Red Hat Enterprise Linux 7

内核管理指南

使用模块、kpatch 或 kdump 在 RHEL 中管理 Linux 内核

Last Updated: 2024-05-23

Red Hat Enterprise Linux 7 内核管理指南

使用模块、kpatch 或 kdump 在 RHEL 中管理 Linux 内核

Jaroslav Klech
Red Hat Customer Content Services
jklech@redhat.com

Sujata Kurup
Red Hat Customer Content Services
skurup@redhat.com

Khushbu Borole
Red Hat Customer Content Services
kborole@redhat.com

Marie Dolezelova
Red Hat Customer Content Services

Mark Flitter
Red Hat Customer Content Services

Douglas Silas
Red Hat Customer Content Services

Eliska Slobodova
Red Hat Customer Content Services

Jaromir Hradilek
Red Hat Customer Content Services

Maxim Svistunov
Red Hat Customer Content Services

Robert Krátký
Red Hat Customer Content Services

Stephen Wadeley
Red Hat Customer Content Services

Florian Nadge
Red Hat Customer Content Services

法律通告

摘要

内核管理指南记录了维护红帽企业 Linux 7 内核的任务。这个版本，包括使用 kpatch、管理内核模块和手动更新内核的信息。

目录

前言 4

第 1 章 使用内核模块 5
1.1. 什么是内核模块？ 5
1.2. 内核模块依赖关系 5
1.3. 列出当前加载的模块 6
1.4. 显示模块信息 7
1.5. 在系统运行时载入内核模块 7
1.6. 在系统运行时卸载内核模块 8
1.7. 在系统引导时自动载入内核模块 9
1.8. 防止在系统引导时自动载入内核模块 10
1.9. 为安全引导签名内核模块 12

第 2 章 使用 SYSCTL 和内核可调项 19
2.1. 什么是内核可调项？ 19
2.2. 如何使用内核可调项 19
2.3. 可以控制哪些可调项？ 20

第 3 章 内核参数和值列表 33
3.1. 内核命令行参数 33

第 4 章 内核特性 36
4.1. CONTROL GROUPS 36
4.2. 内核源检查器 37
4.3. 直接访问文件(DAX) 37
4.4. 用户空间的内存保护密钥（也称为 PKU 或 PKEYS） 37
4.5. 内核请求空间布局随机化 38
4.6. 高级错误报告(AER) 38

第 5 章 手动升级内核 40
5.1. 内核软件包概述 40
5.2. 准备升级 41
5.3. 下载升级的内核 42
5.4. 执行升级 42
5.5. 验证初始 RAM 文件系统镜像 43
5.6. 验证引导装载程序 46

第 6 章 使用内核实时修补程序应用补丁 47
6.1. KPATCH 的限制 47
6.2. 对第三方实时补丁的支持 47
6.3. 获得内核实时补丁 48
6.4. 组件内核实时修补 48
6.5. 内核实时补丁如何工作 48
6.6. 启用内核实时补丁 49
6.7. 更新内核补丁模块 50
6.8. 禁用内核实时补丁 51

第 7 章 内核崩溃转储指南 55
7.1. KDUMP 简介 55
7.2. 安装和配置 KDUMP 56
7.3. 为 KDUMP 将内核驱动程序列入黑名单 64
7.4. 测试 KDUMP 配置 65
7.5. 固件支持的转储机制 66

7.6. 分析内核转储 68
7.7. 常见问题解答 73
7.8. 支持的 KDUMP 配置和目标 75
7.9. 使用 KEXEC 重启内核 80
7.10. 与 KDUMP 相关的门户实验 80

第 8 章 使用内核完整性子系统提高安全性 82
8.1. 内核完整性子系统 82
8.2. 完整性测量架构 82
8.3. 扩展的验证模块 83
8.4. 可信和加密的密钥 83
8.5. 启用完整性测量架构和扩展验证模块 83
8.6. 使用完整性测量架构收集文件哈希 85

第 9 章 修订历史记录 87

前言

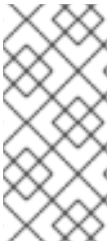
《*内核管理指南*》描述了使用内核并显示了几个实际任务。从有关使用内核模块的信息开始，指南涵盖了与 **sysfs** 设备 的交互、内核手动升级和使用 kpatch。本指南还引入了崩溃转储机制，它在出现内核故障时设置和测试 **vmcore** 集合的过程。

《*内核管理指南*》还包括了管理内核的选定用例，包括有关命令行选项、内核可调项（也称为交换机）的参考材料，以及对内核功能的简要讨论。

第 1 章 使用内核模块

本章解释了：

- 什么是内核模块.
- 如何使用 **kmod** 实用程序管理模块及其依赖项。
- 如何配置模块参数以控制内核模块的行为。
- 如何在引导时加载模块.



注意

要使用本章中描述的内核模块工具，请首先确保以 root 用户身份在您的系统中安装 **kmod** 软件包：

```
# yum install kmod
```

1.1. 什么是内核模块？

Linux 内核在设计上都是单体式的。但是，它使用每个用例所需的可选模块或其他模块编译。*这意味着，您可以使用动态加载的内核模块扩展内核的功能。*内核模块可以提供：

- 添加对新硬件支持的设备驱动程序。
- 支持文件系统，如 **GFS2 或 NFS**。

与内核本身一样，模块也可以采用自定义其行为的参数。尽管默认参数在大多数情况下运行良好。对于内核模块，用户空间工具可以执行以下操作：

- 列出当前加载到运行的内核的模块.
- 查询所有可用模块以获取可用的参数和特定于模块的信息。
- 在运行的内核中动态加载或卸载（删除）模块.

这些实用程序中，很多是由 **kmod** 软件包提供的，在执行操作时会考虑模块依赖关系。因此，很少需要手动跟踪依赖关系。

在现代系统中，在需要时，各种机制会自动载入内核模块。不过，有时需要手动加载或卸载模块。例如，当一个模块优先于另一个模块时，虽然其中一个模块能够提供基本功能，或者在模块执行意外运行时仍然首选。

1.2. 内核模块依赖关系

某些内核模块有时依赖一个或多个内核模块。**/lib/modules/<KERNEL_VERSION>/modules.dep** 文件包含对应内核版本的完整内核模块依赖关系列表。

依赖项文件由 **depmod** 程序生成，该程序是 **kmod** 软件包的一部分。**kmod** 提供的许多实用程序在执行操作时会考虑模块依赖关系，因此很少需要 **手动** 跟踪依赖项。

警告

内核模块的代码在内核空间中是在不受限制模式下执行的。因此，您应该了解您载入的模块。

其它资源

- 有关 **/lib/modules/<KERNEL_VERSION>/modules.dep** 的更多信息，请参阅 **modules.dep(5)** 手册页。
- 有关具体详情，包括 Compopsis 和 **depmod** 选项，请参阅 **depmod(8)手册**页。

1.3. 列出当前加载的模块

您可以通过运行 **lsmod** 命令列出当前载入内核中的所有内核模块，例如：

```
# lsmod
Module                  Size  Used by
tcp_lp                 12663  0
bnep                   19704  2
bluetooth             372662  7 bnep
rfkill                 26536  3 bluetooth
fuse                   87661  3
ebtable_broute         12731  0
bridge                110196  1 ebtable_broute
stp                    12976  1 bridge
llc                    14552  2 stp,bridge
ebtable_filter         12827  0
ebtables               30913  3 ebtable_broute,ebtable_nat,ebtable_filter
ip6table_nat           13015  1
nf_nat_ipv6            13279  1 ip6table_nat
iptable_nat            13011  1
nf_conntrack_ipv4      14862  4
nf_defrag_ipv4         12729  1 nf_conntrack_ipv4
nf_nat_ipv4            13263  1 iptable_nat
nf_nat                 21798  4 nf_nat_ipv4,nf_nat_ipv6,ip6table_nat,iptable_nat
[output truncated]
```

lsmod 输出指定了三列：

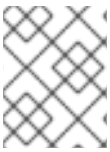
- **模块**
  - 当前加载在内存中的内核模块的名称。
- **Size**
  - 内核模块使用的内存量，以 KB 为单位。
- **使用的**
  - 个十进制数字，表示在 **Module** 字段中存在多少个依赖项。

- **由依赖模块名称组成的逗号分隔字符串**。使用这个列表，您可以首先根据您要卸载的模块卸载所有模块。

最后，请注意 **lsmod** 输出比较详细，且比 **/proc/modules pseudo-** file 的内容更容易阅读。

1.4. 显示模块信息

您可以使用 **modinfo *<MODULE_NAME>*** 命令显示内核模块的详细信息。



注意

当在其中一个 **kmod** 实用程序中输入内核模块名称作为参数时，不要在名称末尾附加 a **.ko** 扩展名。内核模块名称没有扩展名，它们对应的文件有。

例 1.1. 使用 lsmod 列出内核模块的信息

要显示 **e1000e 模块**（Intel PRO/1000 网络驱动程序）的信息，以 **root** 用户身份输入以下命令：

```
# modinfo e1000e
filename:       /lib/modules/3.10.0-
121.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000e/e1000e.ko
version:        2.3.2-k
license:        GPL
description:    Intel(R) PRO/1000 Network Driver
author:         Intel Corporation,
```

1.5. 在系统运行时载入内核模块

扩展 Linux 内核功能的最佳方法是加载内核模块。下面的步骤描述了如何使用 **modprobe** 命令在当前运行的内核中查找并载入内核模块。

先决条件

- root 权限。
- 已安装 **kmod** 软件包。
- 相关的内核模块没有被加载。为确保情况确实如此，请参阅 列出当前载入的模块。

流程

1. 选择您要载入的内核模块。
   模块位于 **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** 目录中。
2. 载入相关内核模块：

```
# modprobe <MODULE_NAME>
```

注意

在输入内核模块的名称时，不要将 **.ko.xz** 扩展附加到名称的末尾。内核模块名称没有扩展名，它们对应的文件有。

3. （可选）验证载入了相关模块：

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

如果正确加载了模块，这个命令会显示相关的内核模块。例如：

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

重要

重启系统后，这个过程中描述的更改**不会保留**。**有关如何在系统重启后载入内核模块使其持续的信息 ???，请参阅在系统引导时自动载入内核模块**。

其它资源

- 有关 **modprobe** 的详情，请查看 **modprobe(8)手册**页。

1.6. 在系统运行时卸载内核模块

有时，您发现您需要从运行的内核中卸载某些内核模块。下面的步骤描述了如何使用 **modprobe** 命令在当前载入的内核运行时在系统运行时查找和卸载内核模块。

先决条件

- root 权限。
- 已安装 **kmod** 软件包。

流程

1. 执行 **lsmod** 命令并选择您要卸载的内核模块。
   如果内核模块有依赖项，请在卸载内核模块前卸载它们。有关使用依赖项识别模块的详情，请参阅列出当前载入的模块和 内核模块依赖项。

2. 卸载相关内核模块：

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

在输入内核模块的名称时，不要将 **.ko.xz** 扩展附加到名称的末尾。内核模块名称没有扩展名，它们对应的文件有。

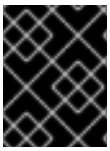
警告

当它们被正在运行的系统使用时，不要卸载这些内核模块。这样做可能会导致不稳定，或系统无法正常操作。

3. （可选）验证相关模块是否已卸载：

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

如果模块被成功卸载，这个命令不会显示任何输出。



重要

完成这个过程后，被定义为在引导是自动加载的内核模块，在重启系统后**将不会处于卸载状态**。有关如何应对这个结果的详情，请参考防止在系统引导时自动载入内核模块。

其它资源

- 有关 **modprobe** 的详情，请查看 **modprobe(8)手册**页。

1.7. 在系统引导时自动载入内核模块

下面的步骤描述了如何配置内核模块以便在引导过程中自动载入该模块。

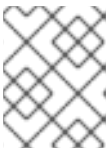
先决条件

- root 权限。
- 已安装 **kmod** 软件包。

流程

1. 选择您要在引导过程中载入的内核模块。
模块位于 **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** 目录中。
2. 为模块创建配置文件：

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



注意

在输入内核模块的名称时，不要将. **ko.xz** 扩展附加到名称的末尾。内核模块名称没有扩展名，它们对应的文件有。

3. 另外，重启后，验证载入了相关模块：

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

上面的示例命令应该成功并显示相关的内核模块。

重要

重启系统后，这个过程中描述的更改**将会保留**。

其它资源

- 有关在引导过程中载入内核模块的详情，请查看 module **-load.d(5)** 手册页。

1.8. 防止在系统引导时自动载入内核模块

下面的步骤描述了如何在 denylist 中添加内核模块使其不会在引导过程中自动载入。

先决条件

- root 权限。
- 已安装 **kmod** 软件包。
- 确定 denylist 中的内核模块对您当前系统配置并不重要。

流程

1. 选择您要放入 denylist 中的内核模块：

```
$ lsmod

Module                  Size  Used by
fuse                  126976  3
xt_CHECKSUM            16384  1
ipt_MASQUERADE         16384  1
uinput                 20480  1
xt_conntrack           16384  1
…
```

lsmod 命令显示载入到当前运行的内核的模块列表。

- 或者，找到您要防止载入的未加载内核模块。
  所有内核模块都位于 **/lib/modules/<KERNEL_VERSION>/kernel/<SUBSYSTEM>/** 目录中。

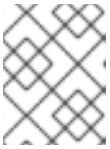
2. 为 denylist 创建配置文件：

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

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

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

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

示例中显示了由 **vim** 编辑器编辑的 **blacklist.conf** 文件的内容。**黑名单** 行可确保在引导过程中不会自动加载相关内核模块。但是，**黑名单** 命令不会阻止将模块作为不在 denylist 中的另一个内核模块的依赖项加载。因此，**install 行** 会导致 **/bin/false** 运行而不是安装模块。

以 hash 符号开头的行是注释以便更易读。



注意

在输入内核模块的名称时，不要将 **ko.xz** 扩展附加到名称的末尾。内核模块名称没有扩展名，它们对应的文件有。

3. 在重新构建前，为当前初始 ramdisk 镜像创建备份副本：

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

如果新版本出现意外问题，以上命令会创建一个备份 **initramfs** 镜像。

- 另外，还可创建其它初始 ramdisk 镜像的备份副本，该副本与您要将内核模块放入 denylist 中的内核版本对应：

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

4. 根据更改生成新的初始 ramdisk 镜像：

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

- 如果您要为当前引导的不同内核版本构建初始 ramdisk 镜像，请指定目标 **initramfs** 和内核版本：

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

5. 重启系统：

```
$ reboot
```

重要

此流程中描述的更改将在重启后**生效并保留**。如果您将关键内核模块错误地放入 denylist 中，您会遇到不稳定的情况或系统无法正常工作。

其它资源

- 如需有关 **dracut** 实用程序的更多详细信息，请参阅 **dracut(8)手册**页。

- 有关防止在 Red Hat Enterprise Linux 8 及更早的版本中系统引导时自动载入内核模块的更多信息，请参阅 如何防止内核模块自动载入？

1.9. 为安全引导签名内核模块

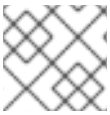
Red Hat Enterprise Linux 7 包括对 UEFI 安全引导功能的支持，这意味着可以在启用了 UEFI 安全引导的系统上安装并运行 Red Hat Enterprise Linux 7。请注意，Red Hat Enterprise Linux 7 不需要在 UEFI 系统中使用安全引导机制。

如果启用了安全引导机制，则必须使用私钥签名并使用对应的公钥验证 UEFI 操作系统引导装载程序、Red Hat Enterprise Linux 内核和所有内核模块。如果未签名和验证，则不允许系统完成引导过程。

Red Hat Enterprise Linux 7 发行版本包括：

- 签名的引导装载程序
- 签名的内核
- 签名的内核模块

此外，签名的第一阶段引导装载程序和签名的内核包括嵌入式红帽公钥。这些签名的可执行二进制文件和嵌入式密钥可让 Red Hat Enterprise Linux 7 在支持 UEFI 安全引导引导的系统中使用 Microsoft UEFI 安全引导认证机构密钥安装、引导和运行这些密钥。



注意

并非所有基于 UEFI 的系统都包括对安全引导的支持。

以下小节中提供的信息描述了在启用了安全引导机制的基于 UEFI 的构建系统上用于 Red Hat Enterprise Linux 7 的自签名专用内核模块的步骤。这些部分还提供了将您的公钥导入到要部署内核模块的目标系统中的可用选项概述。

要签名并载入内核模块，您需要：

1. 在您的系统上安装了相关实用程序。
2. 验证内核模块.
3. 生成公钥和私钥对。
4. 在目标系统中导入公钥。
5. 使用私钥签署内核模块。
6. 加载签名的内核模块。

1.9.1. 先决条件

要能够为外部构建的内核模块签名，请在构建系统上安装下表中列出的实用程序。

表 1.1. 所需工具

工具	由软件包提供	用于	目的
openssl	**openssl**	构建系统	生成公共和专用 X.509 密钥对
Sign-file	**kernel-devel**	构建系统	用于为内核模块签名的 Perl 脚本
perl	**perl**	构建系统	用于运行签名脚本的 Perl 解释器
mokutil	**mokutil**	目标系统	用于手动注册公钥的可选工具
keyctl	**keyutils**	目标系统	用于在系统密钥环中显示公钥的可选工具

注意

构建系统（构建和签署内核模块）不需要启用 UEFI 安全引导，甚至不需要是基于 UEFI 的系统。

1.9.2. 内核模块验证

在 Red Hat Enterprise Linux 7 中，加载内核模块时，将使用内核系统密钥环中的公共 X.509 密钥来检查模块的签名，不包括内核的系统黑名单密钥环中的密钥。以下小节概述了密钥/密钥环的来源，如系统中不同源的载入密钥示例。此外，用户可以查看验证内核模块所需的一切。

1.9.2.1. 用于验证内核模块的公钥源

在启动过程中，内核从一组持久密钥存储中将 X.509 密钥加载到系统密钥环或系统黑名单密钥环中，如下表中所示。

表 1.2. 系统密钥环的源

X.509 密钥源	用户添加密钥的功能	UEFI 安全引导状态	引导过程中载入的密钥
嵌入于内核中	否	-	**.system_keyring**
UEFI 安全引导 "db"	有限	未启用	否
		Enabled	**.system_keyring**
UEFI 安全引导 "dbx"	有限	未启用	否
		Enabled	**.system_keyring**
嵌入 in **shim.efi** 引导装载程序	否	未启用	否

X.509 密钥源	用户添加密钥的功能	UEFI 安全引导状态	引导过程中载入的密钥
		Enabled	**.system_keyring**
Machine Owner Key（MOK）列表	是	未启用	否
		Enabled	**.system_keyring**

如果没有启用 UEFI 安全引导，或者没有启用 UEFI 安全引导，则只有嵌入内核中的密钥才会加载到系统密钥环中。在这种情况下，您无法在不重新构建内核的情况下添加这组密钥。

系统黑名单密钥环是一个已撤销的 X.509 密钥列表。如果您的模块由黑名单中的密钥签名，那么即使您的公钥位于系统密钥环中，它也会失败身份验证。

您可以使用 **keyctl** 实用程序显示系统密钥环上的密钥信息。以下是未启用 UEFI 安全引导的 Red Hat Enterprise Linux 7 系统的一个简化示例输出。

```
# keyctl list %:.system_keyring
3 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...
```

以下是启用了 UEFI 安全引导的 Red Hat Enterprise Linux 7 系统的一个简化示例输出。

```
# keyctl list %:.system_keyring
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...
```

以上输出显示了从 UEFI 安全引导 "db" 密钥以及 **红帽安全引导（CA 密钥 1）中添加了两个密钥**，这些密钥嵌入在 **shim.efi** 启动加载器中。您还可以使用 UEFI 安全引导相关源查找内核控制台信息。这包括 UEFI 安全引导 db、内嵌的 shim 和 MOK 列表。

```
# dmesg | grep 'EFI: Loaded cert'
[5.160660] EFI: Loaded cert 'Microsoft Windows Production PCA 2011: a9290239...
[5.160674] EFI: Loaded cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309b...
[5.165794] EFI: Loaded cert 'Red Hat Secure Boot (CA key 1): 4016841644ce3a8...
```

1.9.2.2. 内核模块验证要求

这部分论述了在启用了 UEFI 安全引导功能的系统中载入内核模块必须满足的条件。

如果启用了 UEFI 安全引导，或者指定了 **模块.sig_enforce** 内核参数，则只能加载使用系统密钥环中密钥进行身份验证的已签名内核模块。此外，公钥不得位于系统黑名单密钥环中。

如果禁用了 UEFI 安全引导，并且没有指定 **模块.sig_enforce** 内核参数，您可以加载未签名的内核模块和签名的内核模块，且没有公钥。下表总结了这一点。

表 1.3. 加载内核模块的验证要求

模块已签名	找到公钥，且签名有效	UEFI 安全引导状态	sig_enforce	模块载入	内核污点
未签名	-	未启用	未启用	成功	是
		未启用	Enabled	Fails	-
		Enabled	-	Fails	-
已签名	否	未启用	未启用	成功	是
		未启用	Enabled	Fails	-
		Enabled	-	Fails	-
已签名	是	未启用	未启用	成功	否
		未启用	Enabled	成功	否
		Enabled	-	成功	否

1.9.3. 生成公共和专用 X.509 密钥对

您需要生成一个公共和私有 X.509 密钥对，才能成功在启用了安全引导的系统上使用内核模块。之后您将使用私钥为内核模块签名。您还必须将对应的公钥添加到用于安全引导的 Machine Owner Key(MOK)中，以验证签名的模块。具体步骤请查看 第 1.9.4.2 节 “系统管理员手动将公钥添加到 MOK 列表中”。

这个密钥对生成的一些参数最好用配置文件指定。

1. 使用密钥对生成参数创建配置文件：

```
# cat << EOF > configuration_file.config
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
O = Organization
CN = Organization signing key
emailAddress = E-mail address

[ myexts ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
EOF
```

2. 如以下示例所示，创建 X.509 公钥和私钥对：

```
# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 \
-batch -config configuration_file.config -outform DER \
-out my_signing_key_pub.der \
-keyout my_signing_key.priv
```

公钥将写入 ***my_signing_key_pub*.der** 文件，私钥将写入 ***my_signing_key*.priv** 文件中。

3. 在您要验证并载入内核模块的所有系统中注册您的公钥。
详情请查看 第 1.9.4 节 “在目标系统中注册公钥”。

警告

应用强大的安全措施和访问策略来保护您的私钥内容。对于一个恶意的用户，可以使用这个密钥破坏所有由对应公钥验证的系统。

1.9.4. 在目标系统中注册公钥

当 Red Hat Enterprise Linux 7 在启用了安全引导机制的基于 UEFI 的系统中引导时，内核将加载到系统密钥环中安全引导 db 密钥数据库中但不在已撤销密钥的 dbx 数据库中的所有公钥。以下小节描述了在目标系统上导入公钥的不同方式，以便系统密钥环可以使用公钥来验证内核模块。

1.9.4.1. 工厂固件映像（包括公钥）

为了便于对系统中的内核模块进行身份验证，请您的系统供应商将公钥合并到其工厂固件镜像中的 UEFI 安全引导密钥数据库中。

1.9.4.2. 系统管理员手动将公钥添加到 MOK 列表中

Machine Owner Key(MOK)功能可以用来扩展 UEFI 安全引导密钥数据库。当 Red Hat Enterprise Linux 7 在启用了安全引导机制的启用了 UEFI 的系统中引导时，MOK 列表中的密钥除密钥数据库中的密钥外也会添加到系统密钥环中。和安全引导数据库密钥相似，MOK 列表密钥会被安全地永久存储。但它们是两个独立的工具。MOK 工具由 **shim.efi**、**MokManager.efi**、**grubx64.efi** 和红帽企业 Linux 7 **mokutil** 实用程序支持。

注册 MOK 密钥需要用户在每个目标系统中在 UEFI 系统控制台上手动互动。MOK 工具为测试新生成的密钥对以及与其签注的内核模块提供了方便的方法。

将您的公钥添加到 MOK 列表中：

1. 请在 MOK 列表中添加您的公钥：

```
# mokutil --import my_signing_key_pub.der
```

会要求您输入并确认此 MOK 注册请求的密码。

2. 重启机器。

待处理的 MOK 密钥注册请求将由 **shim.efi 通知**，它将启动 **MokManager.efi**，以便您从 UEFI 控制台完成注册。

3. 输入您之前与此请求关联的密码并确认注册。
   您的公钥已添加到 MOK 列表中，这是永久的。

密钥位于 MOK 列表中后，它会在启用 UEFI 安全引导时自动传播到此列表中的系统密钥环，并在随后引导时自动传播到系统密钥环。

1.9.5. 使用私钥签名内核模块

假设您的内核模块已就绪：

- 使用 Perl 脚本使用您的私钥为内核模块签名：

```
# perl /usr/src/kernels/$(uname -r)/scripts/sign-file \
sha256 \
my_signing_key.priv\
my_signing_key_pub.der\
my_module.ko
```

注意

Perl 脚本要求您同时提供包含私钥和公钥的文件，以及您要签名的内核模块文件。

您的内核模块采用 ELF 镜像格式，Perl 脚本计算，并将签名直接附加到内核模块文件中的 ELF 镜像中。**modinfo** 实用程序可用于显示有关内核模块签名的信息（如果存在）。有关使用 **modinfo** 的详情请参考 第 1.4 节 “显示模块信息”。

附加的签名不包含在 ELF 镜像部分，不是 ELF 镜像的一个正式部分。因此，**readelf** 等实用程序将无法在内核模块中显示签名。

您的内核模块现在可以被加载。请注意，您签名的内核模块也可以在禁用 UEFI 安全引导的系统或非 UEFI 系统中加载。这意味着您不需要同时提供内核模块的签名和未签名版本。

1.9.6. 载入签名的内核模块

且您的公钥注册并处于系统密钥环中，请使用 **mokutil** 将您的公钥添加到 MOK 列表中。然后，使用 **modprobe** 命令手动加载内核模块。

1. （可选）在注册公钥前验证您的内核模块是否不会被加载。
   有关如何列出当前载入的内核模块的详情请参考 第 1.3 节 “列出当前加载的模块”。

2. 在当前引导中验证哪些密钥被添加到系统密钥环中：

```
# keyctl list %:.system_keyring
```

   由于您的公钥尚未注册，因此不应显示在 命令的输出中。

3. 申请注册您的公钥：

```
# mokutil --import my_signing_key_pub.der
```

4. 重新引导，并在 UEFI 控制台中完成注册：

```
# reboot
```

5. 再次验证系统密钥环中的密钥：

```
# keyctl list %:.system_keyring
```

6. 将模块复制到您想要的内核的 **/extra/** 目录中：

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

7. 更新模块依赖项列表：

```
# depmod -a
```

8. 载入内核模块并确认它已被成功载入：

```
# modprobe -v my_module
# lsmod | grep my_module
```

   a. 另外，要在引导时载入模块，将其添加到 **/etc/modules-loaded.d/my_module.conf** 文件中：

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

第 2 章 使用 SYSCTL 和内核可调项

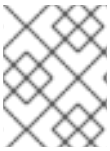
2.1. 什么是内核可调项？

内核可调项用于在引导时或系统运行时按需自定义 Red Hat Enterprise Linux 的行为。某些硬件参数仅在系统启动时指定，且在系统运行后无法更改，但大部分情况下，可以根据需要进行更改，并在下一次启动时进行永久性设置。

2.2. 如何使用内核可调项

可以通过三种方式修改内核可调项：

1. 使用 **sysctl** 命令
2. 通过手动修改 **/etc/sysctl.d/** 目录中的配置文件
3. 通过 shell 与挂载于 **/proc/sys**的虚拟文件系统交互



注意

并非所有引导时间参数都受 sysfs 子系统的控制，必须在内核命令行上设置一些特定于硬件的选项，本指南的内核参数部分解决了这些选项

2.2.1. 使用 sysctl 命令

sysctl 命令用于列出、读取和设置内核可调项。在临时或永久地列出或读取和设置可调项时，它可以过滤可调项。

1. 列出变量

```
# sysctl -a
```

2. 读取变量

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

3. 临时编写变量

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

4. 永久编写变量

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

2.2.2. 修改 /etc/sysctl 中的文件.

要覆盖启动时的默认值，您也可以手动填充 **/etc/sysctl.d** 中的文件。

1. 在 **/etc/sysctl.d**中创建一个新文件

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

-

2. 包括您想要设置的变量（每行一个），格式如下

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

3. 保存文件

4. 重启机器以使更改生效
   或
   Execute **sysctl -p /etc/sysctl.d/99-custom.conf** 以在不重启的情况下应用更改

2.3. 可以控制哪些可调项？

可调项按内核 sybsystem 划分为多个组。Red Hat Enterprise Linux 系统有以下类型的可调项：

表 2.1. sysctl 接口表

class	子系统
abi	执行域和个人
crypto	加密接口
debug	内核调试接口
dev	设备特定信息
fs	全局和特定文件系统可调项
kernel	全局内核 tunables
net	网络 tunables
sunrpc	Sun 远程过程调用(NFS)
user	用户命名空间限制
vm	调整和管理内存、缓冲区和缓存

2.3.1. 网络接口可调项

系统管理员能够通过网络可调项调整正在运行的系统上的网络配置。

网络可调项包含在 **/proc/sys/net** 目录中，其中包含用于各种网络主题的多个子目录。若要调整网络配置，系统管理员需要修改这些子目录中的文件。

最常用的目录有：

1. **/proc/sys/net/core/**

2. **/proc/sys/net/ipv4/**

/proc/sys/net/core/ 目录包含各种设置，这些设置控制内核和网络层之间的交互。通过调整其中一些可调项，您可以提高系统的性能，例如通过增加接收队列的大小、增大接收队列的最大连接数或专用于网络接口的内存。请注意，系统性能取决于每个问题的不同方面。

/proc/sys/net/ipv4/ 目录包含额外的网络设置，这些设置在防止系统上的攻击或使用系统作为路由器时非常有用。目录同时包含 IP 和 TCP 变量。有关这些变量的详细解释，请参阅 **/usr/share/doc/kernel-doc-<version>/Documentation/networking/ip-sysctl.txt**。

/proc/sys/net/ipv4/ 目录中的其他目录涵盖了网络堆栈的不同方面：

1. **/proc/sys/net/ipv4/conf/** - 允许您以不同的方式配置每个系统接口，包括对未配置的设备和设置使用默认设置来覆盖所有特殊配置

2. **/proc/sys/net/ipv4/neigh/** - 包含与直接连接到系统的主机通信的设置，还包含多个步骤的不同系统设置

3. **/proc/sys/net/ipv4/route/** - 包含适用于系统上任何接口路由的规格

这个网络可调项列表与 IPv4 接口相关，并可通过 **/proc/sys/net/ipv4/{all,<interface_name>}/** 目录访问。

以下参数的描述已从内核文档站点采用。[1]

log_martians

在内核日志中记录地址无法记录的数据包。

类型	Default（默认）
布尔值	0

如果一个或多个 **conf/{all,interface}/log_martians** 设置为 TRUE，则启用

更多资源

- 内核参数 net.ipv4.conf.all.log_martians 是什么？
- 为什么我会在消息文件中看到"martian 源"日志？

accept_redirects

接受 ICMP 重定向消息。

类型	Default（默认）
布尔值	1

该接口的 **accept_redirects** 在以下情况下启用：

- Both **conf/{all,interface}/accept_redirects** 是 TRUE（启用接口转发时）
- 至少有一个 **conf/{all,interface}/accept_redirects** 是 TRUE（接口转发已被禁用）

如需更多信息，请参阅如何启用或禁用 ICMP 重定向

转发

在接口上启用 IP 转发。

类型	Default（默认）
布尔值	0

更多资源

- 打开 Packet 转发和非本地绑定

mc_forwarding

进行多播路由。

类型	Default（默认）
布尔值	0

- 只读值
- 需要多播路由守护进程。
- **conf/all/mc_forwarding** 还必须设置为 TRUE，以便为接口启用多播路由

更多资源

- 有关只读行为的说明，请参阅为何系统在设置内核参数"net.ipv4.conf.all.mc_forwarding"时报告"permission denied on key"？

medium_id

用于根据所附加介质区分设备的任意值。

类型	Default（默认）
整数	0

备注

- 当广播数据包只在其中一个上接收时，同一介质中的两个设备可能具有不同的 ID 值。
- 默认值为 0，表示该设备是其介质的唯一接口
- -1 的值表示介质未知。
- 目前，它被用来更改 proxy_arp 行为：
- proxy_arp 功能针对在附加到不同介质的两个设备之间转发的数据包启用。

其他资源 - 例如，请参阅在 Linux 2.2 和 2.4 中使用"medium_id"功能。

proxy_arp

代理 arp.

类型	Default（默认）
布尔值	0

如果至少有一个 **conf/{all,interface}/proxy_arp** 设置为 TRUE，则 **接口的 proxy** _arp 会被启用

proxy_arp_pvlan

专用 VLAN 代理 arp.

类型	Default（默认）
布尔值	0

允许代理 arp 回复同一接口，以支持 RFC 3069等功能

shared_media

send(router)或接受（主机）RFC1620 共享介质重定向.

类型	Default（默认）
布尔值	1

备注

- 覆盖 secure_redirects.
- 如果至少有一个 **conf/{all,interface}/shared_media** 设置为 **TRUE，接口的 shared** _**media** 会被启用

secure_redirects

仅接受 ICMP 重定向消息到接口的当前网关列表中列出的网关。

类型	Default（默认）
布尔值	1

备注

- 即使禁用，仍然应用 RFC1122 重定向规则。
- 被 shared_media 覆盖。

- 如果至少有一个 **conf/{all,interface}/secure_redirects** 设置为 **TRUE**，则接口的 **secure_redirects** 会被启用

send_redirects

发送重定向（如果路由器）。

类型	Default（默认）
布尔值	1

如果至少有一个 conf/{all,interface}/send_redirects 设置为 TRUE，则会启用接口的 send_redirects

bootp_relay

接受源地址为 0.b.c.d 的数据包，其目的地为本地主机。

类型	Default（默认）
布尔值	0

备注

- 必须启用 BOOTP 守护进程来管理这些数据包
- 还必须将 **conf/all/bootp_relay** 设置为 TRUE，以便为接口启用 BOOTP 转发
- 未实施，请参阅 Red Hat Enterprise Linux 网络指南中的 DHCP 转发代理

accept_source_route

通过 SRR 选项接受数据包。

类型	Default（默认）
布尔值	1

备注

- 还必须将 **conf/all/accept_source_route** 设置为 TRUE，以接受接口上的 SRR 选项的数据包

accept_local

接受具有本地源地址的数据包.

类型	Default（默认）
布尔值	0

备注

- 结合合适的路由，这可用于通过线路在两个本地接口之间定向数据包，并让它们正确接受。
- **rp_filter** 必须设置为非零值，以便 accept_local 生效。

route_localnet

在路由时，请勿将环回地址视为 martian 源或目的地。

类型	Default（默认）
布尔值	0

备注

- 这允许将 127/8 用于本地路由。

rp_filter

启用源验证

类型	Default（默认）
整数	0

值	效果
0	没有源验证。
1	RFC3704 Strict Reverse Path 中定义的严格模式
2	松散模式，如 RFC3704 Loose Reverse Path 中定义的

备注

- RFC3704 中的当前推荐做法是启用严格的模式，以防止 IP 欺骗 DDos 攻击。
- 如果使用非对称路由或其他复杂的路由，则建议使用松散模式。
- 在 { **interface} 上进行源验证时，使用来自conf/{all,interface}/rp_filter** 的最高值。

arp_filter

类型	Default（默认）
布尔值	0

值	效果
0	（默认）内核可使用来自其他接口的地址响应 arp 请求。这通常会有意义，因为它会增加成功通信的机率。
1	允许在同一子网上具有多个网络接口，并且每个接口的 ARP 应答取决于内核是否将 ARP 的 IP 中的数据包路由出该接口（因此，您必须使用基于源的路由才能使此接口正常工作）。换句话说，它允许控制响应 arp 请求的卡片（通常是 1）。

备注

- IP 地址归 Linux 上的完整主机所有，而不是由特定接口所有。只有对于更复杂的设置（如负载平衡），这个行为才会造成问题。
- 如果至少有一个 **conf/{all,interface}/arp_filter** 设置为 **TRUE，则 接口的 arp _filter** 会被启用

arp_announce

在接口上发送的 ARP 请求中从 IP 数据包发布本地源 IP 地址时定义不同的限制级别

类型	Default（默认）
整数	0

值	效果
0	（默认）使用任何本地地址，在任何接口上配置
1	尝试避免此接口不在目标子网中的本地地址。当通过此接口访问的目标主机需要 ARP 请求中的源 IP 地址作为其接收接口上配置的逻辑网络的一部分时，此模式很有用。在生成请求时，我们将检查我们包括目标 IP 的所有子网，并在源地址来自此类子网时保留源地址。如果没有这样的子网，则根据级别 2 的规则选择源地址。
2	始终为此目标使用最佳的本地地址。在此模式下，我们忽略 IP 数据包中的源地址，并尝试选择我们喜欢与目标主机通信的本地地址。通过在包含目标 IP 地址的传出接口上在所有子网上查找主 IP 地址来选择此类本地地址。如果没有找到合适的本地地址，我们将选择我们在传出接口上或所有其他接口上具有的第一个本地地址，我们希望我们收到回复，即使我们宣布的源 IP 地址是什么。

备注

- 使用来自 **conf/{all,interface}/arp_announce** 的最大值。
- 提高限制级别，为从已解析的目标处收到答案，同时降低级别宣布更有效的发送者的信息提供了更多机会。

arp_ignore

定义不同的发送回复模式，以响应解析本地目标 IP 地址的接收 ARP 请求

类型	Default（默认）
整数	0

值	效果
0	（默认）：回复任何本地目标 IP 地址，在任何接口上配置
1	仅当目标 IP 地址是传入接口上配置的本地地址时才回复
2	仅当目标 IP 地址是传入接口上配置的本地地址并且发件人 IP 地址是此接口上同一子网中的一部分时才回复
3	不要回复通过范围主机配置的本地地址，只回复全局地址和链接地址的解决方案
4-7	保留
8	在 {interface} 上收到 ARP 请求时，请勿回复来自 conf/{all,interface}/arp_ignore 的最大值。

备注

arp_notify

定义用于地址和设备更改通知模式。

类型	Default（默认）
布尔值	0

值	效果
0	不做任何操作
1	在设备启动或硬件地址更改时生成无饱和的 arp 请求。

备注

arp_accept

定义 ARP 表中尚不存在的粒度 ARP 帧的行为

类型	Default（默认）
布尔值	0

值	效果
0	不要在 ARP 表中创建新条目
1	在 ARP 表中创建新条目。

注意
如果此设置开启，回复和请求类型的粒度 arp 都会触发要更新的 ARP 表。如果 ARP 表已包含粒度 arp 帧的 IP 地址，则会更新 arp 表，不论此设置是开启还是关闭。

app_solicit

在退回到多播探测前通过网络链接发送到用户空间 ARP 守护进程的最大探测数（请参见 mcast_solicit）。

类型	Default（默认）
整数	0

备注
请参阅 **mcast_solicit**

disable_policy

禁用此接口的 IPSEC 策略(SPD)

类型	Default（默认）
布尔值	0

needinfo

disable_xfrm

禁用这个接口上的 IPSEC 加密，无论策略是什么

类型	Default（默认）
布尔值	0

needinfo

igmpv2_unsolicited_report_interval

下一个主动 IGMPv1 或 IGMPv2 报告重新传输的间隔以毫秒为单位。

类型	Default（默认）
整数	10000

备注
Milliseconds

igmpv3_unsolicited_report_interval

下次非请求 IGMPv3 报告重新传输的间隔以毫秒为单位。

类型	Default（默认）
整数	1000

备注
Milliseconds

tag

允许您编写一个数字，根据需要使用。

类型	Default（默认）
整数	0

xfrm4_gc_thresh

为 IPv4 目标缓存条目开始收集垃圾回收的阈值。

类型	Default（默认）
整数	1

注意
在这个值上两次，系统会拒绝新分配。

2.3.2. 全局内核 tunables

系统管理员能够通过全局内核可调项配置和监控正在运行的系统上的常规设置。

全局内核可调项包含在 **/proc/sys/kernel/** 目录中，可直接作为指定控制文件或分组到其他子目录中，以用于各种配置主题。要调整全局内核可调项，系统管理员需要修改控制文件。

以下参数的描述已在内核文档站点中使用。[2]

dmesg_restrict

指明是否禁止非特权用户使用 **dmesg** 命令查看来自内核日志缓冲器的消息。
如需更多信息，请参阅内核 sysctl 文档。

core_pattern

指定核心转储文件模式名称。

最大长度	Default（默认）
128 个字符	"核心"

如需更多信息，请参阅内核 sysctl 文档。

hardlockup_panic

当检测到硬锁定时，控制内核 panic。

类型	值	效果
整数	0	内核在硬锁定时不会 panic
整数	1	硬锁定中的内核 panic

为了 panic，系统需要首先检测硬锁定。检测由 nmi_watchdog 参数控制。

更多资源

- 内核 sysctl 文档
- Softlockup 检测器和硬锁定检测器

softlockup_panic

当检测到软锁定时，控制内核 panic。

类型	值	效果
整数	0	内核在软锁定时不 panic
整数	1	软锁定中的内核 panics

默认情况下，在 RHEL7 上，这个值为 0。

有关 **softlockup_panic** 的更多信息，请参阅 kernel_parameters。

kptr_restrict

指明是否对通过 **/proc** 和其他接口公开内核地址施加限制。

类型	Default（默认）
整数	0

值	效果
0	在打印前对内核地址进行哈希处理
1	在某些情况下，将打印的内核指针替换为 0 的指针
2	无条件地替换打印的内核指针

如需更多信息，请参阅 Kernel sysctl 文档。

nmi_watchdog

控制 x86 系统上的硬锁定检测器。

类型	Default（默认）
整数	0

值	效果
0	禁用锁定检测器
1	启用锁定检测器

硬锁定检测器会监控每个 CPU 是否有响应中断的能力。

如需了解更多详细信息，请参阅 Kernel sysctl 文档。

watchdog_thresh

控制 watchdog **hrtimer**、NMI 事件和软/硬锁定阈值的频率。

默认阈值	软锁定阈值
10 秒	2 * **watchdog_thresh**

将此可调项设置为零可完全禁用锁定检测。

如需更多信息，请参阅内核 sysctl 文档。

panic, panic_on_oops, panic_on_stackoverflow, panic_on_unrecovered_nmi, panic_on_warn, panic_on_rcu_stall, hung_task_panic

这些可调项在哪些情况下指定内核应 panic。
要查看有关一组 **panic** 参数的详情，请参阅 Kernel sysctl 文档。

printk, printk_delay, printk_ratelimit, printk_ratelimit_burst, printk_devkmsg

这些可调项控制记录或打印内核错误消息。

有关 **printk** 参数组的详情，请参阅 Kernel sysctl 文档。

shmall, shmmax, shm_rmid_forced

这些可调项对共享内存的控制限制。
有关 **shm** 参数组的详情，请参考 Kernel sysctl 文档。

threads-max

控制 **fork（）** 系统调用所创建的线程的最大数量。

Min 值	最大值
20	由 FUTEX_TID_MASK(0x3fffff)提供.

按照可用的 RAM 页面检查 threads **-max** 值。如果线程结构占用了太多可用 RAM 页面，readm **-max** 会相应地减少。

如需了解更多详细信息，请参阅 Kernel sysctl 文档。

pid_max

PID 分配总结值.
要查看更多信息，请参阅 Kernel sysctl 文档。

numa_balancing

此参数启用或禁用自动 NUMA 内存均衡。在 NUMA 计算机上，如果 CPU 访问远程内存，则性能会受到影响。
如需了解更多详细信息，请参阅 Kernel sysctl 文档。

numa_balancing_scan_period_min_ms, numa_balancing_scan_delay_ms, numa_balancing_scan_period_max_ms, numa_balancing_scan_size_mb

这些可调项可检测页面是否正确放置到运行任务的本地内存节点。
有关 **numa_balancing_scan** 参数组的详情，请参阅 Kernel sysctl 文档。

[1] https://www.kernel.org/doc/Documentation/

[2] https://www.kernel.org/doc/Documentation/

第 3 章 内核参数和值列表

3.1. 内核命令行参数

内核命令行参数（也称为内核参数）用于在引导时自定义 Red Hat Enterprise Linux 的行为。

3.1.1. 设置内核命令行参数

这部分论述了如何使用 **GRUB2** 引导装载程序更改 AMD64 和 Intel 64 系统和 IBM Power Systems 服务器中的内核命令行参数，以及使用 **zipl** 在 IBM Z 上更改内核命令行参数。

内核命令行参数保存在 **boot/grub/grub.cfg** 配置文件中，该文件由 **GRUB2** 引导加载程序生成。不要编辑这个配置文件。更改此文件仅由配置脚本进行。

为 AMD64 和 Intel 64 系统以及 IBM Power 系统硬件更改 GRUB2 中的内核命令行参数.

1. 以 **root** 用户身份使用文本编辑器（如 **vim** 或 **Gedit** ）打开 **/etc/default/grub** 配置文件。
2. 在这个文件中，找到以 **GRUB_CMDLINE_LINUX** 开头的行，如下所示：

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

3. 更改所需内核命令行参数的值。然后，保存文件并退出编辑器。
4. 使用编辑 **的默认** 文件重新生成 **GRUB2** 配置。如果您的系统使用 BIOS 固件，请执行以下命令：

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

   在带有 UEFI 固件的系统中，执行以下命令：

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

完成上述步骤后，会重新配置引导装载程序，且您在其配置文件中指定的内核命令行参数会在重启后应用。

在 zipl 中为 IBM Z 硬件更改内核命令行参数

1. 以 **root** 用户身份使用文本编辑器（如 **vim** 或 **Gedit** ）打开 **/etc/zipl.conf** 配置文件。
2. 在这个文件中，找到 **parameters=** 部分，再编辑 requiremed 参数，或者添加它（如果不存在）。然后，保存文件并退出编辑器。
3. 重新生成 **zipl** 配置：

   ```
   # zipl
   ```

注意

仅在不附加选项的情况下执行 **zipl** 命令会使用默认值。有关可用选项的详情，请查看 **zipl(8)手册**页。

完成上述步骤后，会重新配置引导装载程序，且您在其配置文件中指定的内核命令行参数会在重启后应用。

3.1.2. 可以控制哪些内核命令行参数

有关完整的内核命令行参数列表，请参阅 https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt。

3.1.2.1. 硬件特定内核命令行参数

pci=option[,option...]

指定 PCI 硬件子系统的行为

设置	效果
earlydump	[X86] 在内核更改前转储 PCI 配置空间
off	[X86] 不要对 PCI 总线进行探测
noaer	[PCIE] 如果启用了 PCIEAER 内核参数，可以使用这个内核引导选项禁用使用 PCIE 高级错误报告。
noacpi	[X86] 请勿将高级配置和电源接口(ACPI)用于中断请求(IRQ)路由或 PCI 扫描。
bfsort	将 PCI 设备按照广度优先的顺序排序.这种排序是为了获得与较旧(4.4)内核兼容的设备顺序。
nobfsort	不要将 PCI 设备按照广度优先的顺序排序。

其他 PCI 选项记录在 **kernel-doc-<version>.noarch** 软件包中的磁盘文档中。其中 '<version>' 需要替换为对应的内核版本。

acpi=option

指定高级配置和电源接口的行为

设置	效果
acpi=off	禁用 ACPI
acpi=ht	使用 ACPI 引导表解析，但不要启用 ACPI 解释器这禁用了 Hyper Threading **不需要的任何** ACPI 功能。
acpi=force	要求启用 ACPI 子系统
acpi=strict	使 ACPI 层无法接受不完全合规 ACPI 规范的平台。
acpi_sci=<value>	设置 ACPI SCI 中断，其中 <value> 是 edge,level,high,low 之一。

设置	效果
acpi=noirq	不要将 ACPI 用于 IRQ 路由
acpi=nocmcff	先禁用固件(FF)模式以更正错误。这会禁用解析 HEST CMC 错误源来检查固件是否设置了 FF 标志。这可能会导致重复的错误报告。

第 4 章 内核特性

本章解释了内核功能的目的和使用，它们启用了许多用户空间工具，还包含用于进一步调查这些工具的资源。

4.1. CONTROL GROUPS

4.1.1. 什么是控制组？

注意

Control Group Namespaces 在 Red Hat Enterprise Linux 7.5 中是一个技术预览功能

Linux 控制组(cgroups)启用对系统硬件使用的限制，确保在 **cgroup** 内运行的单个进程仅利用 **cgroups** 配置中允许的最大数量。

控制组限制由命名空间启用的资源的使用量。例如，网络命名空间允许进程访问特定的网卡，cgroup 确保进程不超过该卡的 50% 使用，从而确保带宽可用于其他进程。

控制组命名空间通过 /proc/self/ns/cgroup 接口提供单个 cgroup 的虚拟化视图。

目的是防止特权数据从全局命名空间中泄漏到 cgroup 并启用其他功能，如容器迁移。

由于容器与单个 cgroup 关联更为简单，容器具有更加一致的 cgroup 视图，它还允许容器内的任务具有它所属的 cgroup 的虚拟化视图。

4.1.2. 什么是命名空间？

命名空间是一种内核功能，允许对隔离的系统资源进行虚拟查看。通过将进程与系统资源隔离，您可以指定和控制进程能够与之交互的内容。命名空间是控制组的基本部分。

4.1.3. 支持的命名空间

Red Hat Enterprise Linux 7.5 及更新的版本支持以下命名空间

- Mount
  - 挂载命名空间隔离文件系统挂载点，使每个进程在 Wich 内都有一个不同的文件系统空间。
- UTS
  - 主机名和 NIS 域名
- IPC
  - 系统 V IPC, POSIX 消息队列
- PID
  - 进程 ID
- Network
  - 网络设备、堆栈、端口等.

- 用户
  - 用户和组群 ID
- 控制组群
  - 隔离 cgroups

注意

控制组的使用情况记录在资源管理指南中

4.2. 内核源检查器

Linux 内核模块源检查器(ksc)是一种用于检查给定内核模块中非白名单符号的工具。红帽合作伙伴还可以使用此工具请求审查白名单包含的符号，方法是在红帽 bugzilla 数据库中修复一个错误。

4.2.1. 使用

工具接受具有"-k"选项的模块路径

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

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

输出保存在 **$HOME/ksc-result.txt** 中。如果请求查看白名单加法符号，则必须将每个非白名单符号的用法描述添加到 **ksc-result.txt** 文件中。然后，可以通过使用 "-p" 选项运行 **ksc** 提交请求错误。

注意

KSC 目前不支持 **xz** 压缩。**ksc** 工具无法处理 **xz** 压缩方法，并报告以下错误：

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

在解决此限制之前，系统管理员需要先使用 **xz** 压缩手动解压缩任何第三方模块，然后才能运行 **ksc** 工具。

4.3. 直接访问文件(DAX)

直接访问名为"文件系统 dax"或"fs dax"的文件，可使应用程序在支持 dax 的存储设备上读取和写入数据，而无需使用页面缓存来缓冲对设备的访问。

使用 'ext4' 或 'xfs' 文件系统时可使用此功能，并通过使用 **-o dax** 挂载文件系统或向 **/etc/fstab** 中挂载条目的 options 部分添加 **dax** 来启用。

如需更多信息，包括代码示例，可在 **kernel-doc** 软件包中找到，并存储在 **/usr/share/doc/kernel-doc-<version>/Documentation/filesystems/dax.txt**，其中 '<version>' 是对应的内核版本号。

4.4. 用户空间的内存保护密钥（也称为 PKU 或 PKEYS）

内存保护密钥提供了一种机制，可以强制实施基于页面的保护域，但不需要在应用更改保护域时修改页表。它的工作原理是将每个页表条目中的 4 位忽略到一个"保护键"，从而给出 16 个可能的键。

内存保护密钥是某些 Intel CPU 芯片组的硬件功能。要确定您的处理器是否支持此功能，请检查 **/proc/cpuinfo** 中是否存在 **pku**

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

为了支持此功能，CPU 提供了一个新的用户可访问寄存器(PKRU)，每个密钥有两个独立位(Access Disable and Write Disable)。有两个新指令（RDPKRU 和 WRPKRU）可用于读取和写入新寄存器。

更多文档（包括编程示例）可在由 **kernel -doc 软件包提供的 /usr/share/doc/kernel-doc-*/Documentation/x86/protection- keys.txt** 中找到。

4.5. 内核请求空间布局随机化

内核 Adress Space Layout Randomization(KASLR)由两个部分组成，它们共同工作来增强 Linux 内核的安全性：

- 内核文本 KASLR
- 内存管理 KASLR

内核文本本身的物理地址和虚拟地址分别随机化为不同的位置。内核的物理地址可以为 64TB 以下的任意位置，而内核的虚拟地址在 [0xffffffff80000000、0xffffc0000000]（1GB 空间）之间受到限制。

内存管理 KASLR 有三个部分，其起始地址在特定区域随机化。因此，如果此代码依赖于知道感兴趣的符号位于内核地址空间中的什么位置，KASLR 可以防止将内核执行并重定向到恶意代码。

内存管理 KASLR 部分包括：

- 直接映射部分
- vmalloc 部分
- vmemmap 部分

KASLR 代码现在编译到 Linux 内核中，它默认是启用的。要明确禁用它，请在内核命令行中添加 **nokaslr** 内核选项。

4.6. 高级错误报告(AER)

4.6.1. 什么是 AER

高级错误报告 (**AER**)是一个内核功能，为 **Peripheral Component Interconnect Express** (**PCIe**)设备提供增强的错误报告功能。**AER** 内核驱动程序附加支持 **PCIe AER** 功能的 root 端口，以便：

- 在发生错误时收集全面的错误信息
- 向用户报告错误
- 执行错误恢复操作

例 4.1. AER 输出示例

```
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
```

当 **AER** 捕获错误时，它会向控制台发送错误消息。如果错误可被修复，则控制台输出会发出警告。

4.6.2. 收集并显示 AER 信息

要收集并显示 AER 信息，请使用 **rasdaemon** 程序。

流程

1. 安装 **rasdaemon** 软件包。

```
~]# yum install rasdaemon
```

2. 启用并启动 **rasdaemon** 服务。

```
~]# systemctl enable --now rasdaemon
```

3. 运行 **ras-mc-ctl** 命令，该命令显示记录错误的摘要（ **--summary** 选项），或者显示错误数据库中存储的错误（ **--errors** 选项）。

```
~]# ras-mc-ctl --summary
~]# ras-mc-ctl --errors
```

其它资源

- 有关 **rasdaemon** 服务的更多信息，请参阅 **rasdaemon(8)** 手册页。
- 有关 **ras-mc-ctl** 服务的更多信息，请参阅 **ras-mc-ctl(8)** man page。

第 5 章 手动升级内核

Red Hat Enterprise Linux 内核由 Red Hat Enterprise Linux 内核团队自定义构建，以确保其与支持的硬件的完整性和兼容性。在红帽发布内核前，必须先通过一组严格的质量保证测试。

Red Hat Enterprise Linux 内核以 RPM 格式打包，以便使用 **Yum** 或 **PackageKit 软件包** 管理器轻松升级和验证。**PackageKit** 自动查询 Red Hat Content Delivery Network 服务器，并通过包括内核软件包在内的可用更新通知您软件包。

因此，本章 仅对 需要使用 **rpm** 命令而不是 **yum** 手动更新内核软件包的用户有用。

警告

尽可能使用 **Yum** 或 **PackageKit** 软件包管理器来安装新的内核，因为它们始终 **安装新的** 内核，而不是替换当前的内核，这可能会导致系统无法引导。

警告

红帽不支持自定义内核。但是，可从解决方案文章中获得指导。

有关使用 **yum** 安装内核软件包的详情，请参考《 系统管理员指南》 中的相关章节。

有关 Red Hat Content Delivery Network 的详情，请查看《系统管理员指南》中的相关章节。

5.1. 内核软件包概述

Red Hat Enterprise Linux 包含以下内核软件包：

- **内核** - 包含单核、多核和多处理器系统的内核.
- **kernel-debug** - 包含内核诊断启用大量调试选项的内核，但牺牲了性能降低。
- **kernel-devel** - 包含内核标头，并且使文件足以根据 **内核** 软件包构建模块。
- **kernel-debug-devel** - 包含内核的开发版本，并为内核诊断启用大量调试选项，但牺牲了性能降低。
- **kernel-doc** - 内核源的文档文件.这些文件中记录了 Linux 内核的各个部分和设备驱动程序。安装此软件包提供对可在加载时传输到 Linux 内核模块的选项的引用。
  默认情况下，这些文件放置在 **/usr/share/doc/kernel-doc-*kernel_version*/** 目录中。
- **kernel-headers** - 包含指定 Linux 内核与用户空间库和程序之间的接口的 C 标头文件。头文件定义了构建大多数标准程序所需的常量结构和常量。
- **Linux-firmware** - 包含运行各种设备所需的所有固件文件。

- **perf** - 此软件包包含 **perf** 工具，可启用对 Linux 内核的性能监控。
- **kernel-abi-whitelists** - 包含与 Red Hat Enterprise Linux 内核 ABI 相关的信息，包括外部 Linux 内核模块所需的内核符号列表和 **yum** 插件以协助执行。
- **kernel-tools** - 包含操作 Linux 内核和支持文档的工具.

5.2. 准备升级

在升级内核前，建议您采取一些步骤。

首先，确保在出现问题时系统存在工作引导介质。如果引导装载程序没有正确配置来引导新内核，您可以使用此介质引导至 Red Hat Enterprise Linux

USB 介质通常采用闪存设备的形式，有时称为 *pen drive*、*thumb 磁盘或 密钥，或者作为外部连接的硬盘设备。这种类型的几乎所有介质都被格式化为 **VFAT** 文件系统。您可以在格式化为 **ext2**、ext**3**、ext**4** 或 **VFAT** 的介质上创建可引导 USB 介质。*

您可以将分发镜像文件或最小引导介质镜像文件传输到 USB 介质中。确保设备中有足够的可用空间。发行 DVD 镜像大约需要 4 GB，发行 CD 镜像大约需要 700 MB，最小引导介质镜像大约需要 10 MB。

*您必须有 Red Hat Enterprise Linux 安装 DVD 中的 **boot.iso** 文件的副本，或者安装 CD-ROM #1，您需要一个使用 **VFAT 文件系统** 和大约 16 MB 可用空间格式化的 USB 存储设备。*

有关使用 USB 存储设备的更多信息，请参阅如何格式化 USB 密钥以及如何在 非图形环境解决方案文章中手动挂载 USB 闪存驱动器。

*以下步骤不会影响 USB 存储设备上的现有文件，除非它们的路径名称与您复制到其中的文件相同。要创建 USB 引导介质，以 **root** 用户身份运行以下命令：*

1. *如果系统上尚未安装 **syslinux** 软件包，请安装该软件包。为此，请以 root 身份运行 **yum install syslinux** 命令。*
2. *在 USB 存储设备中安装 **SYSLINUX** 引导装载程序：*

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

   ... 其中 sdX 是设备名称。
3. *为 **boot.iso** 和 USB 存储设备创建挂载点：*

   ```
   # mkdir /mnt/isoboot /mnt/diskboot
   ```
4. *挂载 **boot.iso**:*

   ```
   # mount -o loop boot.iso /mnt/isoboot
   ```
5. *挂载 USB 存储设备：*

   ```
   # mount /dev/sdX1 /mnt/diskboot
   ```
6. *将 **ISOLINUX** 文件从 **boot.iso** 复制到 USB 存储设备中：*

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

7. 使用 **boot .iso 中的 isolinux. cfg** 文件作为 USB 设备的 **syslinux.cfg** 文件：

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

8. 卸载 **boot.iso** 和 USB 存储设备：

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

9. 使用启动介质重新启动计算机，并在继续之前验证您可以使用它启动。

或者，在具有软盘驱动器的系统上，您可以通过安装 **mkbootdisk** 软件包并以 **root** 身份运行 **mkbootdisk** 命令来创建引导磁盘。有关使用信息，请参阅安装软件包后的 **man mkbootdisk** man page。

要确定安装了哪些内核软件包，请在 shell 提示符下执行 **yum 列表安装"kernel-*"** 的命令。根据系统的架构，输出包含一些或全部以下软件包，版本号可能会有所不同：

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

从输出中，确定内核升级需要下载哪些软件包。对于单个处理器系统，唯一需要的包是 **内核** 软件包。有关不同软件包的描述，请查看 第 5.1 节 "内核软件包概述"。

5.3. 下载升级的内核

可以通过几种方法确定系统是否有更新的内核。

- 安全勘误 - 请参阅客户门户，了解有关安全勘误的信息，包括修复安全问题的内核升级。
- Red Hat Content Delivery Network - 对于订阅 Red Hat Content Delivery Network 的系统，**yum** 软件包管理器可以下载最新的内核并在系统上升级内核。在需要时，**D** racut 实用程序会创建一个初始 RAM 文件系统镜像，并将引导加载程序配置为引导新内核。有关从 Red Hat Content Delivery Network 安装软件包的更多信息，请参阅《系统管理员指南》的相关部分。有关将系统订阅到红帽内容交付网络的更多信息，请参阅《系统管理员指南》的相关章节。

如果使用 **yum** 从 Red Hat Network 下载并安装更新的内核，请只按照 第 5.5 节 "验证初始 RAM 文件系统镜像" 和 第 5.6 节 "验证引导装载程序" 中的说明 **操作，不要将** 内核更改为默认引导。Red Hat Network 自动将默认内核更改为最新版本。要手动安装内核，请继续 第 5.4 节 "执行升级"。

5.4. 执行升级

在获取了所有必要的软件包后，开始升级现有内核。

重要

强烈建议您在新内核出现问题时保留旧内核。

在 shell 提示符下，更改到包含内核 RPM 软件包的目录。使用 **-i** 参数和 **rpm** 命令，以保留旧内核。**不要** 使用 **-U** 选项，因为它会覆盖当前安装的内核，从而造成启动加载器问题。例如：

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

下一步是验证初始 RAM 文件系统映像是否已创建。详情请查看 第 5.5 节 "验证初始 RAM 文件系统镜像"。

5.5. 验证初始 RAM 文件系统镜像

初始 RAM 文件系统映像的作用是预加载块设备模块，如 IDE、SCSI 或 RAID，以便可以访问和挂载这些模块通常所在的根文件系统。在红帽企业 Linux 7 系统上，每当使用 **Yum**、**PackageKit** 或 **RPM** 软件包管理器安装新内核时，安装脚本始终会调用 **Dracut** 实用程序来创建 initramfs （初始 RAM 文件系统映像）。

如果您通过修改 **/etc/sysctl.conf 文件或其他 sysctl** 配置文件更改内核属性，如果在引导过程早期使用了更改的设置，则可能需要运行 **dracut -f** 命令来重建初始 RAM 文件系统镜像。例如，如果您已对联网进行了更改，并且从网络连接存储引导。

在 IBM eServer System i 以外的所有构架中（请参阅 "在 IBM eServer System i 上验证初始 RAM 文件系统镜像和内核"一节），您可以通过运行 **dracut** 命令创建 **initramfs**。但是，您通常不需要手动创建 **initramfs** ：如果从红帽分发的 RPM 软件包安装或升级了内核及其相关软件包，则会自动执行这一步。

您可以按照以下步骤验证与当前内核版本对应的 **initramfs** 是否存在，并在 **grub.cfg** 配置文件中正确指定：

验证初始 RAM 文件系统镜像

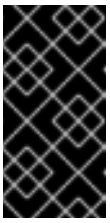
1. 以 **root** 身份列出 **/boot** 目录中的内容，并使用最新（最新的）版本号查找内核(**vmlinuz -kernel_version)和 initramfs- kernel_version** ：

   例 5.1. 确保内核和 initramfs 版本匹配

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

   例 5.1 "确保内核和 initramfs 版本匹配" 显示：

   - 我们已安装三个内核（或者 **/boot** 目录中存在三个内核文件）
   - 最新的内核是 **vmlinuz-3.10.0-78.el7.x86_64**，并且
   - 与内核版本 **initramfs -3.10.0-78.el7.x86_64kdump.img** 相匹配的 initramfs 文件也存在。



重要

在 **/boot** 目录中，您可能找到几个 **initramfs-kernel_versionkdump.img** 文件。这些是由 Kdump 机制创建的特殊文件用于内核调试目的，不用于引导系统，可以安全地忽略。有关 **kdump** 的详情请参考 Red Hat Enterprise Linux 7 内核崩溃指南。

2. 如果您的 **initramfs-kernel_version** 文件与 **/boot** 目录中的最新内核版本不匹配，或者在某些情况下，您可能需要使用 **Dracut** 实用程序生成 **initramfs** 文件。简单地在不使用选项的情况下以 **root** 身份调用 **dracut** 会导致它在 **/boot** 中为该目录中存在的最新内核生成 **initramfs** 文件：

```
# dracut
```

如果您希望 **dracut** 覆盖现有的 **initramfs**，则必须使用 **-f,--force** 选项（例如，如果您的 **initramfs** 已损坏）。Other **therdracut** 拒绝覆盖现有的 **initramfs** 文件：

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

您可以通过调用 **dracut initramfs _namekernel_version在当前目录中创建 initramfs** ：

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

如果您需要指定预加载的特定内核模块，请在 **add_dracut . conf 配置文件的括号中添加这些模块的名称（减去任何文件名后缀，　如.ko** ）。您可以使用 **lsinitrd initramfs _file 命令列出 dracut 创建的 initramfs 镜像** 文件的文件内容：

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

dracut-033-68.el7
========================================================================


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

有关选项和用法的更多信息，请参阅 **man dracut 和 man dracut.conf**。

3. 检查 **/boot/grub2/grub.cfg** 配置文件，以确保您正在引导的内核版本存在 **initramfs-kernel_version.img** 文件。例如：

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

如需更多信息，请参阅 第 5.6 节 "验证引导装载程序"。

在 IBM eServer System i 上验证初始 RAM 文件系统镜像和内核

在 IBM eServer System i 机器上，初始 RAM 文件系统和内核文件合并到一个文件中，该文件通过 **addRamDisk** 命令创建。如果从红帽分发的 RPM 软件包安装或升级了内核及其相关软件包，则会自动执行此步骤，因此不需要手动执行该步骤。要验证它是否已创建，请以 **root 用户身份** 运行以下命令，确保 **/boot/vmlinitrd-kernel_version** 文件已存在：

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

kernel_version 需要与刚才安装的内核版本匹配。

取消对初始 RAM 文件系统镜像所做的更改

例如，在某些情形中，如果您错误配置系统且不再引导，则需要按照以下步骤撤销对初始 RAM 文件系统镜像所做的更改：

撤销对初始 RAM 文件系统镜像的更改

1. 在 GRUB 菜单中选择救援内核重新引导系统。
2. 更改导致 **initramfs** 出现故障的错误设置。
3. 以 root 用户身份运行以下命令，使用正确的设置重新创建 **initramfs** ：

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

例如，当您在 **sysctl.conf** 文件中错误设置了 **vm.nr_hugepages** 时，上述步骤可能很有用。由于 **sysctl.conf** 文件包含在 **initramfs** 中，新的 **vm.nr_hugepages** 设置会应用到 **initramfs** 中，并导致重新构建 **initramfs**。但是，由于设置不正确，因此新的 **initramfs** 会中断，且新构建的内核不会引导，因此需要使用上述步骤更正设置。

列出初始 RAM 文件系统镜像的内容

要列出 **initramfs** 中包含的文件，以 root 用户身份运行以下命令：

```
# lsinitrd
```

要只列出 **/etc** 目录中的文件，请使用以下命令：

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

要输出存储在 **initramfs** 中为当前内核保存的特定文件的内容，请使用 **-f** 选项：

```
# lsinitrd -f filename
```

例如，要输出 **sysctl.conf** 的内容，请使用以下命令：

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

要指定内核版本，请使用 **--kver** 选项：

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

例如，要列出有关内核版本 3.10.0-327.10.1.el7.x86_64 的信息，请使用以下命令：

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

5.6. 验证引导装载程序

您可以使用 **yum** 命令或 **rpm** 命令安装内核。

当您使用 **rpm** 安装内核时，kernel 软件包会在引导装载程序配置文件中为新内核创建一个条目。

请注意，两个命令将新内核配置为引导为默认内核，只有在 **/etc/sysconfig/kernel** 配置文件中包含以下设置时：

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

The **DEFAULTKERNEL** 选项指定默认内核软件包类型。**UPDATEDEFAULT** 选项指定新内核软件包是否使新内核成为默认内核。

第 6 章 使用内核实时修补程序应用补丁

您可以使用 Red Hat Enterprise Linux 内核实时修补解决方案在不重启或者重启任何进程的情况下对运行的内核进行补丁。

使用这个解决方案，系统管理员需要：

- 可以在内核中立即应用重要的安全补丁。
- 不必等待长时间运行的任务完成、关闭或调度停机时间。
- 可以控制系统的正常运行时间，且不会牺牲安全性和稳定性。

请注意，并非所有关键或重要的 CVE 都使用内核实时补丁解决方案来解决。我们的目标是，在应用安全相关的补丁时，尽量减少重启的需要，但无法完全避免重启。有关实时补丁范围的详情，请参阅 客户门户网站解决方案文章。

警告

内核实时补丁和其它内核子组件之间存在一些不兼容。使用内核实时补丁前请仔细阅读 第 6.1 节 "kpatch 的限制" 部分。

注意

有关内核实时补丁更新支持节奏的详情，请参考：

- 内核实时补丁支持 Cadence 更新
- 内核实时补丁生命周期

6.1. KPATCH 的限制

- **kpatch** 功能不是一个通用内核升级机制。它可用于在无法立即重启系统时应用简单的安全性和程序错误修复更新。
- 不要在载入补丁期间或之后使用 **SystemTap** 或 **kprobe** 工具。在删除此类探测后，补丁可能无法生效。

6.2. 对第三方实时补丁的支持

kpatch 实用程序是红帽通过红帽软件仓库提供的 RPM 模块支持的唯一内核实时补丁程序。红帽不支持任何不是由红帽提供的实时补丁。

要获得第三方实时补丁支持，请联系提供修补程序的供应商。

对于任何使用了第三方补丁程序运行的系统，红帽保留请求用户使用由红帽提供并支持的软件重现问题的权利。如果无法做到这一点，我们需要在测试环境中部署类似的系统和工作负载，而无需应用实时补丁，以确认是否观察到了相同的行为。

有关第三方软件支持政策的更多信息，请参阅红帽全球支持服务如何处理第三方软件、驱动程序和/或未经认证的硬件/管理程序或虚拟机操作系统？

6.3. 获得内核实时补丁

内核实时补丁功能作为内核模块（**.ko** 文件）实施，该模块作为 RPM 软件包提供。

所有客户都可以访问内核实时补丁，这些补丁通过常用的通道提供。但是，在下一个次版本发布后，未订阅延长支持服务的客户将无法访问当前次要版本的新修补程序。例如，具有标准订阅的客户只能在 RHEL 8.3 发布前提供 RHEL 8.2 内核补丁。

6.4. 组件内核实时修补

内核实时补丁的组件如下：

内核补丁模块

- 内核实时补丁的交付机制。
- 为内核建补丁的内核模块。
- patch 模块包含内核所需修复的代码。
- patch 模块使用 **livepatch** 内核子系统注册，并提供要替换的原始功能的信息，并提供与替换功能对应的指针。内核补丁模块以 RPM 的形式提供。
- 命名规则为 **kpatch_<kernel version>_<kpatch version>_<kpatch release>**。名称的 "kernel version" 部分的 dots 和 dashes 会被 underscores 替换。

kpatch 工具

用于管理补丁模块的命令行工具。

kpatch 服务

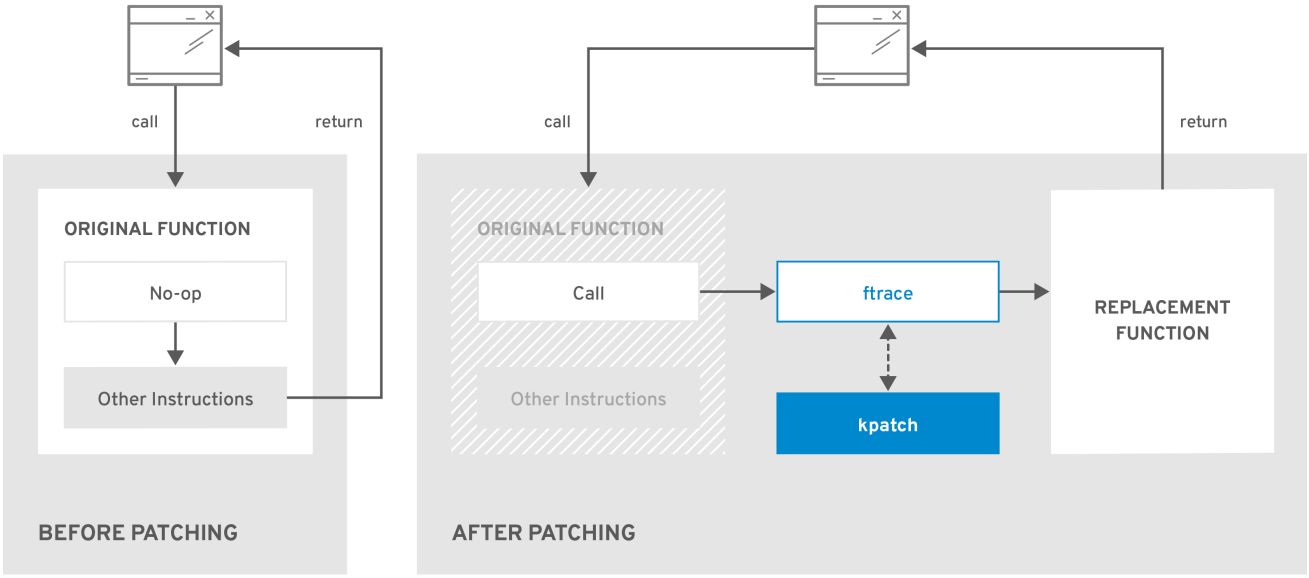
multiuser.target 所需的 **systemd** 服务。这个目标会在引导时载入内核补丁模块。

6.5. 内核实时补丁如何工作

kpatch 内核补丁解决方案使用 **livepatch** 内核子系统将旧功能重定向到新功能。当实时内核补丁应用到系统时，会出现以下情况：

1. 内核补丁模块复制到 **/var/lib/kpatch/** 目录中，并在下次引导时由 **systemd** 注册以重新应用到内核。
2. kpatch 模块被加载到正在运行的内核中，补丁的功能会注册到 **ftrace** 机制中，带有指向新代码内存中位置的指针。
3. 当内核访问补丁的功能时，它将由 **ftrace** 机制重定向，该机制绕过原始功能并将内核重定向到功能补丁版本。

图 6.1. 内核实时补丁如何工作



6.6. 启用内核实时补丁

内核补丁模块在 RPM 软件包中提供，具体取决于被修补的内核版本。每个 RPM 软件包将随着时间不断累积更新。

以下小节描述了如何确保您将来收到给定内核的所有实时补丁更新。

> **警告**
>
> *红帽不支持任何适用于红帽支持系统的第三方实时补丁。*

6.6.1. 订阅实时补丁流

这个步骤描述了安装特定的实时补丁软件包。通过这样做，您可以订阅给定内核的实时补丁流，并确保您收到以后为该内核的所有实时补丁更新。

> **警告**
>
> *因为实时补丁是累计的，所以您无法选择为一个特定的内核部署哪些单独的补丁。*

先决条件

- *根权限*

流程

1. 另外，还可检查您的内核版本：

```
# uname -r
3.10.0-1062.el7.x86_64
```

2. 搜索与内核版本对应的实时补丁软件包：

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

3. 安装 live patching 软件包：

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

以上命令只为特定内核安装并应用最新的实时补丁。

如果软件包的版本是 1-1 或更高版本，则 live patching 软件包包含一个补丁模块。在这种情况下，内核会在安装 live patching 软件包期间自动修补。

内核补丁模块也安装到 **/var/lib/kpatch/** 目录中，供 **systemd** 系统和服务管理器以后重启时载入。

注意

如果给定内核还没有可用的实时补丁，则会安装一个空的 live patching 软件包。空的 live patching 软件包会有一个 0-0 的 kpatch_version-kpatch_release，如 **kpatch-patch-3_10_0-1062-0-0.el7.x86_64.rpm**。空 RPM 安装会将系统订阅到以后为给定内核提供的所有实时补丁。

4. 另外，用来确认内核已应用了补丁程序：

```
# kpatch list
Loaded patch modules:
kpatch_3_10_0_1062_1_1 [enabled]

Installed patch modules:
kpatch_3_10_0_1062_1_1 (3.10.0-1062.el7.x86_64)
…
```

输出显示内核补丁模块已加载到内核中，现在它提供了来自 **kpatch-patch-3_10_0-1062-1-1.el7.x86_64.rpm** 软件包的最新修复。

其它资源

- 有关 **kpatch** 命令行工具的详情请参考 **kpatch(1)** 手册页。
- 有关 RHEL 7 中软件包的更多信息，请参阅系统管理员指南 中的相关章节。

6.7. 更新内核补丁模块

由于内核补丁模块是通过 RPM 软件包交付和应用，更新累积内核补丁模块就如同更新任何其他 RPM 软件包一样。

先决条件

- 根权限
- 系统订阅了实时补丁流，如 第 6.6.1 节 "订阅实时补丁流" 所述。

流程

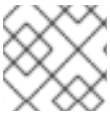
- 更新至当前内核的新累计版本：

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

以上命令会自动安装并应用所有当前运行的内核可用的更新。包括将来发布的所有在线补丁。

- 另外，更新所有安装的内核补丁模块：

```
# yum update "kpatch-patch*"
```



注意

当系统重启到同一内核时，**kpatch.service** 服务会再次对内核进行补丁。

其它资源

- 有关更新软件包的更多信息，请参阅《系统管理员指南》的 相关章节。

6.8. 禁用内核实时补丁

如果系统管理员遇到与 Red Hat Enterprise Linux 内核实时修补解决方案相关联的一些无法预期的负面影响，他们可以选择禁用该机制。以下小节描述了如何禁用实时补丁解决方案。

重要

目前，红帽不支持在不重启系统的情况下还原实时补丁。如有任何问题，请联系我们的支持团队。

6.8.1. 删除 live patching 软件包

下面的步骤描述了如何通过删除 live patching 软件包禁用Red Hat Enterprise Linux 内核实时修补解决方案。

先决条件

- 根权限
- 已安装 live patching 软件包。

流程

1. 选择 live patching 软件包：

```
# yum list installed | grep kpatch-patch
kpatch-patch-3_10_0-1062.x86_64        1-1.el7        @@commandline
…
```

上面的输出示例列出了您安装的实时补丁软件包。

2. 删除 live patching 软件包：

```
# yum remove kpatch-patch-3_10_0-1062.x86_64
```

删除实时补丁软件包后，内核将保持补丁，直到下次重启为止，但内核补丁模块会从磁盘中删除。下一次重启后，对应的内核将不再修补。

3. 重启您的系统。

4. 验证 live patching 软件包是否已删除：

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

如果软件包已被成功删除，命令不会显示任何输出。

5. （可选）验证内核实时补丁解决方案是否已禁用：

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

示例输出显示内核没有补丁，实时补丁解决方案没有激活，因为目前没有加载补丁模块。

其它资源

- 有关 **kpatch** 命令行工具的详情请参考 **kpatch(1)** 手册页。
- 有关使用软件包的更多信息，请参阅《系统管理员指南》的 相关章节。

6.8.2. 卸载内核补丁模块

下面的步骤描述了如何防止 Red Hat Enterprise Linux 内核实时修补解决方案，使其在以后的引导中应用内核补丁模块。

先决条件

- 根权限
- 已安装实时补丁软件包。
- 已安装并载入内核补丁模块。

流程

1. 选择内核补丁模块：

```
# kpatch list
Loaded patch modules:
kpatch_3_10_0_1062_1_1 [enabled]

Installed patch modules:
kpatch_3_10_0_1062_1_1 (3.10.0-1062.el7.x86_64)
…
```

2. 卸载所选内核补丁模块：

```
# kpatch uninstall kpatch_3_10_0_1062_1_1
uninstalling kpatch_3_10_0_1062_1_1 (3.10.0-1062.el7.x86_64)
```

- 请注意，卸载的内核补丁模块仍然被加载：

```
# kpatch list
Loaded patch modules:
kpatch_3_10_0_1062_1_1 [enabled]

Installed patch modules:
<NO_RESULT>
```

卸载所选模块后，内核将保持补丁，直到下次重启为止，但已从磁盘中删除内核补丁模块。

3. 重启您的系统。

4. （可选）验证内核补丁模块是否已卸载：

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

以上输出示例显示没有加载或已安装的内核补丁模块，因此没有修补内核，且内核实时补丁解决方案未激活。

其它资源

- 有关 **kpatch** 命令行工具的详情请参考 **kpatch(1)** 手册页。

6.8.3. 禁用 kpatch.service

下面的步骤描述了如何防止 Red Hat Enterprise Linux 内核实时修补解决方案在以后的引导中全局应用所有内核补丁模块。

先决条件

- 根权限
- 已安装实时补丁软件包。
- 已安装并载入内核补丁模块。

流程

1. 验证 **kpatch.service** 是否已启用：

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

2. 禁用 **kpatch.service** ：

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

- 请注意，应用的内核补丁模块仍然被载入：

```
# kpatch list
Loaded patch modules:
kpatch_3_10_0_1062_1_1 [enabled]

Installed patch modules:
kpatch_3_10_0_1062_1_1 (3.10.0-1062.el7.x86_64)
```

3. 重启您的系统。

4. （可选）验证 **kpatch.service** 的状态：

```
# 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)
```

示例输出测试 **kpatch.service** 已被禁用且没有在运行。因此，内核实时修补解决方案不活跃。

5. 确认内核补丁模块已被卸载：

```
# kpatch list
Loaded patch modules:

Installed patch modules:
kpatch_3_10_0_1062_1_1 (3.10.0-1062.el7.x86_64)
```

上面的示例输出显示内核补丁模块仍处于安装状态，但没有修补内核。

其它资源

- 有关 **kpatch** 命令行工具的详情请参考 **kpatch(1)** 手册页。
- 有关 **systemd** 系统和服务管理器、单元配置文件、位置以及完整的 **systemd** 单元类型列表的更多信息，请参阅 系统管理员指南中 的相关部分。

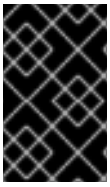
第 7 章 内核崩溃转储指南

7.1. KDUMP 简介

7.1.1. 关于 kdump 和 kexec

kdump 是提供崩溃转储机制的服务。该服务允许您保存系统内存内容进行分析。

kdump 使用 **kexec** 系统调用在不重启的情况下引导至第二个内核（捕获内核），然后捕获崩溃内核的内存（崩溃转储 或 vmcore）的内容，并将其保存到文件中。这个第二个内核位于系统内存保留的一部分。



重要

内核崩溃转储可能是失败时唯一可用的信息，将这些数据置于业务关键型环境中的重要性是不容忽视的。红帽建议系统管理员在正常内核更新周期内定期更新和测试 **kexec-tools**。这在部署了新内核功能时尤为重要。

注意

HP Watchdog 定时器(hpwdt)驱动程序在作为 RHEV 系统管理程序运行的 HP 系统中预先加载，因此这些系统可以使用 NMI 监视器。更新了 kexec-tools 软件包，从 kexec-tools-2.0.15-33.el7.x86_64 开始预加载 hpwdt 驱动程序。

如果 knx2x 和 bmx2fc 没有在 kdump 内核中列入黑名单，则第二个内核会导致 panic，且转储不会被捕获。

7.1.2. 内存要求

为了使 kdump 能够捕获内核崩溃转储并保存以便进一步分析，系统内存部分必须永久保留给捕获内核。保留时，系统内存的这一部分不适用于主内核。

内存要求因某些系统参数而异。主要因素之一就是系统的硬件构架。要找出机器架构（如 **x86_64**）的确切名称并将其输出到标准输出，在 shell 提示符下键入以下命令：

```
uname -m
```

影响要保留的内存量的另一个因素是安装的系统内存总量。例如，在 x86_64 体系结构中，每 4 KB RAM 的保留内存大小为 160 MB + 2 位。在安装了 1 TB 物理内存的系统上，这意味着 224 MB(160 MB + 64 MB)。如需基于系统架构和物理内存量的 kdump 的完整内存要求列表，请参阅 第 7.8.1 节 “kdump 的内存要求”。

在很多系统中，kdump 可以估算所需内存量并自动保留。默认情况下，此行为是启用的，但仅适用于内存总量超过特定数量的系统，这些内存因系统架构而异。有关根据系统架构自动内存保留的最低要求列表，请参阅 第 7.8.2 节 “自动内存保留的最小阈值”。

如果系统自动分配所需的最小内存量小于自动分配所需的最小内存量，或者您的用例需要不同的值，则可以手动配置保留的内存量。有关如何在命令行中执行此操作的详情请参考 第 7.2.2.1 节 “配置内存用量”。有关如何在图形用户界面中配置保留内存量的详情请参考 第 7.2.3.1 节 “配置内存用量”。

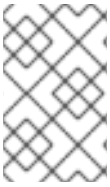
重要

强烈建议您在设置 kdump 服务后测试配置，即使使用自动内存保留。有关如何测试配置的步骤，请参阅 第 7.4 节 “测试 kdump 配置”。

7.2. 安装和配置 KDUMP

7.2.1. 安装 kdump

*许多情况下，在新的 Red Hat Enterprise Linux 7 安装中默认安装并激活 **kdump** 服务。在使用图形或文本界面执行交互式安装时，**Anaconda** 安装程序为 kdump 配置提供了一个屏幕。安装程序屏幕名为 **Kdump**，可在主 **安装概述** 屏幕中访问，且只允许有限的配置 - 您只能选择是否启用了 kdump 以及保留多少内存。有关 kdump 的内存要求的信息请参考 第 7.8.1 节 "kdump 的内存要求"。安装程序中的 Kdump 配置屏幕记录在 Red Hat Enterprise Linux 7 安装指南中 。*



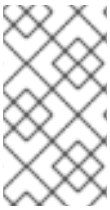
注意

*在之前的 Red Hat Enterprise Linux 版本中，kdump 配置 **在安装** 完成后自动执行的 **Firstboot** 实用程序中可用，该系统第一次重启。从 Red Hat Enterprise Linux 7.1 开始，kdump 配置已移至安装程序。*

*有些安装选项（如自定义 Kickstart 安装）在默认情况下不必安装或启用 kdump。如果您的系统中是这种情况，且您想要额外安装 kdump，请在 shell 提示符后以 **root** 用户身份执行以下命令：*

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

*假设您的系统有一个有效的订阅或包含用于您的系统构架的 **kexec-tools** 软件包的自定义软件仓库，以上命令可保证 kdump 的安装和其他所有必要的软件包的安全。*



注意

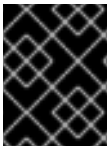
*如果您不知道系统中是否安装了 kdump，您可以使用 **rpm** 进行检查：*

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

*此外，如果使用上述命令，也可以使用图形配置工具，但默认情况下不会安装。要安装这个工具，如 第 7.2.3 节 "在图形用户界面中配置 kdump" 所述，以 **root 用户身份** 运行以下命令：*

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

*有关如何使用 **Yum** 软件包管理器在 Red Hat Enterprise Linux 7 中安装新软件包的更多信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南。*



重要

*从 Red Hat Enterprise Linux 7.4 开始，kdump 支持 **Intel IOMMU** 驱动程序 。从 7.3 或更早版本运行内核时，建议禁用 **Intel IOMMU** 支持。*

7.2.2. 在命令行中配置 kdump

7.2.2.1. 配置内存用量

在系统引导过程中，为 kdump 内核保留的内存始终保留，这意味着在系统的引导装载程序配置中指定了内存量。

要指定 kdump 内核保留的内存，将 **crashkernel=** 选项设置为所需的值。例如：要保留 128 MB 内存，请使用：

```
crashkernel=128M
```

有关如何使用 **GRUB2** 引导装载程序在 AMD64 和 Intel 64 系统以及 IBM Power Systems 服务器以及使用 **zipl** 在 IBM Z 中更改 **crashkernel=** 选项的详情请参考 第 3.1.1 节 "设置内核命令行参数"。

crashkernel= 选项可以通过多种方式定义。**自动值启用根据系统中的内存总量自动配置** 保留内存。第 7.8.1 节 "kdump 的内存要求" 较大的内存系统（达到操作系统的既定限制）根据 **crashkernel=auto** 选项的架构进行计算。

将 **auto** 值替换为特定内存量来更改此行为。

crashkernel= 选项对于较小的内存系统特别有用。例如：要保留 128 MB 内存，请使用：

```
crashkernel=128M
```

您还可以根据安装的内存总量，将保留内存量设置为变量。变量内存保留的语法为 **crashkernel=<range1>: <size1>,<range2>:<size2>**。例如：

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

如果系统内存总量为 512 MB 或高于 2 GB，则上述示例保留 64 MB 内存。如果内存总量超过 2 GB，则为 kdump 保留 128 MB。

有些系统需要使用特定的固定偏移保留内存。如果设置了偏移，则保留内存从此偏移开始。要偏移保留的内存，请使用以下语法：

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

上面的例子意味着 kdump 从 16 MB 开始保留 128 MB 内存（物理地址 0x01000000）。如果偏移参数设为 0 或完全省略，kdump 会自动偏移保留内存。如上所述，在设置变量内存保留时也可以使用此语法；在本例中，偏移始终被最后指定（例如 **crashkernel=512M-2G:64M,2G-:128M@16M**）。

7.2.2.2. 配置 kdump 目标

当捕获内核崩溃时，核心转储可以作为文件存储在本地文件系统中，直接写入设备，或使用 **NFS**（网络文件系统）或 **SSH** (Secure Shell) 协议通过网络发送。目前只能设置这些选项中的一个。默认选项是将 **vmcore** 文件存储在本地文件系统的 **/var/crash** 目录中。

- 要将 **vmcore** 文件存储在本地文件系统的 **/var/crash/** 目录中，编辑 **/etc/kdump.conf** 文件并指定路径：

  ```
  path /var/crash
  ```

  选项 **路径 /var/crash** 代表 **kdump** 保存 **vmcore** 文件的文件系统路径。

注意

- 当您在 **/etc/kdump.conf** 文件中指定转储目标时，**路径相对于** 指定的转储目标。
- 当您没有在 **/etc/kdump.conf** 文件中指定转储目标时，该路径代表根目录 **的绝对路径**。

根据当前系统中挂载的内容，会自动执行转储目标和调整的转储路径。

例 7.1. kdump 目标配置

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

此处指定转储目标(**ext4 /dev/mapper/vg00-varcrashvol**)，因此 **path** 选项也会设置为 **/var/crash**，因此 **kdump** 会将 **vmcore** 文件保存在 **/var/crash /var/crash** 目录中。

要更改转储位置，以 **root** 用户身份打开文本编辑器中的 **/etc/kdump.conf** 配置文件并编辑选项，如下所述。

要更改保存内核转储的本地目录，请从 **#path /var/crash** 行的开头删除 hash 符号("#")，并使用所需的目录路径替换值。

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

重要

在 Red Hat Enterprise Linux 7 中，当 kdump systemd 服务启动 - 否则服务失败时，使用 **path** 指令定义为 **kdump** 目标的目录必须存在。此行为与早期版本的 Red Hat Enterprise Linux 不同，如果启动服务时不存在 目录，则会自动创建该目录。

另外，如果您要将文件写入不同的分区，请按照以 **#ext4** 开头的行之一执行相同的步骤。您可以在此处使用设备名称（**#ext4 /dev/vg/lv_kdump** 行）、文件系统标签（**#ext4 LABEL=/boot** 行）或 UUID（**#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** 行）。将文件系统类型以及设备名称、标签或者 UUID 更改为所需值。

注意

指定 UUID 值的正确语法是 **UUID="correct-uuid"** 和 **UUID=correct-uuid**。

例如：

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

重要

建议使用 **LABEL= 或 UUID=** 指定存储设备。无法保证 **/dev/sda3** 等磁盘设备名称在重启后保持一致。有关持久磁盘设备命名的详情，请查看 Red Hat Enterprise Linux 7 存储管理指南。

重要

在 s390x 硬件中转储到 DASD 时，在继续前在 **/etc/dasd.conf** 中正确指定转储设备非常重要。

要将转储直接写入设备，请从 **#raw /dev/vg/lv_kdump** 行的开头删除哈希符号("#")，并使用所需的设备名称替换该值。例如：

```
raw /dev/sdb1
```

要使用 **NFS** 协议将转储保存到远程机器中，请从 **#nfs my.server.com:/export/tmp** 行的开头删除 hash 符号("#")，并使用有效的主机名和目录路径替换该值。例如：

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

要使用 **SSH** 协议将转储保存到远程机器中，请从 **#ssh user@my.server.com** 行的开头删除 hash 符号("#")，并将该值替换为有效的用户名和密码。要在配置中包含 SSH 密钥，请从 **#sshkey /root/.ssh/kdump_id_rsa** 行的开头删除哈希符号，并将值改为您要转储的服务器上的密钥位置。例如：

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

有关如何配置 SSH 服务器和设置基于密钥的身份验证的详情，请参考 Red Hat Enterprise Linux 7 系统管理员指南。

有关当前支持的和不受支持的目标的完整列表，请参阅 表 7.3 “支持的 kdump 目标”。

7.2.2.3. 配置内核收集器

要减少 **vmcore** 转储文件的大小，**kdump 允许您** 指定一个外部应用程序（内核收集器）来压缩数据，并选择性地排除所有不相关的信息。目前，唯一完全支持的核心收集器是 **makedumpfile**。

要启用核心收集器，以 **root** 用户身份打开文本编辑器中的 **/etc/kdump.conf** 配置文件，请从 **#core_collector makedumpfile -l --message-level 1 -d 31** 行的开头删除 hash 符号("#")。

要启用转储文件压缩，请添加 **-l** 参数。例如：

```
core_collector makedumpfile -l
```

要从转储中删除某些页面，请添加 **-d value** 参数，其中 值是您要省略的页面值总和，如 表 7.4 “支持的过滤级别” 所述。例如：要删除零和可用页面，请使用：

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

有关可用选项的完整列表，请参阅 **makedumpfile(8)手册**页。

7.2.2.4. 配置默认操作

默认情况下，**kdump** 使用 **kexec** 系统调用在没有重启的情况下引导至第二个内核（捕获内核），然后捕获崩溃内核的内存（崩溃转储 或 vmcore）并将其保存到文件中。成功保存后，**kdump** 重启机器。

但是，当 **kdump** 无法在 第 7.2.2.2 节 “配置 kdump 目标” 中指定的目标位置创建内核转储时，**kdump** 会在不保存 vmcore 的情况下重启系统。要更改此行为，以 **root** 用户身份打开文本编辑器中的

/etc/kdump.conf 配置文件，从 #default shell 行的开头删除 hash 符号("#")，并将值替换为所需的操作，如 表 7.5 "支持的默认操作" 所述。

例如：

```
default reboot
```

7.2.2.5. 启用服务

要在引导时启动 kdump 守护进程，以 root 用户身份在 shell 提示符后输入以下内容：

```
systemctl enable kdump.service
```

这为 multi-user.target 启用服务。同样，按 systemctl disable kdump 可禁用 kdump。要在当前会话中启动该服务，以 root 用户身份运行以下命令：

```
systemctl start kdump.service
```

重要

在 Red Hat Enterprise Linux 7 中，kdump systemd 服务启动时，定义为 kdump 目标的目录必须存在 - 否则服务失败。此行为与早期版本的 Red Hat Enterprise Linux 不同，如果启动服务时不存在 目录，则会自动创建该目录。

有关 systemd 和配置服务的更多信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南 。

7.2.3. 在图形用户界面中配置 kdump

要启动 内核转储配置 实用程序，请 从面板选择Activities Other Kernel crash dumps，或者在 shell 提示符下键入 system-config-kdump。因此，会出现一个窗口，如 图 7.1 "基本设置" 所示。

工具允许您配置 kdump，并在引导时启用或禁用启动该服务。完成后，单击 Apply 以保存更改。除非您已通过身份验证，否则请输入超级用户密码。实用程序向您发出一个提醒，您需要重新引导系统才能应用您对配置所做的任何更改。

重要

在在强制模式中运行的 SELinux 的 IBM Z 或 PowerPC 系统中，启动内核转储配置程序前必须启用 kdumpgui_run_bootloader 布尔值。这个布尔值允许 system-config-kdump 在 bootloader_t SELinux 域中运行引导装载程序。要永久启用该布尔值，请以 root 用户身份运行以下命令：

```
# setsebool -P kdumpgui_run_bootloader 1
```

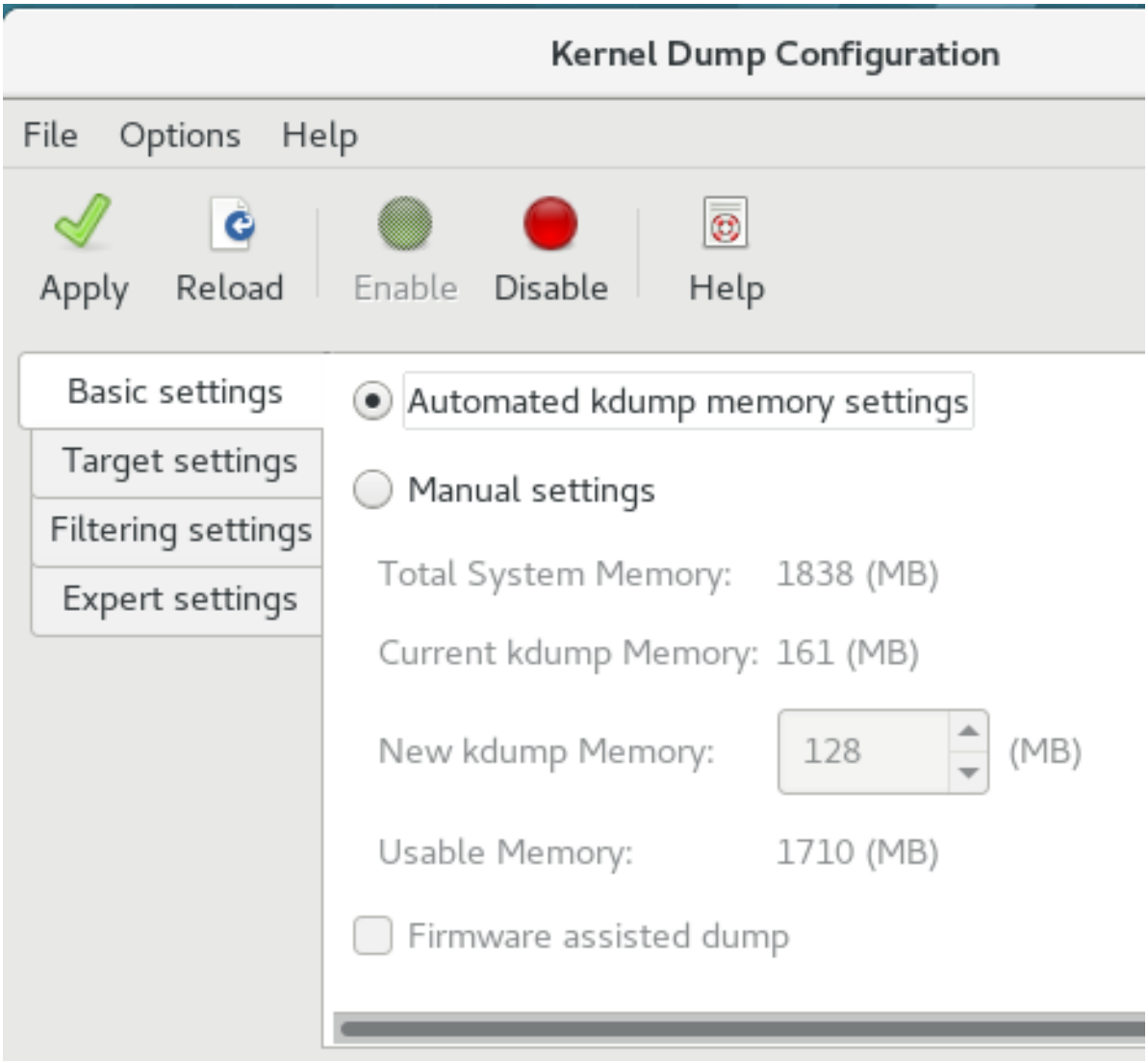
重要

在 s390x 硬件中转储到 DASD 时，在继续前在 /etc/dasd.conf 中正确指定转储设备非常重要。

7.2.3.1. 配置内存用量

Basic Settings 标签页可让您配置为 **kdump** 内核保留的内存量。为此，可选择" **手动设置** "单选按钮，然后单击 **新建 kdump Memory** 字段旁边的向上和向下箭头按钮，以增加或减少要保留的内存量。请注意，**Usable Memory** 字段会相应地更改，显示可供系统使用的剩余内存。有关 kdump 内存要求的详情，请查看 第 7.1.2 节 "内存要求"。

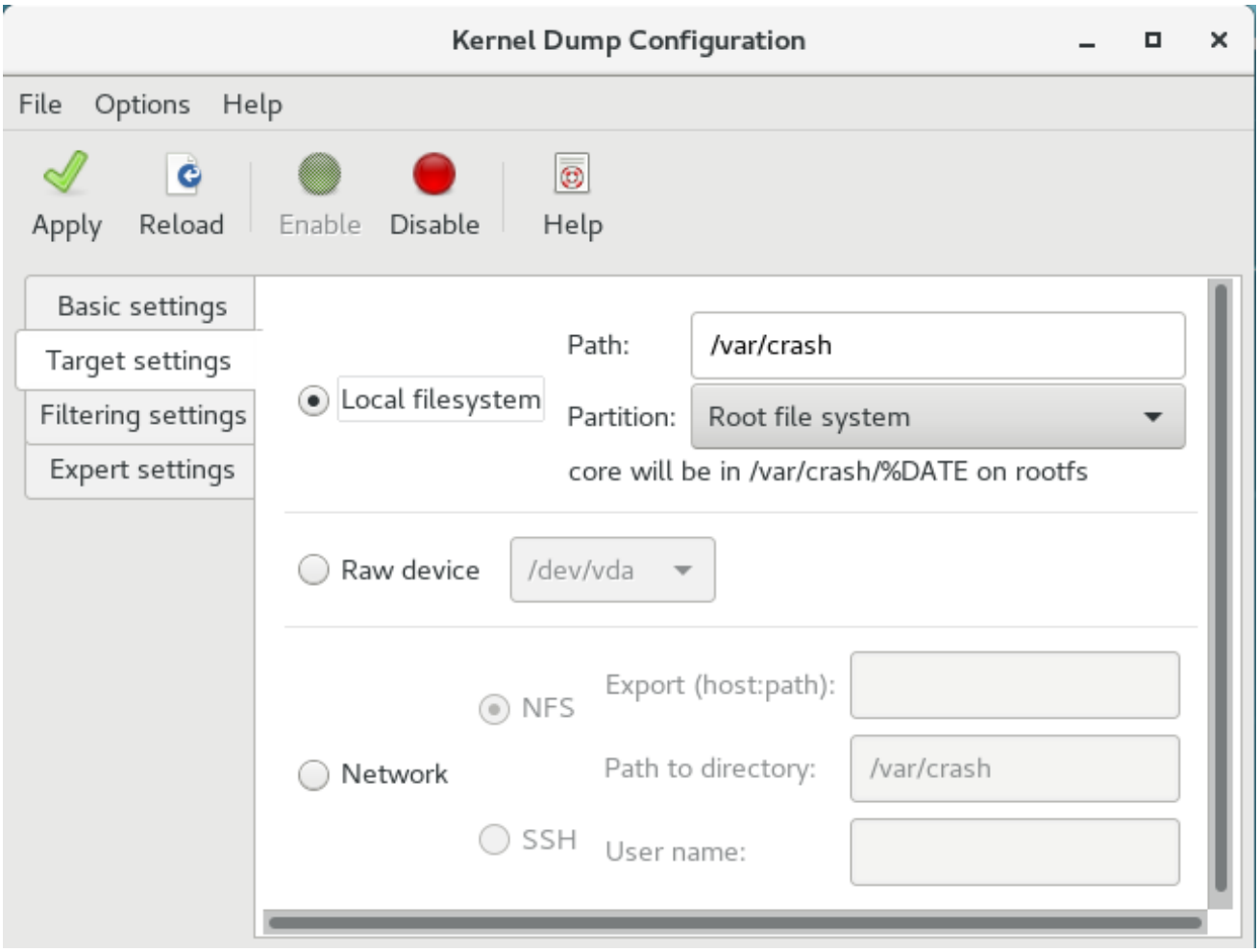
图 7.1. 基本设置



7.2.3.2. 配置 kdump 目标

Target Settings 选项卡允许您指定 **vmcore** 转储的目标位置。转储可以作为文件存储在本地文件系统中，直接写入设备，或者使用 **NFS** （网络文件系统）或 **SSH** (Secure Shell)协议通过网络发送。

图 7.2. 目标设置



*要将转储保存到本地文件系统，请选择 **本地文件系统** 单选按钮。另外，您还可以使用 **Path** 字段选择 **分区** 下拉列表和目标目录不同的分区来自定义设置。*

重要

*在 Red Hat Enterprise Linux 7 中，kdump systemd 服务启动时，定义为 **kdump** 目标的目录必须存在 - 否则服务失败。此行为与早期版本的 Red Hat Enterprise Linux 不同，如果启动服务时不存在 目录，则会自动创建该目录。*

*要将转储直接写入设备，请选择 **Raw 设备** 单选按钮，然后从其旁边的下拉菜单中选择所需的目标设备。*

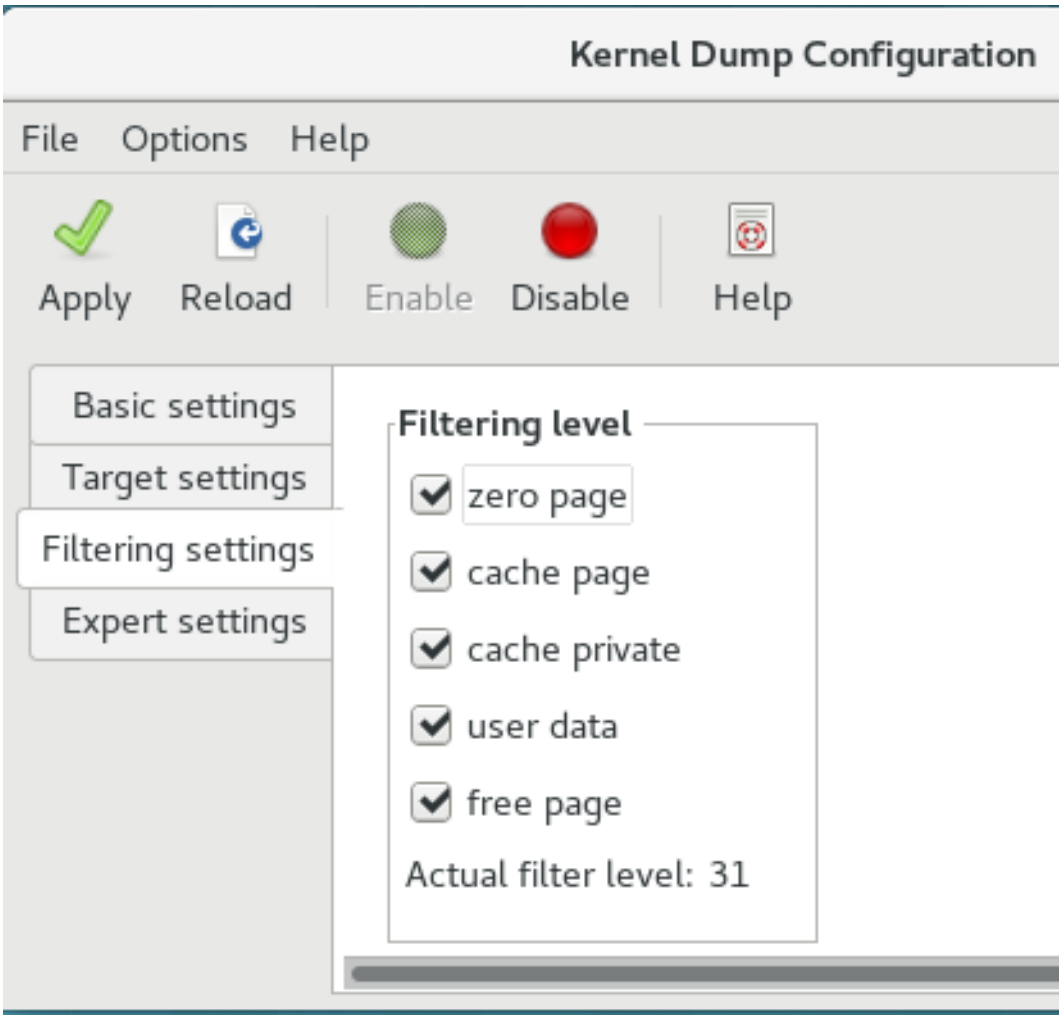
*要通过网络连接将转储发送到远程计算机，请选择 **网络** 单选按钮。要使用 **NFS** 协议，请选择 **NFS** 单选按钮，并填写 **服务器名称和 目录路径** 字段。要使用 **SSH** 协议，请选择 **SSH** 单选按钮，再将 **Server name**、**Path to directory** 和 **User name** 字段分别填入远程服务器地址、目标目录和有效用户名。*

有关如何配置 SSH 服务器和设置基于密钥的身份验证的详情，请参考 Red Hat Enterprise Linux 7 系统管理员指南。有关当前支持的目标的完整列表，请参阅 表 7.3 “支持的 kdump 目标”。

7.2.3.3. 配置内核收集器

***Filtering Settings** 选项卡允许您选择 **vmcore** 转储的过滤级别。*

图 7.3. 过滤设置



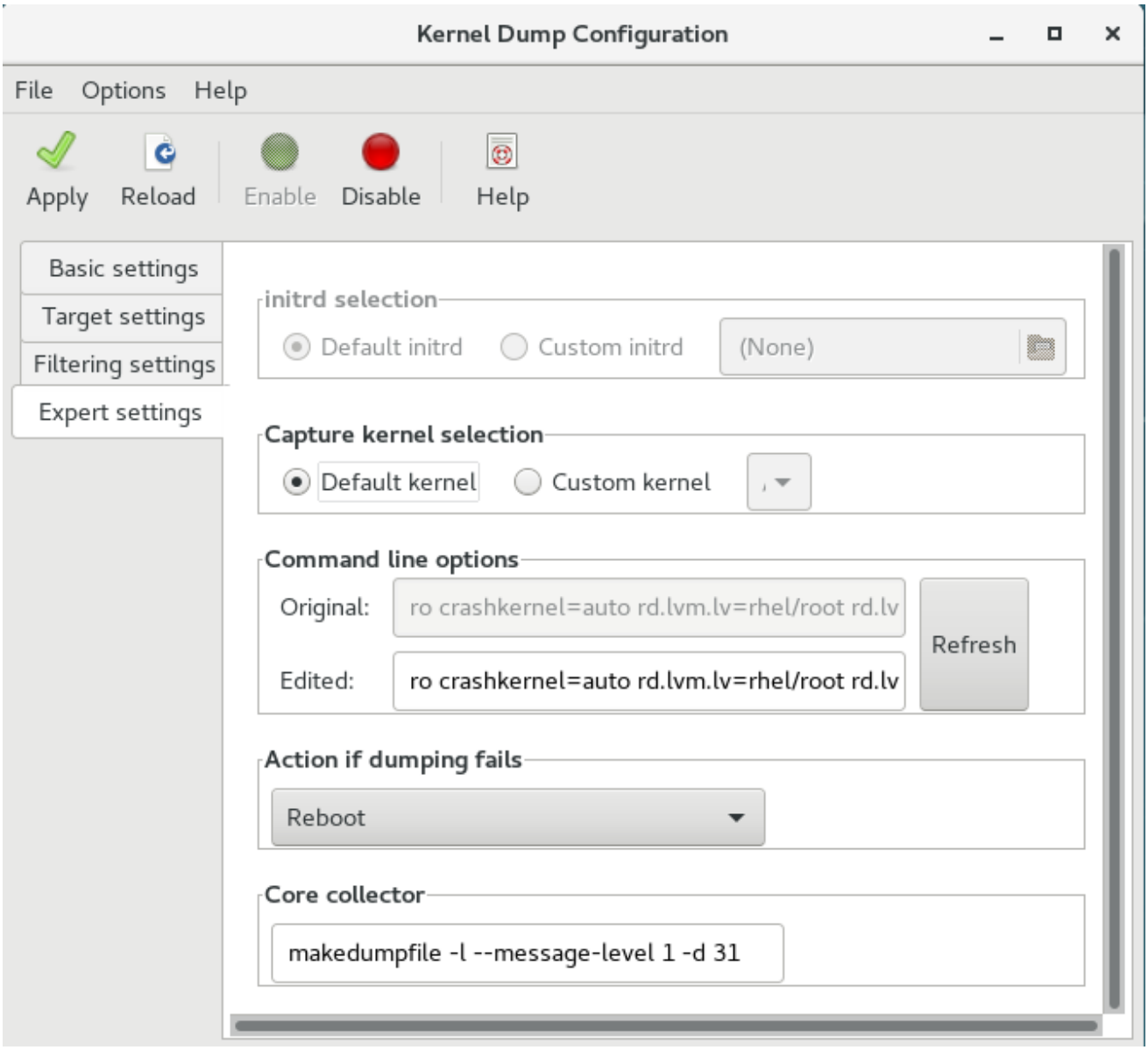
*要排除 **零页面**、**缓存页面**、**缓存页面**、 缓存、**用户数据** 或从转储中 **可用页面**，请选中相应标签旁边的复选框。*

7.2.3.4. 配置默认操作

*要在 **kdump** 无法创建内核转储时选择要执行的操作，**如果转储失败，请从操作** 中选择一个适当的选项。可用选项有：*

- *转储到 rootfs 并重启会尝试在本地保存内核，然后重新启动系统*
- *重启重启系统的默认操作*
- *启动 Shell 以显示具有相互活跃 shell 提示符的用户*
- *停止系统*
- *poweroff 关闭系统*

图 7.4. 过滤设置



*要自定义传递给 **makedumpfile** 内核收集器的选项，编辑 **Core 收集器** 文本字段；如需更多信息，请参阅 第 7.2.2.3 节 "配置内核收集器"。*

7.2.3.5. 启用服务

*要在引导时启动 **kdump** 服务，点工具栏上的 **启用** 按钮，然后点击 **Apply** 按钮。这为 **multi-user.target** 启用并激活服务。单击**"禁用 "按钮，然后单击"应用 "按钮以立即禁用该服务。***

重要

*在 Red Hat Enterprise Linux 7 中，kdump systemd 服务启动时，定义为 **kdump** 目标的目录必须存在 - 否则服务失败。此行为与早期版本的 Red Hat Enterprise Linux 不同，如果启动服务时不存在 目录，则会自动创建该目录。*

有关 systemd 目标和配置服务的更多信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南 。

7.3. 为 KDUMP 将内核驱动程序列入黑名单

将内核驱动程序列入黑名单是一种防止加载和使用内核驱动程序的机制。在 **/etc/sysconfig/kdump** 文件中添加驱动程序可防止 kdump **initramfs** 加载列入黑名单的模块。

将内核驱动程序列入黑名单可防止延迟 **终止** 程序或其他崩溃内核失败。要将内核驱动程序列入黑名单，您可以在 **/etc/sysconfig/kdump** 文件中更新 **KDUMP_COMMANDLINE_APPEND=** 变量，并指定以下黑名单选项之一：

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

流程

1. 选择您要列入黑名单的内核模块：

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

   lsmod 命令显示载入到当前运行的内核的模块列表。

2. 更新 **/etc/sysconfig/kdump** 文件中的 **KDUMP_COMMANDLINE_APPEND=** 行，如下所示：

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

3. 您还可以更新 **/etc/sysconfig/kdump** 文件中的 **KDUMP_COMMANDLINE_APPEND=** 行，如下所示：

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

4. 重启 kdump 服务：

```
$ systemctl restart kdump
```

7.4. 测试 KDUMP 配置

警告

以下命令会导致内核崩溃。遵循这些步骤时要小心，且不得在生产环境中使用它们。

要测试配置，请使用启用 **kdump** 重启系统，并确保该服务正在运行：

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

然后在 shell 提示符后输入以下命令：

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

这会强制 Linux 内核崩溃，**地址-YYYY-MM-DD-HH:MM:SS/vmcore** 文件复制到您在配置中选择的位置（即默认为 **/var/crash/**）。

注意

除了确认配置的有效性外，此操作还可用于记录崩溃转储在代表测试负载下执行时需要多长时间才能完成。

7.4.1. 其它资源

7.4.1.1. 安装的文档

- **kdump.conf**(5)- **/etc/kdump.conf** 配置文件的 man page，包含可用选项的完整文档。
- **zipl.conf**(5)- **/etc/zipl.conf** 配置文件的 man page。
- **zipl**(8)- IBM Z 的 **zipl** 引导装载程序实用程序的 man page。
- **makedumpfile**(8)- **makedumpfile** 内核收集器的 man page。
- **kexec**(8)- **kexec** 的 man page.
- **crash**(8)- **crash** 实用程序的 man page。
- **/usr/share/doc/kexec-tools-版本/kexec-kdump-howto.txt** - 概述 **kdump** 和 kexec 安装及用法。

7.4.1.2. 在线文档

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

关于 **kexec** 和 **kdump** 配置的红帽知识库文章。

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

关于支持的 **kdump** 目标的红帽知识库文章。

https://github.com/crash-utility/crash

crash 实用程序 Git 存储库.

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

GRUB2 引导装载程序主页和文档.

7.5. 固件支持的转储机制

7.5.1. 固件支持的转储案例

kexec 和 **kdump** 机制是在 AMD64 和 Intel 64 系统中捕获内核转储的可靠且可靠的方法。但是，一些历史较长的硬件（尤其是小型和大型机系统）使我们能够利用板载固件隔离内存区域，并防止意外覆盖对崩溃分析很重要的数据。

本章论述了一些可用的固件支持的转储方法，以及它们如何与 Red Hat Enterprise Linux 集成。

7.5.2. 在 IBM PowerPC 硬件中使用 fadump

固件辅助转储(**fadump**)是 IBM PowerPC LPARS 上提供的 **kexec-kdump** 的可靠替代方案。它从完全重设的系统捕获 vmcore，其 PCI 和 I/O 设备重新初始化。虽然这种机制使用固件在崩溃时保留内存，但它会重复使用 **kdump** 用户空间脚本来保存 vmcore"。

为实现这一目标，**fadump** 注册了系统固件发生崩溃时必须保留的内存区域。除了引导内存、系统寄存器和硬件页表条目(PTE)外，这些区域包含所有系统内存内容。

有关 **fadump** 机制（包括针对 PowerPC 重置硬件的方法）的详情，请查看 **/usr/share/doc/kexec-tools-X.y.z/fadump-howto.txt**，其中 "X.y.z" 与您的系统上安装的 **kexec-tools** 版本号对应。

注意

未保留的内存区域和称为 **引导内存** 的区域是崩溃事件后成功启动内核所需的 RAM 量。默认情况下，引导内存大小为 256MB 或系统 RAM 总量的 5%，以较大者为准。

与 **kexec**-initiated 事件不同，**fadump** 进程使用 production 内核恢复崩溃转储。在崩溃后引导时，PowerPC 硬件使设备节点 **/proc/device-tree/rtas/ibm,kernel-dump** 可供 **procfs** 使用，而 procfs 会检查 fadump-aware **kdump** 脚本以保存 vmcore。完成此操作后，系统将完全重启。

启用 fadump

1. 安装和配置 **kdump**，如 第 7.2 节 “安装和配置 kdump” 所述。
2. 在 **/etc/default/grub** 中的 **GRUB_CMDLINE_LINUX** 行中添加 **fadump=on** ：

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

3. （可选）如果要指定保留引导内存而不是接受默认值，请在 **/etc/default/grub** 中将 **crashkernel=xxM** 配置为 **GRUB_CMDLINE_LINUX**，其中 xx 是以 MB 为单位所需的内存量：

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

重要

与所有引导选项一样，强烈建议您在需要前测试配置。如果您在从崩溃内核引导时观察到内存不足(OOM)错误，请提高 **crashkernel=** 中指定的值，直到崩溃内核可以完全引导。在本例中可能会需要进行一些不同的尝试。

7.5.3. IBM Z 支持的固件转储方法

IBM Z 上有两个固件支持的转储机制。它们是独立的 **Dump** 和 **VMDUMP**。

这些系统中支持并使用 **kdump** 基础架构，来自 Red Hat Enterprise Linux 的配置包括在 第 7.2 节 “安装和配置 kdump” 中。但是，使用固件支持的方法之一 IBM Z 硬件可能有一些优点。

单机转储(SADMP)机制从系统控制台启动和控制，必须存储在 IPL 可引导设备中。

与 SADMP 类似，是 VMDUMP。此工具也从系统控制台启动，但具有从硬件检索生成的转储并将其复制到系统以进行分析的机制。

这些方法的一个优点（与其他基于硬件的转储机制类似）是可以在 Early Boot 阶段捕获机器的状态（在 kdump 服务启动前）

尽管 VMDUMP 包含将转储文件接收到 Red Hat Enterprise Linux 系统中的机制，但 SADMP 和 VMDUMP 的配置和控制都是从 IBM Z 硬件控制台管理的。

IBM 在 VMDUMP 文章中详细讨论 SADMP、独立转储程序文章和 VMDUMP。

IBM 还具有用于使用 Red Hat Enterprise Linux 7 中使用 Dump Tools on Red Hat Enterprise Linux 的转储工具的文档集。

7.5.4. 在 Fujitsu PRIMEQUEST 系统中使用 sadump

Fujitsu **sadump** 机制旨在在 **kdump** 无法成功完成时提供回退转储捕获。

sad **ump** 进程是从系统 ManageMent Board(MMB)接口手动调用的。

使用这个系统，为 X86_64 服务器配置 kdump，然后执行以下步骤启用 sad **ump**。

在 **/etc/sysctl.conf** 中添加或编辑以下行，以确保 sad **ump** 的 **kdump** 启动如预期。

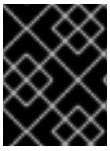
```
kernel.panic=0
kernel.unknown_nmi_panic=1
```

除以上所述外，还必须在 **/etc/kdump.conf** 中添加一些选项，以确保 **kdump** 对 sadump 的行为正确。

特别是，请确保在 **kdump** 后系统不会重启。如果系统在 **kdump** 无法保存内核后重启，则您没有机会调用 sad **ump**。

将 **/etc/kdump.conf 中的默认** 操作设置为 halt 或 shell 以实现此目的。

```
default shell
```



重要

有关为 sad **ump** 配置硬件的详情，请查看 FUJITSU Server PRIMEQUEST 2000 系列安装手册。

7.6. 分析内核转储

要确定系统崩溃的原因，您可以使用 **crash** 实用程序，它提供了一个与 GNU Debugger(GDB)类似的交互式提示符。这个工具允许您交互式地分析正在运行的 Linux 系统，以及由 **netdump**、**diskdump**、xen**dump 或 kdump** 创建的核心转储。

7.6.1. 安装 crash 工具

要安装 **崩溃分析工具**，以 **root** 用户身份从 shell 提示符执行以下命令：

```
yum install crash
```

除了 **崩溃** 外，还需要安装与正在运行的内核对应的 **kernel-debuginfo** 软件包，该软件包提供了转储分析所需的数据。要安装 **kernel-debuginfo**，我们以 **root 用户身份使用 debuginfo-install** 命令：

```
debuginfo-install kernel
```

有关如何使用 **Yum** 软件包管理器在 Red Hat Enterprise Linux 中安装新软件包的更多信息，请参阅 *Red Hat Enterprise Linux 7 系统管理员指南*。

7.6.2. 运行 crash 工具

要启动该实用程序，在 shell 提示符后以以下格式输入命令：

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

使用 **kdump** 捕获的相同 <kernel> 版本。要查找您当前运行的内核，请使用 **uname -r** 命令。

例 7.2. 运行 crash 工具

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

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

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

      KERNEL: /usr/lib/debug/lib/modules/2.6.32-69.el6.i686/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2010-08-25-08:45:02/vmcore  [PARTIAL DUMP]
        CPUS: 4
        DATE: Wed Aug 25 08:44:47 2010
      UPTIME: 00:09:02
LOAD AVERAGE: 0.00, 0.01, 0.00
       TASKS: 140
    NODENAME: hp-dl320g5-02.lab.bos.redhat.com
     RELEASE: 2.6.32-69.el6.i686
     VERSION: #1 SMP Tue Aug 24 10:31:45 EDT 2010
     MACHINE: i686  (2394 Mhz)
```

```
      MEMORY: 8 GB
       PANIC: "Oops: 0002 [#1] SMP " (check log for details)
         PID: 5591
     COMMAND: "bash"
        TASK: f196d560  [THREAD_INFO: ef4da000]
         CPU: 2
       STATE: TASK_RUNNING (PANIC)

crash>
```

7.6.3. 显示消息缓冲

若要显示内核消息缓冲区，可在交互式提示符处键入 **log** 命令。

例 7.3. 显示内核消息缓冲

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

键入 **help log** 以了解有关命令用法的更多信息。

注意

内核消息缓冲区包含有关系统崩溃的最重要信息，因此始终先转储到 **vmcore-dmesg.txt** 文件。当尝试使完整的 **vmcore** 文件失败时（例如，目标位置上缺少空间），这很有用。默认情况下，**vmcore-dmesg.txt** 位于 **/var/crash/** 目录中。

7.6.4. 显示后端

若要显示内核堆栈跟踪，可在交互式提示符处键入 **bt** 命令。您可以使用 **bt <pid>** 来显示单个进程的后端。

例 7.4. 显示内核堆栈追踪

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

键入 **help bt** 以了解有关命令用法的更多信息。

7.6.5. 显示进程状态

要显示系统中进程的状态，请在交互式提示符处键入 **ps** 命令。您可以使用 **ps <pid>** 显示单个进程的状态。

例 7.5. 显示系统中进程的状态

```
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
```

键入 **help ps** 来了解有关命令用法的更多信息。

7.6.6. 显示虚拟内存信息

要显示基本虚拟内存信息，请在交互式提示符下键入 **vm** 命令。您可以使用 **vm <pid>** 显示单个进程的信息。

例 7.6. 显示当前上下文的虚拟内存信息

```
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 ...
```

键入 **help vm** 以了解有关命令使用情况的更多信息。

7.6.7. 显示打开的文件

要显示有关打开文件的信息，请在交互式提示符下键入 **files** 命令。您可以使用 **文件 <pid> 来** 只显示一个选定进程打开的文件。

例 7.7. 显示有关当前上下文打开文件的信息

```
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
```

键入 **help 文件** 以了解有关命令用法的更多信息。

7.6.8. 退出实用程序

要退出交互式提示符并终止 **崩溃**，请键入 **exit** 或 **q**。

例 7.8. 退出 crash 工具

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

7.7. 常见问题解答

在群集环境中使用 Kdump 时需要考虑哪些事项？

如何配置 kdump 以用于 RHEL 6 7 高可用性附加组件？显示使用高可用性附加组件的系统管理员可以使用的选项。

Kdump 在早期启动过程中失败，如何捕获引导日志？

如果启动第二个内核时出现问题，则需要查看早期启动日志，可以通过在受影响的机器上启用串行控制台来获取这些日志。

我如何在 RHEL7 中设置串行控制台？显示启用对早期引导消息的访问所需的配置。

如何增加来自 makedumpfile 进行调试的消息传递？

如果 **makedumpfile** 失败，则需要提高日志级别来了解错误。这与设置转储级别不同，通过编辑 **/etc/kdump.conf** 并增加 **message_level** 选项以在 **core_collector** 行条目上创建 **dumpfile** 来实现。

默认情况下，**makedumpfile** 设置为级别 1，这会将输出限制为进度指示符。通过将此消息级别设置为 31 来启用所有调试信息。消息级别 31 将打印进度指示符、常见消息、错误消息、调试消息和报告消息的详细信息。

注意

有关消息级别选项的详情，请查看 **makedumpfile(8)** 手册页。

确定您的 core_collector 配置行在设置时类似如下：

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

如何调试 Dracut？

一些时间 **dracut** 可能无法构建 initramfs。如果发生这种情况，请提高日志级别 in **dracut** 来隔离问题。

编辑 **/etc/kdump.conf** 并更改 **dracut_args** 行，使其除您需要的任何其他 dracut 参数外还包含选项 **-L 5**。

如果您没有配置其它选项 in **dracut_args**，结果类似如下：

```
dracut_args -L 5
```

虚拟机可以使用哪种转储方式？

在大多数情况下，kdump **机制** 足以在崩溃或 panic 后从机器获取内存转储。这可以像安装到裸机一样进行设置。

然而，在某些情况下，需要直接与管理程序合作来获取崩溃转储。**libvirt** 可通过两种机制实现此目标：pv **panic** 和 **virsh 转储**。这两个方法均在《虚拟化部署和管理指南》中进行了说明。

pvpanic 机制可在《虚拟化部署和管理指南》中找到 - 设置 Panic 设备.

在《虚拟化部署和管理指南》 - 创建域核心的转储文件时，对 **virsh dump** 命令进行了讨论。

如何上传大型转储文件到红帽支持服务？

在某些情况下，可能需要向 Red Hat 全球支持服务发送内核崩溃转储文件进行分析。但是，转储文件可能非常大，即使在过滤后也是如此。由于在打开新的支持问题单时，大于 250 MB 的文件无法通过红帽客户门户网站直接上传，红帽会提供 FTP 服务器以上传大型文件。

FTP 服务器的地址为 **dropbox.redhat.com**，这些文件将上传到 **/in传入/** 目录中。您的 FTP 客户端需要设置为被动模式；如果您的防火墙不允许此模式，请使用活动模式的 **origin-dropbox.redhat.com** 服务器。

确保上传的文件使用 **gzip** 等程序进行压缩，并且使用描述性正确命名。建议在文件名中使用您的支持问题单号。成功上传所有必要文件后，请为工程师提供确切的文件名及其 SHA1 或 MD5 校验和。

如需了解更多具体步骤和其他信息，请参阅如何向红帽支持提供文件。

完成崩溃转储需要多长时间？

对于灾难恢复规划而言，通常需要知道转储完成所需的时间。但是，需要花费的时间长度高度依赖于复制到磁盘的内存量以及 RAM 和存储之间的接口速度。

对于时间测试，系统必须在代表负载下运行，否则页面排除选项可能会使用完全载入的生产系统给出 kdump 行为的假视图。在处理大量 RAM 时，尤其会出现这种差异。

在评估转储时间时，规划中的存储接口也应考虑。由于网络限制，例如，通过 **ssh** 进行连接转储可能需要比本地附加的 SATA 磁盘更长的时间。

安装期间如何配置 Kdump？

您可以在安装过程中使用一组有限的选项在 kickstart 或交互式 GUI 中配置 **kdump**。

使用 **anaconda** 安装 GUI 的 **kdump** 配置记录在《安装指南》的 KDUMP 部分。

kickstart 语法是：

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

使用这个附加组件到 Kickstart，您可以禁用或启用 kdump 功能（可选地定义保留内存大小），或者通过显式调用 auto 的默认选项（忽略整个交换机的情况）或者指定以 MB 为单位的数字值。

要了解如何使用 Kickstart 自动执行系统部署，请阅读《安装指南》中的 Kickstart 安装。

有关 Kickstart 附加组件语法的详情，请查看《安装指南》中的 Kickstart 语法参考。

7.8. 支持的 KDUMP 配置和目标

7.8.1. kdump 的内存要求

为了使 kdump 能够捕获内核崩溃转储并保存以便进一步分析，系统内存部分必须永久保留给捕获内核。

有关如何在命令行中更改内存设置的详情请参考 第 7.2.2.1 节 "配置内存用量"。有关如何在图形用户界面中设置保留内存量的步骤，请参考 第 7.2.3.1 节 "配置内存用量"。

表 7.1 列出 **kdump** 在 **内核和 kernel -alt** 软件包中自动保留的内存。**kdump** 根据 CPU 架构和可用物理内存自动保留内存。

表 7.1. kdump自动保留崩溃内存总数

CPU 架构	可用内存	崩溃内存自动保留
AMD64 和 Intel 64(x86_64)	2 GB 及更多	每个 1 TB 161 MB + 64 MB
64 位 ARM 架构(arm64)	2 GB 及更多	512 MB
IBM POWER ppc64/ppc64le	2 GB 到 4 GB	384 MB
	4 GB 到 16 GB	512 MB
	16 GB 到 64 GB	1 GB
	64 GB 到 128 GB	2 GB
	128 GB 及更多	4 GB
IBM Z (s390x)	4 GB 及更多	每个 1 TB 的 161 MB + 64 MB [1] 160 MB on RHEL-ALT-7.6

有关各种 Red Hat Enterprise Linux 技术功能和限制的详情请参考 https://access.redhat.com/articles/rhel-limits。

7.8.2. 自动内存保留的最小阈值

在某些系统中，可以使用引导加载器配置文件中的 **crashkernel=auto** 参数自动为 kdump 分配内存，也可以在图形配置工具中启用这个选项。但是，要使此自动预留发挥作用，系统中需要提供一定大小的总内存。这一数量因系统架构而异。

下表列出了 **内核和 kernel -alt** 软件包自动分配内存的阈值。如果系统内存少于表中指定的内存，则需要手动保留内存。

有关如何在命令行中更改这些设置的详情请参考 第 7.2.2.1 节 "配置内存用量"。有关如何在图形用户界面中更改保留内存量的步骤请参考 第 7.2.3.1 节 "配置内存用量"。

表 7.2. 自动内存保留所需的最小内存量

构架	所需的内存
AMD64 和 Intel 64(**x86_64**)	2 GB
IBM POWER(**ppc**64)	2 GB
IBM Z (**s390x**)	4 GB
64 位 ARM 架构(**ARM**64)	2 GB

7.8.3. 支持的 kdump 目标

当捕获内核崩溃时，核心转储可以直接写入到设备中，作为文件存储在本地文件系统中，或者通过网络发送。下表包含 kdump 支持或明确不支持的转储目标的完整列表。

有关如何在命令行中配置目标类型的详情请参考 第 7.2.2.2 节 "配置 kdump 目标"。有关如何在图形用户界面中做到这一点的详情请参考 第 7.2.3.2 节 "配置 kdump 目标"。

表 7.3. 支持的 kdump 目标

类型	支持的目标	不支持的目标
原始设备	所有本地附加的原始磁盘和分区。	
本地文件系统	直接附加的磁盘驱动器、硬件 RAID 逻辑驱动器、LVM 设备以及 **mdraid** 阵列中的 **ext 2**、ext**3**、ext4 和 **xfs** 文件系统。	在此表中未明确列出的任何本地文件系统，包括 **自动** 类型（自动文件系统检测）。
远程目录	使用 **NFS** 或 **SSH** 协议通过 **IPv4** 访问的远程目录。	使用 **NFS** 协议访问的 **rootfs** 文件系统上的远程目录。
使用 **FCoE** （*以太网光纤通道*）协议访问的远程目录。	**qla2xxx**、**lpfc** 和 **bfa** 硬件 **FCoE** 目标.**bnx2fc** 和 **ixgbe** 软件 **FCoE** 目标.	
通过硬件和软件启动器使用 **iSCSI** 协议访问的远程目录。	使用 **be2iscsi** 硬件上的 **iSCSI** 协议访问的远程目录.	基于多路径的存储。
		通过 **IPv6** 访问的远程目录.
		使用 **SMB** 或 **CIFS** 协议访问的远程目录.

类型	支持的目标	不支持的目标
		使用无线网络接口访问的远程目录。

注意

*当转储到软件 **FCoE** 目标时，您可能会遇到内存不足(OOM)问题。在这种情况下，会增加默认的 **crashkernel=auto** 参数值。有关如何设置这个内核引导参数的详情请参考 第 7.2.2.1 节 "配置内存用量"。*

7.8.4. 支持的 kdump 过滤等级

*要缩小转储文件的大小，kdump 使用 **makedumpfile** 内核收集器压缩数据，并选择性地退出不相关的信息。下表包含 **makedumpfile** 实用程序目前支持的过滤级别的完整列表。*

有关如何在命令行中配置内核收集器的说明，请参阅 第 7.2.2.3 节 "配置内核收集器"。有关如何在图形用户界面中做到这一点的详情请参考 第 7.2.3.3 节 "配置内核收集器"。

表 7.4. 支持的过滤级别

选项	描述
1	零页
2	缓存页
4	缓存私有
8	用户页
16	可用页

注意

***makedumpfile** 命令支持在 Red Hat Enterprise Linux 7.3 及之后的版本中删除透明大内存页和 hugetlbfs 页面。考虑这两种类型的大页用户页面，并使用 **-8** 级别将其删除。*

7.8.5. 支持的默认操作

默认情况下，当 kdump 创建内核转储失败时，操作系统会重启。但是，您可以将 kdump 配置为在将内核转储保存到主目标时执行不同的操作。下表列出了 kdump 目前支持的所有默认操作。

有关如何在命令行中设置默认操作的详情请参考 第 7.2.2.4 节 "配置默认操作"。有关如何在图形用户界面中做到这一点的详情请参考 第 7.2.3.4 节 "配置默认操作"。

表 7.5. 支持的默认操作

选项	描述
dump_to_rootfs	尝试将内核转储保存到 root 文件系统。这个选项在与网络目标合并时特别有用：如果网络目标无法访问，这个选项配置 kdump 以在本地保存内核转储。之后会重启该系统。
reboot	重启系统，这个过程会丢失 core 转储文件。
halt	关闭系统，这个过程会丢失 core 转储文件。
poweroff	关闭系统，这个此过程会丢失 core 转储。
shell	从 initramfs 内运行 shell 会话，允许用户手动记录核心转储。

7.8.6. 估算 kdump 大小

*在规划和构建 **kdump** 环境时，需要知道在生成转储文件前需要多少空间。**makedumpfile** 命令可以帮助执行此操作。*

*使用 **--mem-usage** 功能估算转储文件所需的空间，如下所示：*

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

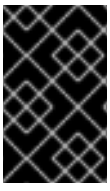
注意

*使用 **-f** 选项的 **--mem-usage** 功能适用于内核版本的 v4.11 及更新的版本。*

*对于早于 v4.11 的内核版本，在使用 **--mem-usage** 选项 **-f** 之前，请确保内核通过上游提交 464920104bf7 进行补丁。*

***mem-usage** 选项提供有关可扩展页面的有用报告，可用于确定您要分配的转储级别。当系统处于代表负载下时运行此命令，否则 **makedumpfile** 会返回比生产环境中预期的值小的值。*

```
[root@hostname ~]# makedumpfile -f --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
```



重要

***makedumpfile** 命令在页面中 **报告**。这意味着您必须根据内核页面大小（在 Red Hat Enterprise Linux 内核中）计算使用的内存大小（在 Red Hat Enterprise Linux 内核中）对于 AMD64 和 Intel 64 构架，以及 64 KB 用于 IBM POWER 架构。*

7.8.7. 支持 内核和 kernel -alt 软件包中的构架

下表提供了 **内核和 kernel -alt** 软件包上的架构和可用内存支持概述。

表 7.6. 内核和 kernel -alt 软件包支持的构架

CPU 架构	可用内存	RHEL 7.5 及更早版本	RHEL 7.6 及更新的版本	RHEL-ALT-7.4	rhel-ALT-7.5 及更新的版本
AMD64 和 Intel 64(x86_64)	2GB 及更多	每个 1TB 161MB + 64MB	161MB + 64MB/1TB	2GB 到 160MB	2GB 到 160MB
64 位 ARM 架构(arm64)	2GB 及更多	不适用	不适用	2GB 到 512MB	2GB 到 512MB
IBM POWER ppc64/ppc64le（下至 POWER8）	2GB 到 4GB	384MB	384MB	不适用	不适用
	4GB 到 16GB	512MB	512MB	不适用	不适用
	16GB 到 64GB	1GB	1GB	不适用	不适用
	64GB 到 128GB	2GB	2GB	不适用	不适用
	128GB 及更多	4GB	4GB	不适用	不适用
IBM POWER ppc64/ppc64le（下至 POWER9）	2GB 到 4GB	384MB	384MB	384MB	384MB
	4GB 到 16GB	512MB	512MB	512MB	512MB
	16GB 到 64GB	1GB	1GB	1GB	1GB
	64GB 到 128GB	2GB	2GB	2GB	2GB
	128GB 及更多	4GB	4GB	4GB	4GB
IBM POWER ppc64le(POWER9)	2GB 到 4GB	不适用	不适用	384MB	384MB
	4GB 到 16GB	不适用	不适用	512MB	512MB
	16GB 到 64GB	不适用	不适用	1GB	1GB
	64GB 到 128GB	不适用	不适用	2GB	2GB
	128GB 及更多	不适用	不适用	4GB	4GB

CPU 架构	可用内存	RHEL 7.5 及更早版本	RHEL 7.6 及更新的版本	RHEL-ALT-7.4	rhel-ALT-7.5 及更新的版本
IBM Z (s390x)	4GB 及更多	每个 1TB 161MB + 64MB	每个 1TB 161MB + 64MB	每个 1TB 161MB + 64MB	160MB

7.9. 使用 KEXEC 重启内核

7.9.1. 使用 kexec 重启内核

kexec 系统调用启用从当前运行的内核加载并引导至另一个内核的功能，从而从内核中执行引导装载程序的功能。

kexec 实用程序为 **kexec** 系统调用加载内核和 **initramfs** 镜像，以引导至另一个内核。

下面的部分论述了如何在使用' **kexec**' 实用程序重启到另一个内核时手动调用 **k**exec 系统调用。

1. 执行 **kexec** 工具：

```
# kexec -l /boot/vmlinuz-3.10.0-1040.el7.x86_64 --initrd=/boot/initramfs-3.10.0-1040.el7.x86_64.img --reuse-cmdline
```

该命令为 **kexec** 系统调用手动加载内核和 initramfs 镜像。

2. 重启系统：

```
# reboot
```

该命令会检测内核，关闭所有服务，然后调用 **kexec** 系统调用来重新引导到您在上一步中提供的内核中。

警告

当您使用 **kexec -e** 命令重新引导内核时，系统不会在启动下一个内核前完成标准关闭序列，这可能会造成数据丢失或系统无响应。

7.10. 与 KDUMP 相关的门户实验

Portal Labs 是小型 Web 应用程序，可帮助系统管理员执行多个系统任务。目前有两个侧重于 Kdump 的实验。Kdump Helper 和 Kernel Oops Analyzer.

7.10.1. kdump helper

Kdump Helper 是一系列问题和操作，可帮助为 **kdump** 准备配置文件。

Lab 的工作流包括集群和独立环境的步骤。

7.10.2. 内核 oops 分析器

Kernel Oops Analyzer 是一个工具，可用于处理 Oops 消息和搜索已知解决方案，而无需取消崩溃转储堆栈。

Kernel Oops Analyzer 使用 **makedumpfile** 中的信息将崩溃机器中的 oops 消息与知识库中的已知问题进行比较。这可让系统管理员在意外中断后快速排除已知问题，并在打开支持问题单以进行进一步分析之前。

第 8 章 使用内核完整性子系统提高安全性

8.1. 内核完整性子系统

完整性子系统是负责维护系统数据完整性的内核的一部分。此子系统有助于防止用户对特定系统文件进行不必要的修改，从而使系统处于初始状态。

内核完整性子系统由两个主要组件组成：

完整性测量架构(IMA)

- 在文件被执行或打开时，会测量文件的内容。用户可以通过应用自定义策略来更改此行为。
- 将测量的值放置在内核的内存空间中，从而防止系统用户进行任何修改。
- 允许本地和远程用户验证测量值。

扩展验证模块(EVM)

- 通过加密其对应的值，保护与系统安全性（如 IMA 测量和 SELinux 属性）相关的文件的扩展属性（也称为 xattr）。

IMA 和 EVM 还包含大量额外功能扩展。例如：

IMA-Appraisal

- 本地根据以前存储在内核内存中测量文件中的值验证当前文件的内容。此扩展禁止在当前测量与之前的测量不匹配的情况下通过特定文件执行任何操作。

EVM 数字签名

- 允许通过存储在内核密钥环中的加密密钥使用数字签名。EVM Digital Signatures 确保包含内容哈希(**security.ima**)的文件的 xattr 值的来源和完整性。

注意

功能扩展相互补充，但您可以独立配置和使用它们。

内核完整性子系统可以使用 Trusted Platform 模块(TPM)来进一步增强系统安全性。TPM 是受信任的计算组(TCG)中有关重要加密功能的规范。TPM 通常作为专用硬件实施，附加到平台的主板，并通过为硬件芯片受保护且受篡改区域提供加密功能来防止基于软件的攻击。TPM 的一些功能有：

- 随机数生成器
- 用于加密密钥的生成器和安全存储
- 哈希生成器
- 远程测试

8.2. 完整性测量架构

完整性测量架构(IMA)是内核完整性子系统的一个组件。IMA 旨在通过在访问之前测量、存储和应用垃圾文件哈希来维护本地文件的内容。这可防止读取和执行不可靠数据并增强系统安全性。

8.3. 扩展的验证模块

扩展验证模块(EVM)是内核完整性子系统的一个组件，可监控文件的扩展属性(xattr)中的更改。许多面向安全的技术，包括完整性测量架构(IMA)，将敏感文件信息（如内容散列）存储在扩展属性中。EVM 从这些扩展属性和特殊密钥创建另一个哈希值，该密钥在引导时加载。每次使用扩展属性时，生成的哈希都会被验证。例如，当 IMA 评估 文件时。

8.4. 可信和加密的密钥

受信任和 加密的密钥是内核生成的可变长度对称密钥，供内核密钥环服务使用。这种类型的密钥永远不会以未加密的形式出现在用户空间中，这意味着可以验证其完整性。因此，例如扩展验证模块(EVM) 可以使用它们来验证并确认运行中系统的完整性。用户级别程序只能访问加密的 Blob 格式的密钥。

受信任的密钥需要硬件组件：受信任的平台模块(TPM)芯片，它用于创建和加密密钥。TPM 使用名为存储根密钥(SRK)的 2048 位 RSA 密钥密封密钥。

有关可信和加密密钥的更多信息，请参阅 RHEL 7 安全指南中的受 信任和加密密钥 部分。

8.5. 启用完整性测量架构和扩展验证模块

完整性测量架构(IMA)和扩展验证模块(EVM)是内核完整性子系统的组成部分，以各种方式增强系统安全性。配置 IMA 和 EVM 可让您签署文件，从而增强系统的安全性。

先决条件

- The **ima-evm-utils** 和 **keyutils** 软件包安装在您的系统上。
- **securityfs** 文件系统挂载到 **/sys/kernel/security/** 目录。
- 存在 **/sys/kernel/security/ima/** 目录。

```
# mount
…
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
…
```

流程

1. 准备您的系统以启用 IMA 和 EVM：
   a. 添加以下内核命令行参数：

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="ima_appraise=fix
ima_appraise_tcb evm=fix"
```

   命令在 fix 模式中为当前引导条目启用 IMA 和 EVM，并允许用户收集和更新 IMA 测量。

   The **ima_appraise_tcb** 内核命令行参数确保内核使用默认的可信计算基础(TCB)测量策略和实例步骤。强制禁止访问之前和当前测量结果不匹配的文件。

   b. 重启以使更改生效。

c. *（可选）验证内核命令行是否包含新参数：*

```
# cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-1136.el7.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto spectre_v2=retpoline rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
LANG=en_US.UTF-8 ima_appraise=fix ima_appraise_tcb evm=fix
```

2. *为 EVM 创建并设置公共和私有密钥对：*

a. *为 EVM 创建新密钥环：*

```
# evm_kr_id=$(keyctl newring _evm @u)
```

*命令创建 **_evm** 密钥环，并将它连接到 **@u** 系统用户密钥环。然后，_ **evm** 的密钥环 ID 分配给 **evm_kr_id** 变量，以便更轻松地处理。*

b. *另外，还可查看新创建的密钥环：*

```
# keyctl show
Session Keyring
1025767139 --alswrv   0     0  keyring: _ses
548660789 --alswrv   0 65534   \_ keyring: _uid.0
456142548 --alswrv   0     0       \_ keyring: _evm
```

c. *为密钥创建一个目录：*

```
# mkdir -p /etc/keys/
```

d. *在 **/etc/keys/privkey.pem** 文件中生成 1024 位 RSA 私钥：*

```
# openssl genrsa -out /etc/keys/privkey.pem 1024
Generating RSA private key, 1024 bit long modulus
.......++++++
...++++++
e is 65537 (0x10001)
```

e. *使用之前创建的 **/etc/keys/privkey.pem** 私钥，将对应的 RSA 公钥派生到 **/etc/keys/pubkey.pem** 文件中：*

```
# openssl rsa -pubout -in /etc/keys/privkey.pem -out /etc/keys/pubkey.pem
writing RSA key
```

f. *将公钥导入到专用的 EVM 密钥环中：*

```
# evmctl import --rsa /etc/keys/pubkey.pem $evm_kr_id
1054989579
```

*命令将 **/etc/keys/pubkey.pem** 公钥导入到 **_evm** 密钥环中。然后，**_evm** 密钥环附加到内核密钥环。密钥序列号位于上例的第二行。*

g. *另外，还可查看新导入的密钥：*

```
# keyctl show
Session Keyring
```

```
1025767139 --alswrv   0     0  keyring: _ses
548660789 --alswrv   0 65534   \_ keyring: _uid.0
456142548 --alswrv   0     0       \_ keyring: _evm
1054989579 --alswrv   0     0          \_ user: FA0EF80BF06F80AC
```

注意

此非对称密钥对可用于使用 **evmctl 符号** 命令以数字方式签署文件的扩展属性的内容。扩展属性稍后由内核验证。

h. 创建一个内核主密钥来保护 EVM 密钥：

```
# dd if=/dev/urandom bs=1 count=32 2>/dev/null | keyctl padd user kmk-user @u
```

内核主密钥(**kmk**)完全保留在内核空间内存中。内核主密钥 **kmk** 的 32 字节长值是从 **/dev/urandom** 文件中随机字节生成的，并放置在用户(@**u**)密钥环中。

3. 根据 **kmk** 密钥创建一个加密的 EVM 密钥：

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
351426499
```

命令使用 **kmk** 生成并加密 64 字节用户密钥（名为 **evm-key**）并将其放置在用户(@**u**)密钥环中。密钥序列号位于上例的第二行。

重要

必须为用户密钥 **evm-key 命名**，因为这是 EVM 子系统预期并可使用的名称。

4. 激活 EVM:

```
# echo 1 > /sys/kernel/security/evm
```

5. 验证 EVM 是否已初始化：

```
dmesg | tail -1
[…] EVM: initialized
```

8.6. 使用完整性测量架构收集文件哈希

完整性测量架构(IMA) 的第一级操作是测量阶段，它允许创建文件哈希并将其存储为这些文件的扩展属性 (xattrs)。下面的部分论述了如何创建和检查文件的哈希。

先决条件

- The **ima-evm-utils**、**attr** 和 **keyutils** 软件包安装在您的系统上。
- 完整性测量架构(IMA)和扩展验证模块(EVM)启用，如 第 8.5 节 "启用完整性测量架构和扩展验证模块" 所述。

流程

1. 创建测试文件：

```
# echo <Test_text> > test_file
```

2. 使用私钥签署文件：

```
# evmctl sign --imahash --key /etc/keys/privkey.pem test_file
```

通过创建 **test_file** 文件的哈希，IMA 验证文件是否仍然未损坏。EVM 通过签名存储在 **test_file** 扩展属性中的哈希内容来确保 IMA 哈希是真实的。

3. （可选）检查签名文件的扩展属性：

```
# getfattr -m . -d test_file
file: test_file
security.evm=0sAwICztVdCQCAZLtD7qAezGl8nGLgqFZzMzQp7Fm1svUet2Hy7TyI2vtT9/9Zf
BTKMK6Mjoyk0VX+DciS85XOCYX6WnV1LF2P/pmPRfputSEq9fVD4SWfKKj2rI7qwpndC1UqR
X1BbN3aRUYeoKQdPdl6Cz+cX4d7vS56FJkFhPGlhq/UQbBnd80=
security.ima=0sAfgqQ5/05X4w/ltZEfbogdI9+KM5
security.selinux="unconfined_u:object_r:admin_home_t:s0"
```

这个示例输出显示了与 SELinux 以及 IMA 和 EVM 哈希值相关的扩展属性。EVM 主动添加 **security.evm** 扩展属性，并检测对其他文件（如 **security.ima** ）与文件内容完整性直接相关的 xattrs 的任何离线篡改。**security.evm** 字段的值位于基于 Hash 的消息身份验证代码(HMAC-SHA1)中，该身份验证代码是使用私钥生成的。

其它资源

- 有关内核完整性子系统的详情，请查看 官方上游 Wiki 页面。
- 有关 TPM 的更多信息，请参阅 受信任的计算组资源。
- 有关创建加密密钥的更多信息，请参阅 RHEL 7 安全指南中的受 信任和加密密钥 部分。

第 9 章 修订历史记录

0.1-8

Tue Sep 29 2020, Jaroslav Klech (jklech@redhat.com)

- 发布 7.9 GA 的文件版本.

0.1-7

Tue Mar 31 2020, Jaroslav Klech (jklech@redhat.com)

- 发布 7.8 GA 的文档版本.

0.1-6

Tue Aug 6 2019, Jaroslav Klech (jklech@redhat.com)

- 发布 7.7 GA 的文档版本.

0.1-5

Fri Oct 19 2018, Jaroslav Klech (jklech@redhat.com)

- 发布 7.6 GA 的文档版本.

0.1-4

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

- 发布 7.5 GA 的文档版本.

0.1-3

2018 年 1 月 5 日，Mark Flitter(mflitter@redhat.com)

- 7.5 Beta 版出版物文档版本.

0.1-2

周六 31 2017 年，Mark Flitter(mflitter@redhat.com)

- 发布 7.4 GA 的文件版本.

0.1-0

2017 年 4 月 20 日，Mark Flitter(mflitter@redhat.com)

- 初始构建供审核