounts-daemon
   └─example.service
      └─929 /bin/bash /home/jdoe/example.sh
         └─4902 sleep 1
...

```

Additional resources

- **systemd.resource-control(5), cgroups(7), fork(), fork(2)** manual pages

25.2. CREATING TRANSIENT CONTROL GROUPS

The transient **cgroups** set limits on resources consumed by a unit (service or scope) during its runtime.

Procedure

- To create a transient control group, use the **systemd-run** command in the following format:

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

This command creates and starts a transient service or a scope unit and runs a custom command in such a unit.

- The **--unit=<name>** option gives a name to the unit. If **--unit** is not specified, the name is generated automatically.

- The **--slice=<name>.slice** option makes your service or scope unit a member of a specified slice. Replace **<name>.slice** with the name of an existing slice (as shown in the output of **systemctl -t slice**), or create a new slice by passing a unique name. By default, services and scopes are created as members of the **system.slice**.
- Replace **<command>** with the command you want to enter in the service or the scope unit. The following message is displayed to confirm that you created and started the service or the scope successfully:

```
# Running as unit <name>.service
```

- *Optional:* Keep the unit running after its processes finished to collect runtime information:

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

The command creates and starts a transient service unit and runs a custom command in the unit. The **--remain-after-exit** option ensures that the service keeps running after its processes have finished.

Additional resources

- The **systemd-run(1)** manual page

25.3. CREATING PERSISTENT CONTROL GROUPS

To assign a persistent control group to a service, you need to edit its unit configuration file. The configuration is preserved after the system reboot to manage services that started automatically.

Procedure

- To create a persistent control group, enter:

```
# systemctl enable <name>.service
```

This command automatically creates a unit configuration file into the **/usr/lib/systemd/system/** directory and by default, it assigns **<name>.service** to the **system.slice** unit.

Additional resources

- **systemd-run(1)** manual page

25.4. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS ON THE COMMAND-LINE

Executing commands on the command line is one of the ways how to set limits, prioritize, or control access to hardware resources for groups of processes.

Procedure

- To limit the memory usage of a service, run the following:

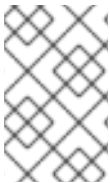
```
# systemctl set-property example.service MemoryMax=1500K
```

The command instantly assigns the memory limit of 1,500 KB to processes executed in a control group the **example.service** service belongs to. The **MemoryMax** parameter, in this configuration variant, is defined in the `/etc/systemd/system.control/example.service.d/50-MemoryMax.conf` file and controls the value of the `/sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes` file.

- Optionally, to temporarily limit the memory usage of a service, run:

```
# systemctl set-property --runtime example.service MemoryMax=1500K
```

The command instantly assigns the memory limit to the **example.service** service. The **MemoryMax** parameter is defined until the next reboot in the `/run/systemd/system.control/example.service.d/50-MemoryMax.conf` file. With a reboot, the whole `/run/systemd/system.control/` directory and **MemoryMax** are removed.



NOTE

The **50-MemoryMax.conf** file stores the memory limit as a multiple of 4096 bytes – one kernel page size specific for AMD64 and Intel 64. The actual number of bytes depends on a CPU architecture.

Additional resources

- **systemd.resource-control(5)** and **cgroups(7)** manual pages
- [Role of systemd in control groups](#)

25.5. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS WITH UNIT FILES

Each persistent unit is supervised by the **systemd** system and service manager, and has a unit configuration file in the `/usr/lib/systemd/system/` directory. To change the resource control settings of the persistent units, modify its unit configuration file either manually in a text editor or from the command line.

Manually modifying unit files is one of the ways how to set limits, prioritize, or control access to hardware resources for groups of processes.

Procedure

1. To limit the memory usage of a service, modify the `/usr/lib/systemd/system/example.service` file as follows:

```
...
[Service]
MemoryMax=1500K
...
```

This configuration places a limit on maximum memory consumption of processes executed in a control group, which **example.service** is a part of.

**NOTE**

Use suffixes K, M, G, or T to identify Kilobyte, Megabyte, Gigabyte, or Terabyte as a unit of measurement.

2. Reload all unit configuration files:

```
# systemctl daemon-reload
```

3. Restart the service:

```
# systemctl restart example.service
```

4. Reboot the system.

Verification

1. Check that the changes took effect:

```
# cat /sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes
1536000
```

The memory consumption was limited to approximately 1,500 KB.

**NOTE**

The **memory.limit_in_bytes** file stores the memory limit as a multiple of 4096 bytes - one kernel page size specific for AMD64 and Intel 64. The actual number of bytes depends on a CPU architecture.

Additional resources

- **systemd.resource-control(5), cgroups(7)** manual pages
- [Managing system services with systemctl](#) in RHEL

25.6. REMOVING TRANSIENT CONTROL GROUPS

You can use the **systemd** system and service manager to remove transient control groups (**cgroups**) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Transient **cgroups** are automatically released when all the processes that a service or a scope unit contains finish.

Procedure

- To stop the service unit with all its processes, enter:

```
# systemctl stop <name>.service
```

- To terminate one or more of the unit processes, enter:

```
# systemctl kill <name>.service --kill-who=PID,... --signal=<signal>
```

-

The command uses the **--kill-who** option to select process(es) from the control group you want to terminate. To kill multiple processes at the same time, pass a comma-separated list of PIDs. The **--signal** option determines the type of POSIX signal to be sent to the specified processes. The default signal is *SIGTERM*.

Additional resources

- [What are control groups](#)
- [What are kernel resource controllers](#)
- **systemd.resource-control(5)** and **cgroups(7)** man pages on your system
- [Role of systemd in control groups version 1](#)
- [Managing systemd](#) in RHEL

25.7. REMOVING PERSISTENT CONTROL GROUPS

You can use the **systemd** system and service manager to remove persistent control groups (**cgroups**) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Persistent **cgroups** are released when a service or a scope unit is stopped or disabled and its configuration file is deleted.

Procedure

1. Stop the service unit:

```
# systemctl stop <name>.service
```

2. Disable the service unit:

```
# systemctl disable <name>.service
```

3. Remove the relevant unit configuration file:

```
# rm /usr/lib/systemd/system/<name>.service
```

4. Reload all unit configuration files so that changes take effect:

```
# systemctl daemon-reload
```

Additional resources

- **systemd.resource-control(5)**, **cgroups(7)**, and **systemd.kill(5)** manual pages

25.8. LISTING SYSTEMD UNITS

Use the **systemd** system and service manager to list its units.

Procedure

- List all active units on the system with the **systemctl** utility. The terminal returns an output similar to the following example:

```
# systemctl
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
...
init.scope                         loaded active running  System and Service Manager
session-2.scope                   loaded active running  Session 2 of user jdoe
abrt-ccpp.service                 loaded active exited  Install ABRT coredump hook
abrt-oops.service                 loaded active running  ABRT kernel log watcher
abrt-vmcore.service               loaded active exited  Harvest vmcores for ABRT
abrt-xorg.service                 loaded active running  ABRT Xorg log watcher
...
-.slice                           loaded active active   Root Slice
machine.slice                     loaded active active   Virtual Machine and Container
Slice system-getty.slice          loaded active active
system-getty.slice
system-lvm2\x2dpvscan.slice       loaded active active   system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice       loaded active active   system-
sshd\x2dkeygen.slice
system-systemd\x2dhibernate\x2dresume.slice loaded active active   system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice loaded active active   system-
user\x2druntime\x2ddir.slice
system.slice                      loaded active active   System Slice
user-1000.slice                   loaded active active   User Slice of UID 1000
user-42.slice                    loaded active active   User Slice of UID 42
user.slice                       loaded active active   User and Session Slice
...
```

UNIT

A name of a unit that also reflects the unit position in a control group hierarchy. The units relevant for resource control are a *slice*, a *scope*, and a *service*.

LOAD

Indicates whether the unit configuration file was properly loaded. If the unit file failed to load, the field provides the state *error* instead of *loaded*. Other unit load states are: *stuck*, *merged*, and *masked*.

ACTIVE

The high-level unit activation state, which is a generalization of **SUB**.

SUB

The low-level unit activation state. The range of possible values depends on the unit type.

DESCRIPTION

The description of the unit content and functionality.

- List all active and inactive units:

```
# systemctl --all
```

- Limit the amount of information in the output:

```
# systemctl --type service,masked
```


The **--type** option requires a comma-separated list of unit types such as a *service* and a *slice*, or unit load states such as *loaded* and *masked*.

Additional resources

- [Managing system services with systemctl](#) in RHEL
- The **systemd.resource-control(5)**, **systemd.exec(5)** manual pages

25.9. VIEWING SYSTEMD CGROUPS HIERARCHY

Display control groups (**cgroups**) hierarchy and processes running in specific **cgroups**.

Procedure

- Display the whole **cgroups** hierarchy on your system with the **systemd-cgls** command.

```
# systemd-cgls
Control group /:
-.slice
  └─user.slice
     └─user-42.slice
        └─session-c1.scope
           └─ 965 gdm-session-worker [pam/gdm-launch-environment]
              └─1040 /usr/libexec/gdm-x-session gnome-session --autostart
                 /usr/share/gdm/greeter/autostart
...
  └─init.scope
     └─ 1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
        └─system.slice
...
  └─example.service
     └─ 6882 /bin/bash /home/jdoe/example.sh
        └─ 6902 sleep 1
     └─systemd-journald.service
        └─ 629 /usr/lib/systemd/systemd-journald
...

```

The example output returns the entire **cgroups** hierarchy, where the highest level is formed by *slices*.

- Display the **cgroups** hierarchy filtered by a resource controller with the **systemd-cgls** **<resource_controller>** command.

```
# systemd-cgls memory
Controller memory; Control group /:
└─ 1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
  └─user.slice
     └─user-42.slice
        └─session-c1.scope
           └─ 965 gdm-session-worker [pam/gdm-launch-environment]
...
  └─system.slice
     |
     ...

```

```

├─chronyd.service
│   └─844 /usr/sbin/chronyd
├─example.service
│   ├──8914 /bin/bash /home/jdoe/example.sh
│   └─8916 sleep 1
...

```

The example output lists the services that interact with the selected controller.

- Display detailed information about a certain unit and its part of the **cgroups** hierarchy with the **systemctl status <system_unit>** command.

```

# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset:
disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
     Main PID: 17737 (bash)
        Tasks: 2 (limit: 11522)
       Memory: 496.0K (limit: 1.5M)
      CGroup: /system.slice/example.service
              └─17737 /bin/bash /home/jdoe/example.sh
                └─17743 sleep 1
Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST 2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST 2019

```

Additional resources

- **systemd.resource-control(5)** and **cgroups(7)** man pages on your system

25.10. VIEWING RESOURCE CONTROLLERS

Identify the processes that use resource controllers.

Procedure

1. View which resource controllers a process interacts with, enter the **cat proc/<PID>/cgroup** command.

```

# cat /proc/11269/cgroup
12:freezer:/
11:cpuset:/
10:devices:/system.slice
9:memory:/system.slice/example.service
8:pids:/system.slice/example.service
7:hugetlb:/
6:rdma:/
5:perf_event:/
4:cpu,cpuacct:/
3:net_cls,net_prio:/
2:blkio:/
1:name=systemd:/system.slice/example.service

```

The example output is of the process **PID 11269**, which belongs to the **example.service** unit. You can verify the process was placed in a correct control group as defined by the **systemd** unit file specifications.



NOTE

By default, the items and their ordering in the list of resource controllers is the same for all units started by **systemd**, since it automatically mounts all the default resource controllers.

Additional resources

- The **cgroups(7)** manual page
- Documentation in the `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/` directory

25.11. MONITORING RESOURCE CONSUMPTION

View a list of currently running control groups (**cgroups**) and their resource consumption in real-time.

Procedure

1. Display a dynamic account of currently running **cgroups** with the **systemd-cgtop** command.

```
# systemd-cgtop
Control Group           Tasks %CPU  Memory Input/s Output/s
/                        607  29.8  1.5G   -      -
/system.slice           125   -    428.7M   -      -
/system.slice/ModemManager.service      3   -     8.6M   -      -
/system.slice/NetworkManager.service    3   -    12.8M   -      -
/system.slice/accounts-daemon.service    3   -     1.8M   -      -
/system.slice/boot.mount                  -   -     48.0K   -      -
/system.slice/chronyd.service             1   -     2.0M   -      -
/system.slice/cockpit.socket              -   -     1.3M   -      -
/system.slice/colord.service              3   -     3.5M   -      -
/system.slice/crond.service               1   -     1.8M   -      -
/system.slice/cups.service                1   -     3.1M   -      -
/system.slice/dev-hugepages.mount          -   -    244.0K   -      -
/system.slice/dev-mapper-rhel\x2dswap.swap -   -    912.0K   -      -
/system.slice/dev-mqueue.mount             -   -     48.0K   -      -
/system.slice/example.service             2   -     2.0M   -      -
/system.slice/firewalld.service           2   -    28.8M   -      -
...
```

The example output displays currently running **cgroups** ordered by their resource usage (CPU, memory, disk I/O load). The list refreshes every 1 second by default. Therefore, it offers a dynamic insight into the actual resource usage of each control group.

Additional resources

- The **systemd-cgtop(1)** manual page

CHAPTER 26. CONFIGURING RESOURCE MANAGEMENT BY USING CGROUPS-V2 AND SYSTEMD

The primary function of the **systemd** suite is service management and supervision. As such **systemd** ensures that the appropriate services start at the appropriate time and in the correct order during the boot process. When the services are running, they have to run smoothly to use the underlying hardware platform optimally. Therefore, **systemd** also provides capabilities to define resource management policies and to tune various options, which can improve the performance of the service.

26.1. PREREQUISITES

- Basic knowledge of the [Linux cgroup](#) subsystem.

26.2. INTRODUCTION TO RESOURCE DISTRIBUTION MODELS

For resource management, **systemd** uses the control groups version 2 (**cgroups-v2**) interface.



NOTE

By default, RHEL 8 uses **cgroups-v1**. Therefore, you must enable **cgroups-v2** so that **systemd** can use the cgroups-v2 interface for resource management. For more information about how to enable **cgroups-v2**, see [Mounting cgroups-v2](#).

To modify the distribution of system resources, you can apply one or more of the following resource distribution models:

Weights

A resource is distributed by adding up the weights of all sub-groups and giving each sub-group a fraction of the resource based on its weight as compared to the overall sum of all weights.

For example, if you have 10 **cgroups**, each with Weight of value 100, the sum is 1000 and each **cgroup** receives one tenth of the resource.

Weight is usually used to distribute stateless resources. To adjust the weight of a control group, use the *CPUWeight*= option.

Limits

A **cgroup** can consume up to the configured amount of a resource, but you can also overcommit resources. Therefore, the sum of sub-group limits can exceed the limit of the parent **cgroup**.

To adjust the limits of a control group, use the *MemoryMax*= option.

Protections

You can set up a protected amount of a resource for a **cgroup**. If the resource usage is below the protection boundary, the kernel will try not to penalize this **cgroup** in favor of other **cgroups** that compete for the same resource. An overcommit is also possible.

To adjust the protected resource amounts for a control group, use the *MemoryLow*= option.

Allocations

Exclusive allocations of a specific amount of a finite resource, such as the real-time budget. An overcommit is possible.

Additional resources

- [Allocating memory resources using systemd](#)
- [Configuring I/O bandwidth using systemd](#)

26.3. ALLOCATING CPU RESOURCES BY USING SYSTEMD

On a system managed by **systemd**, each system service is started in its **cgroup**. By enabling the support for the CPU **cgroup** controller, the system uses the service-aware distribution of CPU resources instead of the per-process distribution. In the service-aware distribution, each service receives about the same amount of CPU time relative to all other services running on the system, regardless of the number of processes use the same services.

If a specific service requires more CPU resources, you can grant them by changing the CPU time allocation policy for the service.

Procedure

To set a CPU time allocation policy option when using **systemd**:

1. Check the assigned values of the CPU time allocation policy option in the service of your choice:

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

2. Set the required value of the CPU time allocation policy option as a root:

```
# systemctl set-property <service name> <CPU time allocation policy option>=<value>
```

The cgroup properties are applied immediately after they are set. Therefore, the service does not require a restart.

Verification

- To verify whether you successfully changed the required value of the CPU time allocation policy option for your service, enter:

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

Additional resources

- [CPU time allocation policy options for systemd](#)
- [Introduction to resource distribution models](#)

26.4. CPU TIME ALLOCATION POLICY OPTIONS FOR SYSTEMD

The most frequently used CPU time allocation policy options include the following:

CPUWeight=

Assigns **higher priority** to a particular service over all other services. You can select a value from the interval 1 - 10,000. The default value is 100.

For example, to give **httpd.service** twice as much CPU as to all other services, set the value to **CPUWeight=200**.

Note that **CPUWeight=** is applied only in cases when available CPU resources are depleted.

CPUQuota=

Assigns the **absolute CPU time quota** to a service. The value of this option specifies the maximum percentage of CPU time that a service will receive relative to the total CPU time available, for example **CPUQuota=30%**.

Note that **CPUQuota=** represents the limit value for particular resource distribution models, such as Weights and Limits.

For more information about **CPUQuota=**, see the **systemd.resource-control(5)** man page on your system.

26.5. ALLOCATING MEMORY RESOURCES BY USING SYSTEMD

To allocate memory resources by using **systemd**, use any of the memory configuration options:

- **MemoryMin**
- **MemoryLow**
- **MemoryHigh**
- **MemoryMax**
- **MemorySwapMax**

Procedure

To set a memory allocation configuration option when using **systemd**:

1. Check the assigned values of the memory allocation configuration option in the service of your choice.

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

2. Set the required value of the memory allocation configuration option as a **root**.

```
# systemctl set-property <service name> <memory allocation configuration option>=<value>
```



NOTE

The cgroup properties are applied immediately after they are set. Therefore, the service does not require a restart.

Verification

- To verify whether you have successfully changed the required value of the memory allocation configuration option for your service, enter:

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

26.6. MEMORY ALLOCATION CONFIGURATION OPTIONS FOR SYSTEMD

You can use the following options when using **systemd** to configure system memory allocation:

MemoryMin

Hard memory protection. If the memory usage is below the specified limit, the cgroup memory will not be reclaimed.

MemoryLow

Soft memory protection. If the memory usage is below the specified limit, the cgroup memory can be reclaimed only if no memory is reclaimed from unprotected cgroups.

MemoryHigh

Memory throttle limit. If the memory usage goes above the specified limit, the processes in the cgroup are throttled and put under a heavy reclaim pressure.

MemoryMax

Absolute limit for the memory usage. You can use the kilo (K), mega (M), giga (G), tera (T) suffixes, for example **MemoryMax=1G**.

MemorySwapMax

Hard limit on the swap usage.



NOTE

When you exhaust your memory limit, the Out-of-memory (OOM) killer will stop the running service. To prevent this, lower the **OOMScoreAdjust=** value to increase the memory tolerance.

26.7. CONFIGURING I/O BANDWIDTH BY USING SYSTEMD

To improve the performance of a specific service in RHEL 8, you can assign I/O bandwidth resources to that service by using **systemd**.

To do so, you can use the following I/O configuration options:

- IOWeight
- IODeviceWeight
- IOReadBandwidthMax
- IOWriteBandwidthMax
- IOReadIOPSMax
- IOWriteIOPSMax

Procedure

To set a **I/O bandwidth configuration** option by using **systemd**:

1. Check the assigned values of the I/O bandwidth configuration option in the service of your choice:

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

- Set the required value of the I/O bandwidth configuration option as a **root**:

```
# systemctl set-property <service name> <I/O bandwidth configuration option>=  
<value>
```

When the **cgroup** properties are set, they are applied immediately. Therefore, restarting the service is not required.

Verification

- To verify whether you successfully changed the required value of the I/O bandwidth configuration option for your service, enter:

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

26.8. I/O BANDWIDTH CONFIGURATION OPTIONS FOR SYSTEMD

To manage the block layer I/O policies by using **systemd**, the following configuration options are available:

IOWeight

Sets the default I/O weight. The weight value is used as a basis for the calculation of how much of the real I/O bandwidth the service receives in relation to the other services.

IODeviceWeight

Sets the I/O weight for a specific block device.



NOTE

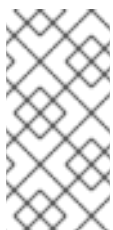
Weight-based options are supported only if the block device is using the CFQ I/O scheduler. No option is supported if the device uses the Multi-Queue Block I/O queuing mechanism.

For example, **IODeviceWeight=/dev/disk/by-id/dm-name-rhel-root 200**.

IOReadBandwidthMax, IOWriteBandwidthMax

Sets the absolute bandwidth per device or a mount point.

For example, **IOWriteBandwidth=/var/log 5M**.



NOTE

systemd handles the file-system-to-device translation automatically.

IOReadIOPSMax, IOWriteIOPSMax

Sets the absolute bandwidth in Input/Output Operations Per Second (IOPS).

26.9. CONFIGURING CPUSET CONTROLLER BY USING SYSTEMD

With the **systemd** resource management API, you can configure limits on a set of CPUs and NUMA nodes that a service can use. This limit restricts access to system resources used by the processes. The requested configuration is written in the **cpuset.cpus** and **cpuset.mems** files.

However, the requested configuration might not be used, as the parent **cgroup** limits either **cpus** or **mems**. To access the current configuration, the **cpuset.cpus.effective** and **cpuset.mems.effective** files are exported to the users.

Procedure

- To set **AllowedCPUs**:

```
# systemctl set-property <service name>.service AllowedCPUs=<value>
```

For example:

```
# systemctl set-property <service name>.service AllowedCPUs=0-5
```

- To set **AllowedMemoryNodes**:

```
# systemctl set-property <service name>.service AllowedMemoryNodes=<value>
```

For example:

```
# systemctl set-property <service name>.service AllowedMemoryNodes=0
```

CHAPTER 27. CONFIGURING CPU AFFINITY AND NUMA POLICIES USING SYSTEMD

The CPU management, memory management, and I/O bandwidth options deal with partitioning available resources.

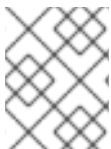
27.1. CONFIGURING CPU AFFINITY USING SYSTEMD

CPU affinity settings help you restrict the access of a particular process to some CPUs. Effectively, the CPU scheduler never schedules the process to run on the CPU that is not in the affinity mask of the process.

The default CPU affinity mask applies to all services managed by systemd.

To configure CPU affinity mask for a particular systemd service, systemd provides **CPUAffinity=** both as a unit file option and a manager configuration option in the **/etc/systemd/system.conf** file.

The **CPUAffinity= unit file option** sets a list of CPUs or CPU ranges that are merged and used as the affinity mask. The **CPUAffinity option** in the **/etc/systemd/system.conf** file defines an affinity mask for the process identification number (PID) 1 and all processes forked off of PID1. You can then override the **CPUAffinity** on a per-service basis.



NOTE

After configuring CPU affinity mask for a particular systemd service, you must restart the system to apply the changes.

Procedure

To set CPU affinity mask for a particular systemd service using the **CPUAffinity unit file** option:

1. Check the values of the **CPUAffinity** unit file option in the service of your choice:

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. As a root, set the required value of the **CPUAffinity** unit file option for the CPU ranges used as the affinity mask:

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. Restart the service to apply the changes.

```
# systemctl restart <service name>
```

To set CPU affinity mask for a particular systemd service using the **manager configuration** option:

1. Edit the **/etc/systemd/system.conf** file:

```
# vi /etc/systemd/system.conf
```

2. Search for the **CPUAffinity=** option and set the CPU numbers
3. Save the edited file and restart the server to apply the changes.

27.2. CONFIGURING NUMA POLICIES USING SYSTEMD

Non-uniform memory access (NUMA) is a computer memory subsystem design, in which the memory access time depends on the physical memory location relative to the processor.

Memory close to the CPU has lower latency (local memory) than memory that is local for a different CPU (foreign memory) or is shared between a set of CPUs.

In terms of the Linux kernel, NUMA policy governs where (for example, on which NUMA nodes) the kernel allocates physical memory pages for the process.

systemd provides unit file options **NUMAPolicy** and **NUMAMask** to control memory allocation policies for services.

Procedure

To set the NUMA memory policy through the **NUMAPolicy** unit file option:

1. Check the values of the **NUMAPolicy** unit file option in the service of your choice:

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. As a root, set the required policy type of the **NUMAPolicy** unit file option:

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. Restart the service to apply the changes.

```
# systemctl restart <service name>
```

To set a global **NUMAPolicy** setting using the [Manager] configuration option:

1. Search in the **/etc/systemd/system.conf** file for the **NUMAPolicy** option in the [Manager] section of the file.
2. Edit the policy type and save the file.
3. Reload the **systemd** configuration:

```
# systemd daemon-reload
```

4. Reboot the server.



IMPORTANT

When you configure a strict NUMA policy, for example **bind**, make sure that you also appropriately set the **CPUAffinity=** unit file option.

Additional resources

- [NUMA policy configuration options for systemd](#)
- The **systemd.resource-control(5)**, **systemd.exec(5)**, and **set_mempolicy(2)** man pages.

27.3. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD

Systemd provides the following options to configure the NUMA policy:

NUMAPolicy

Controls the NUMA memory policy of the executed processes. You can use these policy types:

- default
- preferred
- bind
- interleave
- local

NUMAMask

Controls the NUMA node list that is associated with the selected NUMA policy.

Note that you do not have to specify the **NUMAMask** option for the following policies:

- default
- local

For the preferred policy, the list specifies only a single NUMA node.

Additional resources

- **systemd.resource-control(5)**, **systemd.exec(5)**, and **set_mempolicy(2)** man pages on your system
- [NUMA policy configuration options for systemd](#)

CHAPTER 28. ANALYZING SYSTEM PERFORMANCE WITH BPF COMPILER COLLECTION

The BPF Compiler Collection (BCC) analyzes system performance by combining the capabilities of Berkeley Packet Filter (BPF). With BPF, you can safely run the custom programs within the kernel to access system events and data for performance monitoring, tracing, and debugging. BCC simplifies the development and deployment of BPF programs with tools and libraries for users to extract important insights from their systems.

28.1. INSTALLING THE BCC-TOOLS PACKAGE

Install the **bcc-tools** package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Procedure

- Install **bcc-tools**.

```
# yum install bcc-tools
```

The BCC tools are installed in the **/usr/share/bcc/tools/** directory.

Verification

- Inspect the installed tools:

```
# ls -l /usr/share/bcc/tools/
...
-rwxr-xr-x. 1 root root 4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root 3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root 7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root 8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root 7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root 6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

The **doc** directory in the listing provides documentation for each tool.

28.2. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES

Use certain pre-created programs from the BPF Compiler Collection (BCC) library to efficiently and securely analyze the system performance on the per-event basis. The set of pre-created programs in the BCC library can serve as examples for creation of additional programs.

Prerequisites

- [Installed bcc-tools package](#)
- Root permissions

Procedure

Using **execsnoop** to examine the system processes

1. Run the **execsnoop** program in one terminal:

```
# /usr/share/bcc/tools/execsnoop
```

1. To create a short-lived process of the **ls** command, in another terminal, enter:

```
$ ls /usr/share/bcc/tools/doc/
```

2. The terminal running **execsnoop** shows the output similar to the following:

```
PCOMM PID  PPID  RET ARGS
ls  8382  8287   0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

The **execsnoop** program prints a line of output for each new process that consume system resources. It even detects processes of programs that run very shortly, such as **ls**, and most monitoring tools would not register them.

The **execsnoop** output displays the following fields:

PCOMM

The parent process name. (**ls**)

PID

The process ID. (**8382**)

PPID

The parent process ID. (**8287**)

RET

The return value of the **exec()** system call (**0**), which loads program code into new processes.

ARGS

The location of the started program with arguments.

To see more details, examples, and options for **execsnoop**, see **/usr/share/bcc/tools/doc/execsnoop_example.txt** file.

For more information about **exec()**, see **exec(3)** manual pages.

Using **opensnoop** to track what files a command opens

1. In one terminal, run the **opensnoop** program to print the output for files opened only by the process of the **uname** command:

```
# /usr/share/bcc/tools/opensnoop -n uname
```

1. In another terminal, enter the command to open certain files:

```
$ uname
```

2. The terminal running **opensnoop** shows the output similar to the following:

```
PID  COMM  FD ERR PATH
8596  uname  3  0  /etc/ld.so.cache
8596  uname  3  0  /lib64/libc.so.6
8596  uname  3  0  /usr/lib/locale/locale-archive
...
```

The **opensnoop** program watches the **open()** system call across the whole system, and prints a line of output for each file that **uname** tried to open along the way.

The **opensnoop** output displays the following fields:

PID

The process ID. (**8596**)

COMM

The process name. (**uname**)

FD

The file descriptor – a value that **open()** returns to refer to the open file. (**3**)

ERR

Any errors.

PATH

The location of files that **open()** tried to open.

If a command tries to read a non-existent file, then the **FD** column returns **-1** and the **ERR** column prints a value corresponding to the relevant error. As a result, **opensnoop** can help you identify an application that does not behave properly.

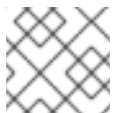
To see more details, examples, and options for **opensnoop**, see **/usr/share/bcc/tools/doc/opensnoop_example.txt** file.

For more information about **open()**, see **open(2)** manual pages.

Use the **biotop** to monitor the top processes performing I/O operations on the disk

1. Run the **biotop** program in one terminal with argument **30** to produce 30 second summary:

```
# /usr/share/bcc/tools/biotop 30
```



NOTE

When no argument provided, the output screen by default refreshes every 1 second.

1. In another terminal, enter command to read the content from the local hard disk device and write the output to the **/dev/zero** file:

```
# dd if=/dev/vda of=/dev/zero
```

This step generates certain I/O traffic to illustrate **biotop**.

2. The terminal running **biotop** shows the output similar to the following:

```

PID  COMM      D MAJ MIN DISK   I/O Kbytes  AVGms
9568 dd        R 252 0 vda    16294 14440636.0 3.69
48  kswapd0    W 252 0 vda     1763 120696.0 1.65
7571 gnome-shell R 252 0 vda     834 83612.0 0.33
1891 gnome-shell R 252 0 vda    1379 19792.0 0.15
7515 Xorg       R 252 0 vda     280 9940.0 0.28
7579 llvmpipe-1 R 252 0 vda     228 6928.0 0.19
9515 gnome-control-c R 252 0 vda     62 6444.0 0.43
8112 gnome-terminal- R 252 0 vda     67 2572.0 1.54
7807 gnome-software R 252 0 vda     31 2336.0 0.73
9578 awk       R 252 0 vda     17 2228.0 0.66
7578 llvmpipe-0 R 252 0 vda     156 2204.0 0.07
9581 pgrep      R 252 0 vda     58 1748.0 0.42
7531 InputThread R 252 0 vda     30 1200.0 0.48
7504 gdbus     R 252 0 vda     3 1164.0 0.30
1983 llvmpipe-1 R 252 0 vda     39 724.0 0.08
1982 llvmpipe-0 R 252 0 vda     36 652.0 0.06
...

```

The **biotop** output displays the following fields:

PID

The process ID. (**9568**)

COMM

The process name. (**dd**)

DISK

The disk performing the read operations. (**vda**)

I/O

The number of read operations performed. (16294)

Kbytes

The amount of Kbytes reached by the read operations. (14,440,636)

AVGms

The average I/O time of read operations. (3.69)

For more details, examples, and options for **biotop**, see the **/usr/share/bcc/tools/doc/biotop_example.txt** file.

For more information about **dd**, see **dd(1)** manual pages.

Using **xfsslower** to expose unexpectedly slow file system operations

The **xfsslower** measures the time spent by XFS file system in performing read, write, open or sync (**fsync**) operations. The **1** argument ensures that the program shows only the operations that are slower than 1 ms.

1. Run the **xfsslower** program in one terminal:

```
# /usr/share/bcc/tools/xfsslower 1
```




NOTE

When no arguments provided, **xfsslower** by default displays operations slower than 10 ms.

- In another terminal, enter the command to create a text file in the **vim** editor to start interaction with the XFS file system:

```
$ vim text
```

- The terminal running **xfsslower** shows something similar upon saving the file from the previous step:

```
TIME    COMM      PID  T BYTES  OFF_KB  LAT(ms)  FILENAME
13:07:14 b'bash'    4754  R 256    0        7.11 b'vim'
13:07:14 b'vim'    4754  R 832    0        4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'    4754  R 32     20       1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'    4754  R 1982   0        2.30 b'vimrc'
13:07:14 b'vim'    4754  R 1393   0        2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'    4754  S 0      0        6.71 b'text'
13:07:45 b'pool'   2588  R 16     0        5.58 b'text'
...
```

Each line represents an operation in the file system, which took more time than a certain threshold. **xfsslower** detects possible file system problems, which can take form of unexpectedly slow operations.

The **xfsslower** output displays the following fields:

COMM

The process name. (**b'bash'**)

T

The operation type. (**R**)

- Read
- Write
- Sync

OFF_KB

The file offset in KB. (0)

FILENAME

The file that is read, written, or synced.

To see more details, examples, and options for **xfsslower**, see **/usr/share/bcc/tools/doc/xfsslower_example.txt** file.

For more information about **fsync**, see **fsync(2)** manual pages.

CHAPTER 29. ENHANCING SECURITY WITH THE KERNEL INTEGRITY SUBSYSTEM

You can improve the protection of your system by using components of the kernel integrity subsystem. Learn more about the relevant components and their configuration.

29.1. THE KERNEL INTEGRITY SUBSYSTEM

The integrity subsystem is the kernel component that maintains the overall integrity of system data. This subsystem helps in maintaining the system in the same state from the time it was built. Using this subsystem, you can protect executable files, libraries, and configuration files.

The kernel integrity subsystem consists of two major components:

Integrity Measurement Architecture (IMA)

- IMA measures file content whenever it is executed or accessed by cryptographically hashing or signing with cryptographic keys. The keys are stored in the kernel keyring subsystem.
- IMA places the measured values within the kernel's memory space. This prevents users of the system from modifying the measured values.
- IMA allows local and remote parties to verify the measured values.
- IMA provides local validation of the current content of files against the values previously stored in the measurement list within the kernel memory. This extension forbids performing any operation on a specific file in case the current and the previous measures do not match.

Extended Verification Module (EVM)

- EVM protects extended attributes of files (also known as *xattr*) related to system security, such as IMA measurements and SELinux attributes. EVM cryptographically hashes their corresponding values or signs them with cryptographic keys. The keys are stored in the kernel keyring subsystem.

The kernel integrity subsystem can use the Trusted Platform Module (TPM) to further harden system security.

A TPM is a hardware, firmware, or virtual component with integrated cryptographic keys that are built according to the TPM specification by the Trusted Computing Group (TCG) for important cryptographic functions. By providing cryptographic functions from a protected and tamper-proof area of the hardware chip, TPMs are protected from software-based attacks. TPMs provide the following features:

- Random-number generator
- Generator and secure storage for cryptographic keys
- Hashing generator
- Remote attestation

Additional resources

- [Security hardening](#)
- [Basic and advanced configuration of Security-Enhanced Linux \(SELinux\)](#)

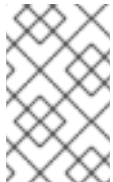
29.2. TRUSTED AND ENCRYPTED KEYS

Trusted keys and *encrypted keys* are an important part of enhancing system security.

Trusted and encrypted keys are variable-length symmetric keys generated by the kernel that use the kernel keyring service. You can verify the integrity of the keys, for example, to allow the extended verification module (EVM) to verify and confirm the integrity of a running system. User-level programs can only access the keys in the form of encrypted *blobs*.

Trusted keys

Trusted keys need the Trusted Platform Module (TPM) chip, which is used to both create and encrypt (seal) the keys. Each TPM has a master wrapping key, called the storage root key, which is stored within the TPM itself.



NOTE

Red Hat Enterprise Linux 8 supports both TPM 1.2 and TPM 2.0. For more information, see the Red Hat Knowledgebase solution [Is Trusted Platform Module \(TPM\) supported by Red Hat?](#).

You can verify the status of TPM 2.0 chip:

```
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

You can also enable a TPM 2.0 chip and manage the TPM 2.0 device through settings in the machine firmware.

In addition to that, you can seal the trusted keys with a specific set of the TPM's *platform configuration register* (PCR) values. PCR contains a set of integrity-management values that reflect the firmware, boot loader, and operating system. PCR-sealed keys can only be decrypted by the TPM on the system where they were encrypted. However, when you load a PCR-sealed trusted key to a keyring, its associated PCR values are verified. After verification, you can update the key with new or future PCR values, for example, to support booting a new kernel. Also, you can save a single key as multiple blobs, each with a different PCR value.

Encrypted keys

Encrypted keys do not require a TPM, because they use the kernel Advanced Encryption Standard (AES), which makes them faster than trusted keys. Encrypted keys are created using kernel-generated random numbers and encrypted by a *master key* when they are exported into user-space blobs.

The master key is either a trusted key or a user key. If the master key is not trusted, the security of the encrypted key depends on the user key that was used to encrypt it.

29.3. WORKING WITH TRUSTED KEYS

You can improve system security by using the **keyctl** utility to create, export, load and update trusted keys.

Prerequisites

- For the 64-bit ARM architecture and IBM Z, the **trusted** kernel module is loaded.

```
# modprobe trusted
```

For more information about how to load kernel modules, see [Loading kernel modules at system runtime](#).

- Trusted Platform Module (TPM) is enabled and active. See [The kernel integrity subsystem](#) and [Trusted and encrypted keys](#).



NOTE

Red Hat Enterprise Linux 8 supports both TPM 1.2 and TPM 2.0. If you use TPM 1.2, skip step 1.

Procedure

- Create a 2048-bit RSA key with an SHA-256 primary storage key with a persistent handle of, for example, *81000001*, by using one of the following utilities:

- By using the **tss2** package:

```
# TPM_DEVICE=/dev/tpm0 tsscreateprimary -hi o -st
Handle 80000000
# TPM_DEVICE=/dev/tpm0 tssevictcontrol -hi o -ho 80000000 -hp 81000001
```

- By using the **tpm2-tools** package:

```
# tpm2_createprimary --key-algorithm=rsa2048 --key-context=key.ctx
name-alg:
  value: sha256
  raw: 0xb
...
sym-keybits: 128
rsa: xxxxxx...

# tpm2_evictcontrol -c key.ctx 0x81000001
persistentHandle: 0x81000001
action: persisted
```

- Create a trusted key:

- By using a TPM 2.0 with the syntax of **keyctl add trusted <NAME> "new <KEY_LENGTH> keyhandle=<PERSISTENT-HANDLE> [options]" <KEYRING>**. In this example, the persistent handle is *81000001*.

```
# keyctl add trusted kmk "new 32 keyhandle=0x81000001" @u
642500861
```

The command creates a trusted key called **kmk** with the length of **32** bytes (256 bits) and places it in the user keyring (**@u**). The keys may have a length of 32 to 128 bytes (256 to 1024 bits).

- b. By using a TPM 1.2 with the syntax of **keyctl add trusted <NAME> "new <KEY_LENGTH>" <KEYRING>**:

```
# keyctl add trusted kmk "new 32" @u
```

3. List the current structure of the kernel keyrings:

```
# keyctl show
Session Keyring
  -3 --alswrv 500 500 keyring: ses 97833714 --alswrv 500 -1 | keyring: uid.1000
642500861 --alswrv 500 500 | trusted: kmk
```

4. Export the key to a user-space blob by using the serial number of the trusted key:

```
# keyctl pipe 642500861 > kmk.blob
```

The command uses the **pipe** subcommand and the serial number of **kmk**.

5. Load the trusted key from the user-space blob:

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

6. Create secure encrypted keys that use the TPM-sealed trusted key (**kmk**). Follow this syntax:
keyctl add encrypted <NAME> "new [FORMAT] <KEY_TYPE>:<PRIMARY_KEY_NAME> <KEY_LENGTH>" <KEYRING>:

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

Additional resources

- the **keyctl(1)** manual page

29.4. WORKING WITH ENCRYPTED KEYS

You can improve system security on systems where a Trusted Platform Module (TPM) is not available by managing encrypted keys.

Encrypted keys, unless sealed by a trusted primary key, inherit the security level of the user primary key (random-number key) used for encryption. Therefore, it is highly recommended to load the primary user key securely, ideally early in the boot process.

Prerequisites

- For the 64-bit ARM architecture and IBM Z, the **encrypted-keys** kernel module is loaded:

```
# modprobe encrypted-keys
```

For more information about how to load kernel modules, see [Loading kernel modules at system runtime](#).

Procedure

1. Generate a user key by using a random sequence of numbers:

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

The command generates a user key called **kmk-user** which acts as a *primary key* and is used to seal the actual encrypted keys.

2. Generate an encrypted key using the primary key from the previous step:

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

Verification

1. List all keys in the specified user keyring:

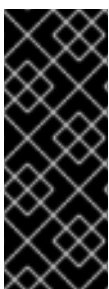
```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv 1000 1000 user: kmk-user
1012412758: --alswrv 1000 1000 encrypted: encr-key
```

Additional resources

- The **keyctl(1)** manual page

29.5. ENABLING IMA AND EVM

You can enable and configure Integrity measurement architecture (IMA) and extended verification module (EVM) to improve the security of the operating system.



IMPORTANT

Always enable EVM together with IMA.

Although you can enable EVM alone, EVM appraisal is only triggered by an IMA appraisal rule. Therefore, EVM does not protect file metadata such as SELinux attributes. If file metadata is tampered with offline, EVM can only prevent file metadata changes. It does not prevent file access, such as executing the file.

Prerequisites

- Secure Boot is temporarily disabled.



NOTE

When Secure Boot is enabled, the **ima_appraise=fix** kernel command-line parameter does not work.

- The **securityfs** file system is mounted on the **/sys/kernel/security/** directory and the **/sys/kernel/security/integrity/ima/** directory exists. You can verify where **securityfs** is mounted by using the **mount** command:

```
# mount
...
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
...
```

- The **systemd** service manager is patched to support IMA and EVM on boot time. Verify by using the following command:

```
# grep <options> pattern <files>
```

For example:

```
# dmesg | grep -i -e EVM -e IMA -w
[ 0.598533] ima: No TPM chip found, activating TPM-bypass!
[ 0.599435] ima: Allocated hash algorithm: sha256
[ 0.600266] ima: No architecture policies found
[ 0.600813] evm: Initialising EVM extended attributes:
[ 0.601581] evm: security.selinux
[ 0.601963] evm: security.ima
[ 0.602353] evm: security.capability
[ 0.602713] evm: HMAC attrs: 0x1
[ 1.455657] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
[ 2.532639] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
```

Procedure

1. Enable IMA and EVM in the *fix* mode for the current boot entry and allow users to gather and update the IMA measurements by adding the following kernel command-line parameters:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="ima_policy=appraise_tcb ima_appraise=fix evm=fix"
```

The command enables IMA and EVM in the *fix* mode for the current boot entry to gather and update the IMA measurements.

The **ima_policy=appraise_tcb** kernel command-line parameter ensures that the kernel uses the default Trusted Computing Base (TCB) measurement policy and the appraisal step. The appraisal step forbids access to files whose prior and current measures do not match.

2. Reboot to make the changes come into effect.
3. Optional: Verify the parameters added to the kernel command line:

cat /proc/cmdline

```
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

4. Create a kernel master key to protect the EVM key:

```
# keyctl add user kmk "$(dd if=/dev/urandom bs=1 count=32 2> /dev/null)" @u
748544121
```

The **kmk** is kept entirely in the kernel space memory. The 32-byte long value of the **kmk** is generated from random bytes from the **/dev/urandom** file and placed in the user (**@u**) keyring. The key serial number is on the first line of the previous output.

5. Create an encrypted EVM key based on the **kmk**:

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

The command uses the **kmk** to generate and encrypt a 64-byte long user key (named **evm-key**) and places it in the user (**@u**) keyring. The key serial number is on the first line of the previous output.

**IMPORTANT**

It is necessary to name the user key as **evm-key** because that is the name the EVM subsystem is expecting and is working with.

6. Create a directory for exported keys.

```
# mkdir -p /etc/keys/
```

7. Search for the **kmk** and export its unencrypted value into the new directory.

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

8. Search for the **evm-key** and export its encrypted value into the new directory.

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

The **evm-key** has been encrypted by the kernel master key earlier.

9. Optional: View the newly created keys:

```
# keyctl show
Session Keyring
974575405 --alswrv 0 0 keyring: ses 299489774 --alswrv 0 65534 | keyring:
uid.0 748544121 --alswrv 0 0 | user: kmk
641780271 --alswrv 0 0 \_ encrypted: evm-key

# ls -l /etc/keys/
```



```
total 8
-rw-r--r--. 1 root root 246 Jun 24 12:44 evm-key
-rw-r--r--. 1 root root 32 Jun 24 12:43 kmk
```

10. Optional: If the keys are removed from the keyring, for example after system reboot, you can import the already exported **kmk** and **evm-key** instead of creating new ones.

- a. Import the **kmk**.

```
# keyctl add user kmk "$(cat /etc/keys/kmk)" @u
451342217
```

- b. Import the **evm-key**.

```
# keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
924537557
```

11. Activate EVM.

```
# echo 1 > /sys/kernel/security/evm
```

12. Relabel the whole system.

```
# find / -fstype xfs -type f -uid 0 -exec head -n 1 '{}' >/dev/null \;
```



WARNING

Enabling IMA and EVM without relabeling the system might make the majority of the files on the system inaccessible.

Verification

- Verify that EVM has been initialized:

```
# dmesg | tail -1
[...] evm: key initialized
```

29.6. COLLECTING FILE HASHES WITH INTEGRITY MEASUREMENT ARCHITECTURE

In the *measurement* phase, you can create file hashes and store them as extended attributes (*xattrs*) of those files. With the file hashes, you can generate either an RSA-based digital signature or a Hash-based Message Authentication Code (HMAC-SHA1) and prevent offline tampering attacks on the extended attributes.

Prerequisites

- IMA and EVM are enabled. For more information, see [Enabling integrity measurement architecture and extended verification module](#).
- A valid trusted key or encrypted key is stored in the kernel keyring.
- The **ima-evm-utils**, **attr**, and **keyutils** packages are installed.

Procedure

1. Create a test file:

```
# echo <Test_text> > test_file
```

IMA and EVM ensure that the **test_file** example file has assigned hash values that are stored as its extended attributes.

2. Inspect the file's extended attributes:

```
# getfattr -m . -d test_file
# file: test_file
security.evm=0sAnDly4VPA0HArpPO/EqiutnNyBql
security.ima=0sAQOEDeuUnWzwwKYk+n66h/vby3eD
```

The example output shows extended attributes with the IMA and EVM hash values and SELinux context. EVM adds a **security.evm** extended attribute related to the other attributes. At this point, you can use the **evmctl** utility on **security.evm** to generate either an RSA-based digital signature or a Hash-based Message Authentication Code (HMAC-SHA1).

Additional resources

- [Security hardening](#)

CHAPTER 30. CONFIGURING KERNEL PARAMETERS PERMANENTLY BY USING RHEL SYSTEM ROLES

You can use the **kernel_settings** RHEL system role to configure kernel parameters on multiple clients simultaneously. Simultaneous configuration has the following advantages:

- Provides a friendly interface with efficient input setting.
- Keeps all intended kernel parameters in one place.

After you run the **kernel_settings** role from the control machine, the kernel parameters are applied to the managed systems immediately and persist across reboots.



IMPORTANT

Note that RHEL system role delivered over RHEL channels are available to RHEL customers as an RPM package in the default AppStream repository. RHEL system role are also available as a collection to customers with Ansible subscriptions over Ansible Automation Hub.

30.1. APPLYING SELECTED KERNEL PARAMETERS BY USING THE **KERNEL_SETTINGS** RHEL SYSTEM ROLE

You can use the **kernel_settings** RHEL system role to remotely configure various kernel parameters across multiple managed operating systems with persistent effects. For example, you can configure:

- Transparent hugepages to increase performance by reducing the overhead of managing smaller pages.
- The largest packet sizes to be transmitted over the network with the loopback interface.
- Limits on files to be opened simultaneously.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configuring kernel settings
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure hugepages, packet size for loopback device, and limits on
      simultaneously open files.
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.kernel_settings
      vars:
```

```

kernel_settings_sysctl:
  - name: fs.file-max
    value: 400000
  - name: kernel.threads-max
    value: 65536
kernel_settings_sysfs:
  - name: /sys/class/net/lo/mtu
    value: 65000
kernel_settings_transparent_hugepages: advise
kernel_settings_reboot_ok: true

```

The settings specified in the example playbook include the following:

kernel_settings_sysfs: <list_of_sysctl_settings>

A YAML list of **sysctl** settings and the values you want to assign to these settings.

kernel_settings_transparent_hugepages: <value>

Controls the memory subsystem Transparent Huge Pages (THP) setting. You can disable THP support (**never**), enable it system wide (**always**) or inside **MAD_HUGEPAGE** regions (**advise**).

kernel_settings_reboot_ok: <true/false>

The default is **false**. If set to **true**, the system role will determine if a reboot of the managed host is necessary for the requested changes to take effect and reboot it. If set to **false**, the role will return the variable **kernel_settings_reboot_required** with a value of **true**, indicating that a reboot is required. In this case, a user must reboot the managed node manually.

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.kdump/README.md` file on the control node.

1. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

2. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Verify the affected kernel parameters:

```

# ansible managed-node-01.example.com -m command -a 'sysctl fs.file-max
kernel.threads-max net.ipv6.conf.lo.mtu'
# ansible managed-node-01.example.com -m command -a 'cat
/sys/kernel/mm/transparent_hugepage/enabled'

```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md` file
- `/usr/share/doc/rhel-system-roles/kernel_settings/` directory

CHAPTER 31. CONFIGURING THE GRUB BOOT LOADER BY USING RHEL SYSTEM ROLES

By using the **bootloader** RHEL system role, you can automate the configuration and management tasks related to the GRUB boot loader.

This role currently supports configuring the GRUB boot loader, which runs on the following CPU architectures:

- AMD and Intel 64-bit architectures (x86-64)
- The 64-bit ARM architecture (ARMv8.0)
- IBM Power Systems, Little Endian (POWER9)

31.1. UPDATING THE EXISTING BOOT LOADER ENTRIES BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to update the existing entries in the GRUB boot menu in an automated fashion. This way you can efficiently pass specific kernel command-line parameters that can optimize the performance or behavior of your systems.

For example, if you leverage systems, where detailed boot messages from the kernel and init system are not necessary, use **bootloader** to apply the **quiet** parameter to your existing boot loader entries on your managed nodes to achieve a cleaner, less cluttered, and more user-friendly booting experience.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- You identified the kernel that corresponds to the boot loader entry you want to update.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Update existing boot loader entries
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_settings:
          - kernel:
              path: /boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64
              options:
```

```
- name: quiet
  state: present
  bootloader_reboot_ok: true
```

The settings specified in the example playbook include the following:

kernel

Specifies the kernel connected with the boot loader entry that you want to update.

options

Specifies the kernel command-line parameters to update for your chosen boot loader entry (kernel).

bootloader_reboot_ok: true

The role detects that a reboot is required for the changes to take effect and performs a restart of the managed node.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Check that your specified boot loader entry has updated kernel command-line parameters:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'grubby --
info=ALL'
managed-node-01.example.com | CHANGED | rc=0 >>
...
index=1
kernel="/boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64"
args="ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd"
title="Red Hat Enterprise Linux (5.14.0-362.24.1.el9_3.aarch64) 9.4 (Plow)"
id="2c9ec787230141a9b087f774955795ab-5.14.0-362.24.1.el9_3.aarch64"
...
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory

- [Working With Playbooks](#)
- [Using Variables](#)
- [Roles](#)
- [Configuring kernel command-line parameters](#)

31.2. SECURING THE BOOT MENU WITH PASSWORD BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to set a password to the GRUB boot menu in an automated fashion. This way you can efficiently prevent unauthorized users from modifying boot parameters, and to have better control over the system boot.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Store your sensitive variables in an encrypted file:

- a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Set the bootloader password
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_password: "{{ pwd }}"
        bootloader_reboot_ok: true
```

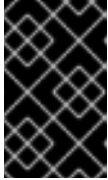
The settings specified in the example playbook include the following:

bootloader_password: "{{ pwd }}"

The variable ensures protection of boot parameters with a password.

bootloader_reboot_ok: true

The role detects that a reboot is required for the changes to take effect and performs a restart of the managed node.



IMPORTANT

Changing the boot loader password is not an idempotent transaction. This means that if you apply the same Ansible playbook again, the result will not be the same, and the state of the managed node will change.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check --ask-vault-pass ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. On your managed node during the GRUB boot menu screen, press the **e** key for edit.



2. You are prompted for a username and a password:

```
Enter username:
root
Enter password:
-
```

Enter username: root

The boot loader username is always **root** and you do not need to specify it in your Ansible playbook.

Enter password: <password>

The boot loader password corresponds to the **pwd** variable that you defined in the **vault.yml** file.

3. You can view or edit configuration of the particular boot loader entry:

```
GRUB version 2.06

load_video
set gfxpayload=keep
insmod gzio
linux ($root)/vmlinuz-5.14.0-362.24.1.el9_3.aarch64 root=/dev/mapper/rhel-r\
oot ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root rd.\
lum.lv=rhel/swap quiet
initrd ($root)/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd

Minimum Emacs-like screen editing is supported. TAB lists
completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
command-line or ESC to discard edits and return to the GRUB menu.
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file
- **/usr/share/doc/rhel-system-roles/bootloader/** directory

31.3. SETTING A TIMEOUT FOR THE BOOT LOADER MENU BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to configure a timeout for the GRUB boot loader menu in an automated way. You can update a period of time to intervene and select a non-default boot entry for various purposes.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configuration and management of the GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Update the boot loader timeout
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_timeout: 10
```

The settings specified in the example playbook include the following:

bootloader_timeout: 10

Input an integer to control for how long the GRUB boot loader menu is displayed before booting the default entry.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Remotely restart your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.reboot
managed-node-01.example.com | CHANGED => {
  "changed": true,
  "elapsed": 21,
  "rebooted": true
}
```

2. On the managed node, observe the GRUB boot menu screen.



The highlighted entry will be executed automatically in 10s

For how long this boot menu is displayed before GRUB automatically uses the default entry.

- Alternative: you can remotely query for the "timeout" settings in the `/boot/grub2/grub.cfg` file of your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a "grep
'timeout' /boot/grub2/grub.cfg"
managed-node-01.example.com | CHANGED | rc=0 >>
if [ x$feature_timeout_style = xy ] ; then
    set timeout_style=menu
    set timeout=10
# Fallback normal timeout code in case the timeout_style feature is
set timeout=10
if [ x$feature_timeout_style = xy ] ; then
    set timeout_style=menu
    set timeout=10
    set orig_timeout_style=${timeout_style}
    set orig_timeout=${timeout}
    # timeout_style=menu + timeout=0 avoids the countdown code keypress check
    set timeout_style=menu
    set timeout=10
    set timeout_style=hidden
    set timeout=10
if [ x$feature_timeout_style = xy ] ; then
    if [ "${menu_show_once_timeout}" ]; then
        set timeout_style=menu
        set timeout=10
        unset menu_show_once_timeout
        save_env menu_show_once_timeout
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md` file
- `/usr/share/doc/rhel-system-roles/bootloader/` directory

31.4. COLLECTING THE BOOT LOADER CONFIGURATION INFORMATION BY USING THE BOOTLOADER RHEL SYSTEM ROLE

You can use the **bootloader** RHEL system role to gather information about the GRUB boot loader entries in an automated fashion. You can use this information to verify the correct configuration of system boot parameters, such as kernel and initial RAM disk image paths.

As a result, you can for example:

- Prevent boot failures.
- Revert to a known good state when troubleshooting.
- Be sure that security-related kernel command-line parameters are correctly configured.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configuration and management of GRUB boot loader
  hosts: managed-node-01.example.com
  tasks:
    - name: Gather information about the boot loader configuration
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_gather_facts: true

    - name: Display the collected boot loader configuration information
      debug:
        var: bootloader_facts
```

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- After you run the preceding playbook on the control node, you will see a similar command-line output as in the following example:

```
...
  "bootloader_facts": [
    {
      "args": "ro crashkernel=1G-4G:256M,4G-64G:320M,64G-:576M rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params quiet",
      "default": true,
      "id": "2c9ec787230141a9b087f774955795ab-5.14.0-362.24.1.el9_3.aarch64",
      "index": "1",
      "initrd": "/boot/initramfs-5.14.0-362.24.1.el9_3.aarch64.img $tuned_initrd",
      "kernel": "/boot/vmlinuz-5.14.0-362.24.1.el9_3.aarch64",
      "root": "/dev/mapper/rhel-root",
      "title": "Red Hat Enterprise Linux (5.14.0-362.24.1.el9_3.aarch64) 9.4 (Plow)"
    }
  ]
...
```

The command-line output shows the following notable configuration information about the boot entry:

args

Command-line parameters passed to the kernel by the GRUB2 boot loader during the boot process. They configure various settings and behaviors of the kernel, initramfs, and other boot-time components.

id

Unique identifier assigned to each boot entry in a boot loader menu. It consists of machine ID and the kernel version.

root

The root filesystem for the kernel to mount and use as the primary filesystem during the boot.

Additional resources

- [/usr/share/ansible/roles/rhel-system-roles.bootloader/README.md](#) file
- [/usr/share/doc/rhel-system-roles/bootloader/](#) directory
- [Understanding boot entries](#)

CHAPTER 32. USING ADVANCED ERROR REPORTING

When you use the **Advanced Error Reporting (AER)**, you receive notifications of error events for **Peripheral Component Interconnect Express (PCIe)** devices. RHEL enables this kernel feature by default and collects the reported errors in the kernel logs. If you use the **rasdaemon** program, these errors are parsed and stored in its database.

32.1. OVERVIEW OF AER

Advanced Error Reporting (AER) is a kernel feature that provides enhanced error reporting for **Peripheral Component Interconnect Express (PCIe)** devices. The **AER** kernel driver attaches root ports which support **PCIe AER** capability in order to:

- Gather the comprehensive error information
- Report errors to the users
- Perform error recovery actions

When **AER** captures an error, it sends an *error* message to the console. For a repairable error, the console output is a *warning*.

Example 32.1. Example AER output

```
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:  device [8086:2030] error
status/mask=000000c0/00002000
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:  [ 6] Bad TLP
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:  [ 7] Bad DLLP
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:  device [8086:2030] error
status/mask=00000040/00002000
```

32.2. COLLECTING AND DISPLAYING AER MESSAGES

To collect and display AER messages, use the **rasdaemon** program.

Procedure

1. Install the **rasdaemon** package.

```
# yum install rasdaemon
```

2. Enable and start the **rasdaemon** service.

```
# systemctl enable --now rasdaemon
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/rasdaemon.service →  
/usr/lib/systemd/system/rasdaemon.service.
```

3. Issue the **ras-mc-ctl** command.

```
# ras-mc-ctl --summary
```

```
# ras-mc-ctl --errors
```

The command displays a summary of the logged errors (the **--summary** option) or displays the errors stored in the error database (the **--errors** option).

Additional resources

- The **rasdaemon(8)** manual page
- The **ras-mc-ctl(8)** manual page