Red Hat Enterprise Linux 9

커널 관리, 모니터링 및 업데이트

Red Hat Enterprise Linux 9에서 Linux 커널 관리 가이드

Last Updated: 2024-07-18

Red Hat Enterprise Linux 9 커널 관리, 모니터링 및 업데이트

Red Hat Enterprise Linux 9에서 Linux 커널 관리 가이드

법적 공지

초록

시스템 관리자는 운영 체제를 최적화하도록 Linux 커널을 구성할 수 있습니다. Linux 커널을 변경하면 시스템 성능, 보안 및 안정성이 향상되고 시스템을 감사하고 문제를 해결할 수 있습니다.

차례

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

Jira를 통해 피드백 제출 (등록 필요)

1. Jira 웹 사이트에 로그인합니다.
2. 상단 탐색 모음에서 **생성** 을 클릭합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. 대화 상자 하단에서 **생성** 을 클릭합니다.

1장. LINUX 커널

Red Hat (Red Hat 커널)에서 제공하고 유지 관리하는 Linux 커널 및 Linux 커널 RPM 패키지에 대해 알아보십시오. 운영 체제에 모든 최신 버그 수정, 성능 향상 및 패치가 보장되고 새 하드웨어와 호환되는 Red Hat 커널을 계속 업데이트합니다.

1.1. 커널의 정의

커널은 시스템 리소스를 관리하고 하드웨어 및 소프트웨어 애플리케이션 간의 인터페이스를 제공하는 Linux 운영 체제의 핵심 부분입니다.

Red Hat 커널은 Red Hat 엔지니어가 최신 기술 및 하드웨어와의 안정성과 호환성에 중점을 두고 개발 및 강화되는 업스트림 Linux 메인 라인 커널을 기반으로 하는 맞춤형 커널입니다.

Red Hat이 새 커널 버전을 출시하기 전에 커널은 엄격한 품질 보증 테스트를 통과해야 합니다.

Red Hat 커널은 **DNF** 패키지 관리자가 쉽게 업그레이드 및 확인할 수 있도록 RPM 형식으로 패키지됩니다.

주의

Red Hat에서 컴파일하지 않은 커널은 Red Hat에서 **지원하지** 않습니다.

1.2. RPM 패키지

RPM 패키지는 이러한 파일을 설치 및 삭제하는 데 사용되는 파일 및 메타데이터의 아카이브로 구성됩니다. 특히 RPM 패키지에는 다음 부분이 포함되어 있습니다.

GPG 서명

GPG 서명은 패키지의 무결성을 확인하는 데 사용됩니다.

헤더(패키지 메타데이터)

RPM 패키지 관리자는 이 메타데이터를 사용하여 패키지 종속성, 파일 설치 위치 및 기타 정보를 확인합니다.

페이로드

페이로드는 시스템에 설치할 파일이 포함된 **cpio** 아카이브입니다.

RPM 패키지에는 두 가지 유형이 있습니다. 두 유형 모두 파일 형식과 툴링을 공유하지만 콘텐츠가 다르며 다른 용도로 사용됩니다.

- 소스 RPM(SRPM)
  SRPM에는 소스 코드와 **사양** 파일이 포함되어 있으며, 바이너리 RPM에 소스 코드를 빌드하는 방법을 설명합니다. 선택적으로 SRPM은 소스 코드에 대한 패치를 포함할 수 있습니다.

  바이너리 RPM

  바이너리 RPM에는 소스 및 패치에서 빌드된 바이너리가 포함되어 있습니다.

1.3. LINUX 커널 RPM 패키지 개요

커널 RPM은 파일을 포함하지 않는 메타 패키지이지만 다음과 같은 필수 하위 패키지가 올바르게 설치되도록 합니다.

kernel-core

Linux 커널의 바이너리 이미지(**vmlinuz**)를 포함합니다.

kernel-modules-core

핵심 기능을 보장하기 위한 기본 커널 모듈이 포함되어 있습니다. 여기에는 가장 일반적으로 사용되는 하드웨어의 적절한 기능에 필요한 모듈이 포함됩니다.

kernel-modules

kernel-core 에 없는 나머지 커널 모듈이 포함되어 있습니다.

커널-core 및 **kernel-modules-core** 하위 패키지는 함께 사용하여 RHEL 9 커널에 빠른 부팅 시간과 작은 디스크 크기의 공간을 제공하는 데 사용할 수 있습니다. **kernel-modules** 하위 패키지는 일반적으로 이러한 배포에 필요하지 않습니다.

선택적 커널 패키지는 다음과 같습니다.

kernel-modules-extra

드문 하드웨어 및 로드가 기본적으로 비활성화되는 모듈의 커널 모듈이 포함되어 있습니다.

kernel-debug

성능 저하를 통해 커널 진단을 위해 다양한 디버깅 옵션을 사용할 수 있는 커널이 포함되어 있습니다.

kernel-tools

Linux 커널 및 지원 문서를 조작하는 도구가 포함되어 있습니다.

kernel-devel

커널 패키지에 대해 모듈을 구축하기에 충분한 **커널** 헤더 및 makefiles를 포함합니다.

kernel-abi-stablelists

외부 Linux 커널 모듈에 필요한 커널 기호 목록과 시행을 돕기 위해 **dnf** 플러그인 등 RHEL 커널 ABI 관련 정보를 포함합니다.

kernel-headers

Linux 커널과 사용자 공간 라이브러리 및 프로그램 간의 인터페이스를 지정하는 C 헤더 파일이 포함되어 있습니다. 헤더 파일은 대부분의 표준 프로그램을 빌드하는 데 필요한 구조 및 상수를 정의합니다.

kernel-uki-virt

RHEL 커널의 통합 커널 이미지(UKI)를 포함합니다.
UKI는 Linux 커널, **initramfs** 및 커널 명령줄을 UEFI 펌웨어에서 직접 부팅할 수 있는 하나의 서명된 바이너리에 결합합니다.

kernel-uki-virt 에는 가상화 및 클라우드 환경에서 실행하는 데 필요한 커널 모듈이 포함되어 있으며 **kernel-core** 하위 패키지 대신 사용할 수 있습니다.

중요

kernel-uki-virt 는 RHEL 9.2에서 기술 프리뷰로 제공됩니다.

추가 리소스

- *kernel-core, kernel-modules, kernel-modules-extras 패키지는 무엇입니까?*

1.4. 커널 패키지의 콘텐츠 표시

커널 패키지가 모듈과 같은 특정 파일을 제공하는지 확인하려면 리포지토리를 쿼리하여 아키텍처의 파일 목록을 표시할 수 있습니다. 파일 목록을 표시하기 위해 패키지를 다운로드하거나 설치할 필요는 없습니다.

dnf 유틸리티를 사용하여 **kernel-core**,**kernel-modules-core** 또는 **kernel-modules** 패키지의 파일 목록을 쿼리합니다. **커널** 패키지는 파일이 포함되어 있지 않은 메타 패키지입니다.

절차

1. 사용 가능한 패키지 버전을 나열합니다.

```
$ dnf repoquery <package_name>
```

예를 들어 사용 가능한 **kernel-core** 패키지 버전을 나열합니다.

```
$ dnf repoquery kernel-core
kernel-core-0:5.14.0-162.12.1.el9_1.x86_64
kernel-core-0:5.14.0-162.18.1.el9_1.x86_64
kernel-core-0:5.14.0-162.22.2.el9_1.x86_64
kernel-core-0:5.14.0-162.23.1.el9_1.x86_64
...
```

2. 패키지에 있는 파일 목록을 표시합니다.

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

예를 들어 **kernel-core-0:5.14.0-162.23.1.el9_1.x86_64** 패키지의 파일 목록을 표시합니다.

```
$ dnf repoquery -l kernel-core-0:5.14.0-162.23.1.el9_1.x86_64
/boot/System.map-5.14.0-162.23.1.el9_1.x86_64
/boot/config-5.14.0-162.23.1.el9_1.x86_64
/boot/initramfs-5.14.0-162.23.1.el9_1.x86_64.img
/boot/symvers-5.14.0-162.23.1.el9_1.x86_64.gz
/boot/vmlinuz-5.14.0-162.23.1.el9_1.x86_64
/lib/modules
/lib/modules/5.14.0-162.23.1.el9_1.x86_64
/lib/modules/5.14.0-162.23.1.el9_1.x86_64/.vmlinuz.hmac
/lib/modules/5.14.0-162.23.1.el9_1.x86_64/System.map
...
```

추가 리소스

- 소프트웨어 패키지 및 배포

1.5. 특정 커널 버전 설치

dnf 패키지 관리자를 사용하여 새 커널을 설치합니다.

절차

- 특정 커널 버전을 설치하려면 다음 명령을 입력합니다.

```
# dnf install kernel-{version}
```

추가 리소스

- *Red Hat 코드 브라우저*
- *Red Hat Enterprise Linux Release Dates*

1.6. 커널 업데이트

dnf 패키지 관리자를 사용하여 커널을 업데이트합니다.

절차

1. 커널을 업데이트하려면 다음 명령을 입력합니다.

```
# dnf update kernel
```

이 명령은 사용 가능한 최신 버전으로 모든 종속 항목과 함께 커널을 업데이트합니다.

2. 변경 사항을 적용하려면 시스템을 재부팅합니다.

추가 리소스

- 패키지 관리자
- **dnf(8)** 매뉴얼 페이지

1.7. 커널 설정을 기본값으로 설정

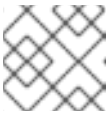
grubby 명령줄 도구 및 GRUB을 사용하여 특정 커널을 기본값으로 설정합니다.

절차

- **grubby** 툴을 사용하여 커널을 기본값으로 설정
  - 다음 명령을 입력하여 **grubby** 툴을 사용하여 커널을 기본값으로 설정합니다.

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

이 명령은 **.conf** 접미사가 없는 머신 ID를 인수로 사용합니다.

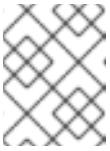


참고

시스템 ID는 **/boot/loader/entries/** 디렉터리에 있습니다.

- **id** 인수를 사용하여 커널을 기본값으로 설정
  - **id** 인수를 사용하여 부팅 항목을 나열한 다음 원하는 커널을 기본값으로 설정합니다.

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



참고

title 인수를 사용하여 부팅 항목을 나열하려면 **# grubby --info=ALL | grep title** 명령을 실행합니다.

- 다음 부팅에만 기본 커널 설정
  - **grub2-reboot** 명령을 사용하여 다음 재부팅에 대해서만 기본 커널을 설정하려면 다음 명령을 실행합니다.

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

주의

다음 부팅만 주의해서 사용하도록 기본 커널을 설정합니다. 새 커널 RPM의 자체 내장 커널을 설치하고 **/boot/loader/entries/** 디렉터리에 항목을 수동으로 추가하면 인덱스 값이 변경될 수 있습니다.

2장. 64K 페이지 크기 커널

kernel-64k 는 64k 페이지를 지원하는 추가 선택적 64비트 ARM 아키텍처 커널 패키지입니다. 이 추가 커널은 4k 페이지를 지원하는 ARM 커널의 RHEL 9와 함께 존재합니다.

최적의 시스템 성능은 다양한 메모리 구성 요구 사항과 직접 관련이 있습니다. 이러한 요구 사항은 각각 다른 워크로드에 적합한 두 가지 커널 변형으로 해결됩니다. 64비트 ARM 하드웨어에서 RHEL 9는 두 개의 MMU 페이지 크기를 제공합니다.

- 작은 환경에서 효율적인 메모리 사용을 위한 4K 페이지 커널,
- **kernel-64k**: 연속된 대규모 메모리 작업 세트가 있는 워크로드용입니다.

4k 페이지 커널과 **kernel-64k** 는 사용자 공간이 동일하기 때문에 사용자 경험이 다르지 않습니다. 상황을 가장 잘 다루는 변형을 선택할 수 있습니다.

4K 페이지 커널

소규모 환경에서 4k 페이지를 사용하여 Edge 및 저비용의 작은 클라우드 인스턴스의 메모리 사용량을 보다 효율적으로 사용할 수 있습니다. 이러한 환경에서는 물리적 시스템 메모리 양을 늘리는 것은 공간, 전력 및 비용 제약으로 인해 실용적이지 않습니다. 또한 일부 64비트 ARM 아키텍처 프로세서는 64k 페이지 크기를 지원하는 것은 아닙니다.
4k 페이지 커널은 Anaconda, 시스템 또는 클라우드 이미지 기반 설치와 Kickstart를 사용하여 고급 설치를 사용하여 그래픽 설치를 지원합니다.

kernel-64k

64k 페이지 크기 커널은 ARM 플랫폼의 대규모 SSDT에 유용한 옵션입니다. **kernel-64k** 는 전체 시스템 성능, 즉 대용량 데이터베이스, HPC 및 네트워크 성능이 크게 향상되었기 때문에 메모리 집약적 워크로드에 적합합니다.
설치 시 64비트 ARM 아키텍처 시스템에서 페이지 크기를 선택해야 합니다. **Kickstart** 파일의 패키지 목록에 **kernel-64k** 패키지를 추가하여 Kickstart를 통해서만 **kernel-64k** 를 설치할 수 있습니다.

추가 리소스

- ARM에 Kernel-64k 설치

3장. 커널 모듈 관리

커널 모듈, 정보를 표시하는 방법, 커널 모듈을 사용하여 기본 관리 작업을 수행하는 방법을 알아봅니다.

3.1. 커널 모듈 소개

Red Hat Enterprise Linux 커널은 시스템을 재부팅하지 않고도 커널 모듈이라는 선택적 추가 기능을 사용하여 확장할 수 있습니다. Red Hat Enterprise Linux 9에서 커널 모듈은 압축된 < **KERNEL_MODULE_NAME>.ko.xz** 오브젝트 파일에 내장된 추가 커널 코드입니다.

커널 모듈에서 사용하는 가장 일반적인 기능은 다음과 같습니다.

- 새 하드웨어 지원을 추가하는 장치 드라이버
- polkit2 또는 NFS와 같은 파일 시스템 지원
- 시스템 호출

최신 시스템에서는 필요에 따라 커널 모듈이 자동으로 로드됩니다. 그러나 경우에 따라 모듈을 수동으로 로드하거나 언로드해야 하는 경우가 있습니다.

커널 자체와 마찬가지로 모듈은 필요한 경우 동작을 사용자 지정하는 매개변수를 사용할 수 있습니다.

현재 실행 중인 모듈, 커널에 로드할 수 있는 모듈과 모듈에서 허용하는 매개 변수를 검사하기 위해 툴이 제공됩니다. 툴링은 커널 모듈을 실행 중인 커널에 로드하고 언로드하는 메커니즘을 제공합니다.

3.2. 커널 모듈 종속 항목

특정 커널 모듈은 하나 이상의 다른 커널 모듈에 따라 다를 수 있습니다. **/lib/modules/<KERNEL_VERSION>/modules.dep** 파일에는 해당 커널 버전에 대한 커널 모듈 종속 항목의 전체 목록이 포함되어 있습니다.

depmod

종속성 파일은 **kmod** 패키지의 일부인 **depmod** 프로그램에 의해 생성됩니다. **kmod** 에서 제공하는 많은 유틸리티가 작업을 수행할 때 모듈 종속성을 고려해야 하므로 **수동** 종속성 추적이 거의 필요하지 않습니다.

주의

커널 모듈의 코드는 무제한 모드에서 커널 스페이스에서 실행됩니다. 이 때문에 로드 중인 모듈을 고려해야 합니다.

약한 모듈

Red Hat Enterprise Linux는 **depmod** 뿐만 아니라 **kmod** 패키지와 함께 제공되는 **약한 모듈** 스크립트를 제공합니다. **약한 모듈은** 설치된 커널과 kABI와 호환되는 모듈을 결정합니다. 모듈 커널 호환성을 확인하는 동안 **약한 모듈** 처리는 더 높음부터 더 낮은 커널 릴리스까지 기호를 처리합니다. 즉, **약한 모듈이** 빌드된 커널 릴리스와 관계없이 각 모듈을 처리합니다.

추가 리소스

- **modules.dep(5)** 매뉴얼 페이지
- **depmod(8)** 매뉴얼 페이지
- Red Hat Enterprise Linux에서 제공되는 약한 모듈 스크립트의 목적은 무엇입니까?
- Kernel Application Binary Interface (kABI)란 무엇인가?

3.3. 설치된 커널 모듈 나열

grubby --info=ALL 명령은 **!BLS** 및 **BLS** 설치에 설치된 커널의 인덱싱된 목록을 표시합니다.

절차

- 다음 명령을 사용하여 설치된 커널을 나열합니다.

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

설치된 모든 커널 목록은 다음과 같이 표시됩니다.

```
title="Red Hat Enterprise Linux (5.14.0-1.el9.x86_64) 9.0 (Plow)"
title="Red Hat Enterprise Linux (0-rescue-0d772916a9724907a5d1350bcd39ac92) 9.0
(Plow)"
```

위 예제는 GRUB 메뉴에 설치된 kernels list of grubby-8.40-17을 표시합니다.

3.4. 현재 로드된 커널 모듈 나열

현재 로드된 커널 모듈을 확인합니다.

사전 요구 사항

- **kmod** 패키지가 설치되어 있습니다.

절차

- 현재 로드된 모든 커널 모듈을 나열하려면 다음을 입력합니다.

```
$ 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
…
```

위의 예에서 다음을 수행합니다.

a. **Module** 열은 현재 로드된 모듈의 **이름을** 제공합니다.

b. **Size** 열에는 모듈당 **메모리** 양이 킬로바이트로 표시됩니다.

c. **Used by** 열에는 특정 모듈에 **종속되** 는 모듈의 이름과 숫자(선택 사항)가 표시됩니다.

추가 리소스

- **/usr/share/doc/kmod/README** 파일
- **lsmod(8)** 매뉴얼 페이지

3.5. 커널 모듈에 대한 정보 표시

modinfo 명령을 사용하여 지정된 커널 모듈에 대한 세부 정보를 표시합니다.

사전 요구 사항

- **kmod** 패키지가 설치되어 있습니다.

절차

- 커널 모듈에 대한 정보를 표시하려면 다음을 입력합니다.

```
$ modinfo <KERNEL_MODULE_NAME>
```

예를 들면 다음과 같습니다.

```
$ modinfo virtio_net

filename:       /lib/modules/5.14.0-1.el9.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:        GPL
description:    Virtio network driver
rhelversion:    9.0
srcversion:     8809CDDBE7202A1B00B9F1C
alias:          virtio:d00000001v*
depends:        net_failover
retpoline:      Y
intree:         Y
name:           virtio_net
vermagic:       5.14.0-1.el9.x86_64 SMP mod_unload modversions
…
parm:           napi_weight:int
parm:           csum:bool
parm:           gso:bool
parm:           napi_tx:bool
```

로드되었는지 여부에 관계없이 사용 가능한 모든 모듈에 대한 정보를 쿼리할 수 있습니다. **parm** 항목은 사용자가 모듈에 설정할 수 있는 매개변수와 예상 값 유형을 보여줍니다.

참고

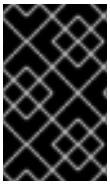
커널 모듈의 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장자가 없습니다. 해당 파일에는 해당 파일이 있습니다.

추가 리소스

- **modinfo(8)** 매뉴얼 페이지

3.6. 시스템 런타임에서 커널 모듈 로드

Linux 커널의 기능을 확장하는 최적의 방법은 커널 모듈을 로드하는 것입니다. **modprobe** 명령을 사용하여 커널 모듈을 찾고 현재 실행 중인 커널에 로드합니다.



중요

이 절차에 설명된 변경 사항은 시스템을 재부팅한 후에도 **유지되지 않습니다**. 시스템 재부팅 시 **지속** 되도록 커널 모듈을 로드하는 방법에 대한 자세한 내용은 시스템 부팅 시 커널 모듈 로드 를 참조하십시오.

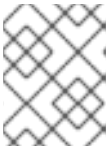
사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치되어 있습니다.
- 각 커널 모듈이 로드되지 않습니다. 이 경우 로드된 커널 모듈을 나열하십시오.

절차

1. 로드할 커널 모듈을 선택합니다.
   모듈은 **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** 디렉터리에 있습니다.
2. 관련 커널 모듈을 로드합니다.

```
# modprobe <MODULE_NAME>
```



참고

커널 모듈의 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장자가 없습니다. 해당 파일에는 해당 파일이 있습니다.

검증

- 선택적으로 관련 모듈이 로드되었는지 확인합니다.

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

모듈이 올바르게 로드되면 이 명령은 관련 커널 모듈을 표시합니다. 예를 들면 다음과 같습니다.

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

추가 리소스

- **modprobe(8)** 매뉴얼 페이지

3.7. 시스템 런타임 시 커널 모듈 언로드

경우에 따라 실행 중인 커널에서 특정 커널 모듈을 언로드해야 합니다. **modprobe** 명령을 사용하여 현재 로드된 커널에서 시스템 런타임 시 커널 모듈을 찾고 언로드합니다.

주의

실행 중인 시스템에서 커널 모듈을 사용할 때 언로드하지 마십시오. 이렇게 하면 불안정하거나 작동하지 않는 시스템이 발생할 수 있습니다.

중요

이 절차를 마친 후 부팅 시 자동으로 로드되도록 정의된 커널 모듈은 시스템을 재부팅한 후 **언로드되지 않습니다**. 이 결과에 대응하는 방법에 대한 자세한 내용은 시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록 방지를 참조하십시오.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치되어 있습니다.

절차

1. 로드된 모든 커널 모듈을 나열합니다.

```
# lsmod
```

2. 언로드할 커널 모듈을 선택합니다.
   커널 모듈에 종속성이 있는 경우 커널 모듈을 언로드하기 전에 해당 모듈을 언로드합니다. 종속성이 있는 모듈 식별에 대한 자세한 내용은 현재 로드된 커널 모듈 및 커널 모듈 종속성 목록을 참조하십시오.

3. 관련 커널 모듈을 언로드합니다.

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

   커널 모듈의 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장자가 없습니다. 해당 파일에는 해당 파일이 있습니다.

검증

- 선택적으로 관련 모듈이 언로드되었는지 확인합니다.

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

모듈이 성공적으로 언로드된 경우 이 명령은 출력을 표시하지 않습니다.

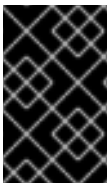
추가 리소스

- **modprobe(8)** 매뉴얼 페이지

3.8. 부팅 프로세스의 초기 단계에서 커널 모듈 언로드 해제

특정 상황에서는 부팅 프로세스 초기에 커널 모듈을 언로드해야 합니다. 예를 들어, kernel 모듈에 시스템이 응답하지 않게 하는 코드가 포함되어 있고 사용자가 악성 커널 모듈을 영구적으로 비활성화하기 위해 단계에 도달할 수 없습니다. 이 경우 부트 로더를 사용하여 커널 모듈의 로드를 일시적으로 차단할 수 있습니다.

관련 부트 로더 항목을 편집하여 부팅 시퀀스가 계속하기 전에 원하는 커널 모듈을 언로드할 수 있습니다.



중요

이 절차에 설명된 변경 사항은 다음 재부팅 **후에도 유지되지 않습니다**. 부팅 프로세스 중에 자동으로 로드되지 않도록 커널 모듈을 거부 목록에 추가하는 방법에 대한 자세한 내용은 시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록 를 참조하십시오.

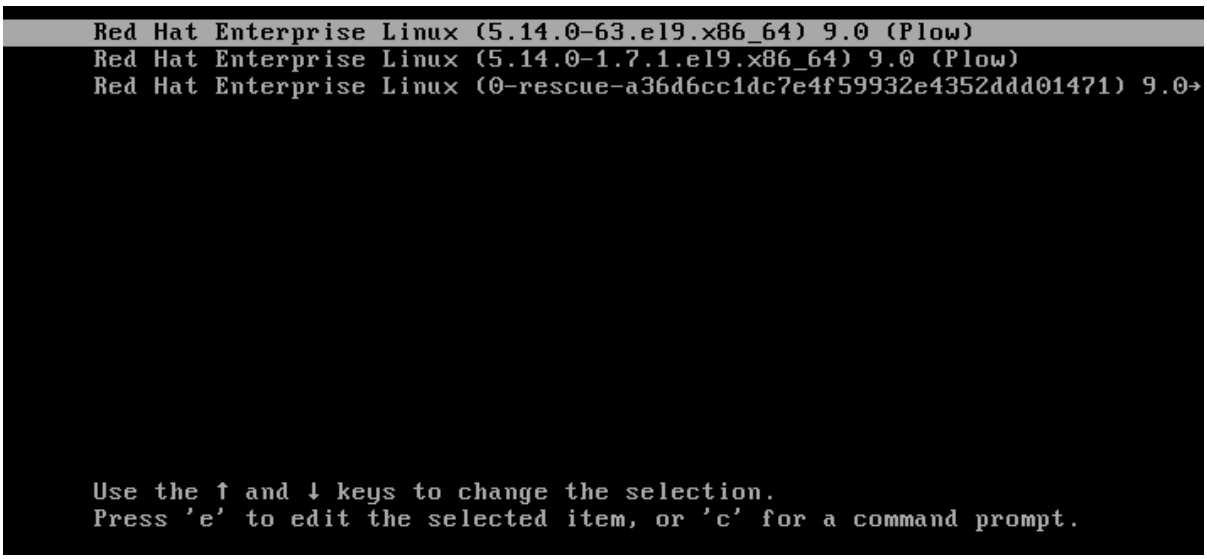
사전 요구 사항

- 어떤 이유로 로드되지 않도록하려는 로드 가능한 커널 모듈이 있습니다.

절차

1. 시스템을 부트 로더로 부팅합니다.
2. 커서 키를 사용하여 관련 부트 로더 항목을 강조 표시합니다.
3. **e** 키를 눌러 항목을 편집합니다.

그림 3.1. 커널 부팅 메뉴



4. 커서 키를 사용하여 **linux** 로 시작하는 행으로 이동합니다.
5. Append **modprobe.blacklist=*module_name*** 을 행 끝에 추가합니다.

그림 3.2. 커널 부팅 항목

serio_raw 커널 모듈은 부팅 프로세스 초기에 로드를 해제하는 악성 모듈을 보여줍니다.

6. **Ctrl+X** 눌러 수정된 구성을 사용하여 부팅합니다.

검증

- 시스템이 완전히 부팅되면 관련 커널 모듈이 로드되지 않았는지 확인합니다.

```
# lsmod | grep serio_raw
```

추가 리소스

- 커널 모듈 관리

3.9. 시스템 부팅 시 커널 모듈을 자동으로 로드

부팅 프로세스 중에 자동으로 로드되도록 커널 모듈을 구성합니다.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치되어 있습니다.

절차

1. 부팅 프로세스 중에 로드할 커널 모듈을 선택합니다.
   모듈은 **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** 디렉터리에 있습니다.
2. 모듈에 대한 구성 파일을 생성합니다.

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

참고

커널 모듈의 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장자가 없습니다. 해당 파일에는 해당 파일이 있습니다.

3. 필요한 경우 재부팅 후 관련 모듈이 로드되었는지 확인합니다.

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

위의 예제 명령은 성공하고 관련 커널 모듈을 표시합니다.

중요

이 절차에 설명된 변경 사항은 시스템을 재부팅한 후에도 **지속됩니다**.

추가 리소스

- **modules-load.d(5)** 매뉴얼 페이지

3.10. 시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록 방지

modprobe 구성 파일에 해당 명령을 사용하여 모듈을 나열하여 부팅 프로세스 중에 시스템이 커널 모듈을 자동으로 로드하지 못하도록 할 수 있습니다.

사전 요구 사항

- 이 절차의 명령에는 root 권한이 필요합니다. **su -** 를 사용하여 root 사용자로 전환하거나 **sudo** 를 사용하여 명령 앞에 추가합니다.
- **kmod** 패키지가 설치되어 있습니다.
- 현재 시스템 구성에 거부하려는 커널 모듈이 필요하지 않은지 확인합니다.

절차

1. **lsmod** 명령을 사용하여 현재 실행 중인 커널에 로드된 모듈을 나열합니다.

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

출력에서 로드되지 않도록 할 모듈을 식별합니다.

- 또는 **/lib/modules/ <KERNEL-VERSION> /kernel/ <SUBSYSTEM> / 디렉터리에 잠재적으로 로드되지 않도록하려는 언로드된 커널** 모듈을 식별합니다. 예를 들면 다음과 같습니다.

```
$ 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. 거부 목록 역할을 하는 구성 파일을 생성합니다.

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

3. 선택한 텍스트 편집기에서 **blacklist** 설정 명령과 함께 커널 자동 로드에서 제외할 모듈 이름을 결합합니다.

```
# 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
…
```

blacklist 명령은 모듈이 거부 목록에 없는 다른 커널 모듈의 종속성으로 로드되지 않도록 하려면 **설치** 행도 정의해야 합니다. 이 경우 시스템은 모듈을 설치하는 대신 **/bin/false** 를 실행합니다. 해시 기호로 시작하는 행은 파일을 더 읽기 쉽게 만드는 데 사용할 수 있는 주석입니다.

참고

커널 모듈의 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장자가 없습니다. 해당 파일에는 해당 파일이 있습니다.

4. 다시 빌드하기 전에 현재 초기 RAM 디스크 이미지의 백업 사본을 생성합니다.

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

- 또는 커널 모듈이 자동 로드되지 않도록하려는 커널 버전에 해당하는 초기 RAM 디스크 이미지의 백업 사본을 생성합니다.

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

5. 새 초기 RAM 디스크 이미지를 생성하여 변경 사항을 적용합니다.

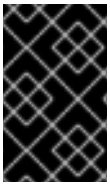
```
# dracut -f -v
```

- 현재 시스템과 다른 커널 버전의 초기 RAM 디스크 이미지를 빌드하는 경우 target **initramfs** 및 커널 버전을 모두 지정합니다.

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

6. 시스템을 다시 시작하십시오.

```
$ reboot
```



중요

이 절차에 설명된 변경 사항은 시스템을 재부팅한 후에도 **적용되며 지속됩니다**. 거부 목록에 키 커널 모듈을 잘못 나열하는 경우 시스템을 불안정하거나 작동하지 않는 상태로 전환할 수 있습니다.

추가 리소스

- 커널 모듈이 자동으로 로드되지 않도록 하려면 어떻게 해야 합니까? 솔루션 문서
- **modprobe.d(5)** 및 **dracut(8)** 도움말 페이지

3.11. 사용자 정의 커널 모듈 컴파일

하드웨어 및 소프트웨어 수준에서 다양한 구성에서 요청한 대로 샘플링 커널 모듈을 빌드할 수 있습니다.

사전 요구 사항

- **kernel-devel**,**gcc** 및 **elfutils-libelf-devel** 패키지를 설치했습니다.

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

- 루트 권한이 있습니다.
- 사용자 지정 커널 모듈을 컴파일하는 **/root/testmodule/** 디렉터리를 생성하셨습니다.

절차

1. 다음 콘텐츠를 사용하여 **/root/testmodule/test.c** 파일을 만듭니다.

```
#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"); }

MODULE_LICENSE("GPL");
```

test.c 파일은 커널 모듈에 기본 기능을 제공하는 소스 파일입니다. 파일은 조직 용도로 전용 **/root/testmodule/** 디렉토리에 생성되었습니다. 모듈 컴파일 후 **/root/testmodule/** 디렉토리에 여러 파일이 포함됩니다.

test.c 파일은 시스템 라이브러리에서 포함합니다.

- 예제 코드의 **printk()** 함수에는 **linux/kernel.h** 헤더 파일이 필요합니다.
- **linux/module.h** 파일에는 C 프로그래밍 언어로 작성된 여러 소스 파일 간에 공유할 함수 선언과 매크로 정의가 포함되어 있습니다.

2. **init_module()** 및 **cleanup_module()** 함수를 따라 텍스트를 출력하는 커널 로깅 함수 **printk()** 를 시작하고 종료합니다.

3. 다음 콘텐츠를 사용하여 **/root/testmodule/Makefile** 파일을 만듭니다.

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

Makefile에는 컴파일러에서 특별히 **test.o** 라는 개체 파일을 생성해야 하는 명령이 포함되어 있습니다. **obj-m** 지시문은 결과 **test.ko** 파일이 로드 가능한 커널 모듈로 컴파일되도록 지정합니다. 또는 **obj-y** 지시문은 **test.ko** 를 기본 제공 커널 모듈로 빌드하도록 지시합니다.

4. 커널 모듈을 컴파일합니다.

```
# make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
make: Entering directory '/usr/src/kernels/5.14.0-70.17.1.el9_0.x86_64'
  CC [M]  /root/testmodule/test.o
  MODPOST /root/testmodule/Module.symvers
  CC [M]  /root/testmodule/test.mod.o
  LD [M]  /root/testmodule/test.ko
  BTF [M] /root/testmodule/test.ko
Skipping BTF generation for /root/testmodule/test.ko due to unavailability of vmlinux
make: Leaving directory '/usr/src/kernels/5.14.0-70.17.1.el9_0.x86_64'
```

컴파일러는 각 소스 파일(**test.c**)에 대해 최종 커널 모듈(**test.ko**)에 연결하기 전에 중간 단계로 오브젝트 파일(**test.o**)을 생성합니다.

컴파일에 성공한 후 **/root/testmodule/** 에는 컴파일된 사용자 지정 커널 모듈과 관련된 추가 파일이 포함되어 있습니다. 컴파일된 모듈 자체는 **test.ko** 파일로 표시됩니다.

검증

1. 선택 사항: **/root/testmodule/** 디렉토리의 내용을 확인합니다.

```
# 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. kernel 모듈을 **/lib/modules/$(uname -r)/** 디렉터리에 복사합니다.

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

3. 모듈식 종속성 목록을 업데이트합니다.

```
# depmod -a
```

4. 커널 모듈을 로드합니다.

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

5. 커널 모듈이 성공적으로 로드되었는지 확인합니다.

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

6. 커널 링 버퍼에서 최신 메시지를 읽습니다.

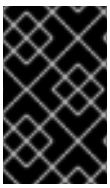
```
# dmesg
[74422.545004] Hello World
               This is a test
```

추가 리소스

- 커널 모듈 관리

4장. 커널 명령줄 매개변수 구성

커널 명령줄 매개 변수를 사용하면 부팅 시 Red Hat Enterprise Linux 커널의 특정 측면의 동작을 변경할 수 있습니다. 시스템 관리자는 부팅 시 설정하는 옵션을 완전히 제어할 수 있습니다. 특정 커널 동작은 부팅 시에만 설정할 수 있으므로 이러한 변경을 수행하는 방법을 이해하는 것이 핵심 관리 기술입니다.



중요

커널 명령줄 매개 변수를 수정하여 시스템 동작을 변경하면 시스템에 부정적인 영향을 미칠 수 있습니다. 프로덕션 환경에 배포하기 전에 항상 변경 사항을 테스트합니다. 자세한 내용은 Red Hat 지원팀에 문의하십시오.

4.1. 커널 명령줄 매개변수란 무엇입니까?

커널 명령줄 매개 변수를 사용하면 기본값을 덮어쓰고 특정 하드웨어 설정을 설정할 수 있습니다. 부팅 시 다음 기능을 구성할 수 있습니다.

- Red Hat Enterprise Linux 커널
- 초기 RAM 디스크
- 사용자 공간 기능

기본적으로 GRUB 부트 로더를 사용하는 시스템의 커널 명령줄 매개 변수는 각 커널 부팅 항목의 부팅 항목 구성 파일에 정의됩니다.

grubby 유틸리티를 사용하여 부트 로더 구성 파일을 조작할 수 있습니다. **grubby** 를 사용하면 다음 작업을 수행할 수 있습니다.

- 기본 부팅 항목을 변경합니다.
- GRUB 메뉴 항목에서 인수를 추가하거나 제거합니다.

추가 리소스

- **kernel-command-line(7)**, **bootparam(7)** 및 **dracut.cmdline(7)** 매뉴얼 페이지
- Red Hat Enterprise Linux 8에서 사용자 정의 커널을 설치 및 부팅하는 방법
- **grubby(8)** 매뉴얼 페이지

4.2. 부팅 항목 이해

부팅 항목은 구성 파일에 저장되고 특정 커널 버전에 연결된 옵션의 컬렉션입니다. 실제로는 시스템에 커널을 설치한 최소 개수의 부팅 항목이 있습니다. 부팅 항목 구성 파일은 **/boot/loader/entries/** 디렉터리에 있으며 다음과 같을 수 있습니다.

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

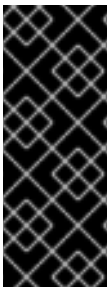
위의 파일 이름은 **/etc/machine-id** 파일 및 커널 버전에 저장된 시스템 ID로 구성됩니다.

부팅 항목 구성 파일에는 커널 버전, 초기 램디스크 이미지 및 커널 명령줄 매개변수에 대한 정보가 포함되어 있습니다. 부팅 항목 구성의 예제 내용은 다음과 같습니다.

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

4.3. 모든 부팅 항목에 대한 커널 명령줄 매개 변수 변경

시스템의 모든 부팅 항목에 대한 커널 명령행 매개변수 변경



중요

RHEL 9 시스템에서 최신 버전의 커널을 설치할 때 **grubby** 툴은 이전 커널 버전의 커널 명령줄 인수를 전달합니다.

그러나 새로 설치된 커널에서 이전 명령줄 옵션이 손실되는 RHEL 버전 9.0에는 적용되지 않습니다. 매개변수를 새 커널에 전달하려면 새로 설치된 커널에서 **grub2-mkconfig** 명령을 실행해야 합니다. 이 알려진 문제에 대한 자세한 내용은 Boot loader 를 참조하십시오.

사전 요구 사항

- **grubby** 유틸리티가 시스템에 설치되어 있는지 확인합니다.
- **zipl** 유틸리티가 IBM Z 시스템에 설치되어 있는지 확인합니다.

절차

- 매개변수를 추가하려면 다음을 수행합니다.

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

  GRUB 부트 로더를 사용하고 IBM Z에서 zIPL 부트 로더를 사용하는 시스템의 경우 명령은 각 **/boot/loader/entries/<*ENTRY*>.conf** 파일에 새 커널 매개변수를 추가합니다.

  - IBM Z에서 부팅 메뉴를 업데이트합니다.

    ```
    # zipl
    ```

- 매개변수를 제거하려면 다음을 수행합니다.

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

  - IBM Z에서 부팅 메뉴를 업데이트합니다.

    ```
    # zipl
    ```

추가 리소스

- 커널 명령줄 매개변수란 무엇입니까?
- **grubby(8)** 및 **zipl(8)** 매뉴얼 페이지
- *grubby 도구*

4.4. 단일 부팅 항목에 대한 커널 명령줄 매개변수 변경

시스템의 단일 부팅 항목에 대한 커널 명령줄 매개변수를 변경합니다.

사전 요구 사항

- **grubby** 및 **zipl** 유틸리티가 시스템에 설치되어 있는지 확인합니다.

절차

- 매개변수를 추가하려면 다음을 수행합니다.

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

  - IBM Z에서 부팅 메뉴를 업데이트합니다.

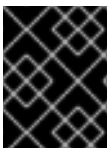
```
# zipl
```

- 매개변수를 제거하려면 다음을 수행합니다.

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

  - IBM Z에서 부팅 메뉴를 업데이트합니다.

```
# zipl
```



중요

- **grubby** 는 **/boot/loader/entries/<*ENTRY*>.conf** 파일에 개별 커널 부팅 항목의 커널 명령줄 매개 변수를 수정하고 저장합니다.

추가 리소스

- 커널 명령줄 매개변수란 무엇입니까?
- **grubby(8)** 및 **zipl(8)** 매뉴얼 페이지
- *grubby 도구*

4.5. 부팅 시 일시적으로 커널 명령줄 매개 변수 변경

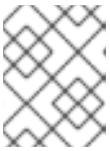
부팅 프로세스 중에만 커널 매개변수를 변경하여 커널 메뉴 항목을 일시적으로 변경합니다.

참고

이 절차는 단일 부팅에만 적용되며 변경 사항이 영구적으로 적용되지 않습니다.

절차

1. GRUB 2 부팅 메뉴로 부팅합니다.
2. 시작할 커널을 선택합니다.
3. **e** 키를 눌러 커널 매개 변수를 편집합니다.
4. 커서를 아래로 이동하여 커널 명령행을 찾습니다. 커널 명령행은 64비트 IBM Power Series 및 x86-64 BIOS 기반 시스템에서 **linux** 로 시작되거나 UEFI 시스템에서 **linuxefi** 로 시작됩니다.
5. 커서를 줄의 끝으로 이동합니다.



참고

Ctrl+**a** 눌러 줄의 시작 부분으로 건너뛰고 **Ctrl**+**e** 를 눌러 줄 끝으로 건너뜁니다. 일부 시스템에서는 **Home** 및 **End** 키도 작동할 수 있습니다.

6. 필요에 따라 커널 매개 변수를 편집합니다. 예를 들어 시스템을 긴급 모드에서 실행하려면 **linux** 행 끝에 **emergency** 매개변수를 추가합니다.

```
linux   ($root)/vmlinuz-5.14.0-63.el9.x86_64 root=/dev/mapper/rhel-root ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet emergency
```

   시스템 메시지를 활성화하려면 **rhgb** 및 **quiet** 매개변수를 제거합니다.

7. **Ctrl**+**x** 를 눌러 선택한 커널 및 수정된 명령행 매개 변수를 사용하여 부팅합니다.

중요

Esc 키를 눌러 명령줄 편집을 종료하면 사용자가 변경한 모든 내용이 삭제됩니다.

4.6. 직렬 콘솔 연결을 사용하도록 GRUB 설정 설정

직렬 콘솔은 헤드리스 서버 또는 임베디드 시스템에 연결해야 하고 네트워크가 다운된 경우 유용합니다. 또는 보안 규칙을 방지하고 다른 시스템에서 로그인 액세스 권한을 얻어야 합니다.

직렬 콘솔 연결을 사용하도록 몇 가지 기본 GRUB 설정을 구성해야 합니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. **/etc/default/grub** 파일에 다음 두 행을 추가합니다.

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

첫 번째 줄에서는 그래픽 터미널을 비활성화합니다. **GRUB_TERMINAL** 키는 **GRUB_TERMINAL_INPUT** 및 **GRUB_TERMINAL_OUTPUT** 키의 값을 재정의합니다.

두 번째 줄은 baud 속도(**--speed**), 패리티 및 기타 값을 사용자 환경과 하드웨어에 맞게 조정합니다. 예를 들어 115200과 같은 작업에는 훨씬 높은 세례 비율을 사용하는 것이 다음 로그 파일과 같은 작업에 적합합니다.

2. GRUB 설정 파일을 업데이트합니다.

   - BIOS 기반 시스템에서 다음을 수행합니다.

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

   - UEFI 기반 시스템에서 다음을 수행합니다.

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

3. 변경 사항을 적용하려면 시스템을 재부팅합니다.

4.7. GRUB 설정 파일을 사용하여 부팅 항목 변경

/etc/default/grub GRUB 설정 파일에는 Linux 커널의 부팅 항목에 추가할 커널 명령줄 인수를 나열하는 **GRUB_CMDLINE_LINUX** 키가 포함되어 있습니다. 예를 들면 다음과 같습니다.

```
GRUB_CMDLINE_LINUX="crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap"
```

부팅 항목을 변경하려면 **GRUB_CMDLINE_LINUX** 값의 내용으로 BLS(Boot Loader Specification) 스니펫을 덮어씁니다.

사전 요구 사항

- 새로운 RHEL 9 설치.

절차

1. **grubby** 를 사용하여 설치 후 스크립트에서 개별 커널의 커널 매개 변수를 추가하거나 제거합니다.

   ```
   # grubby --update-kernel <PATH_TO_KERNEL> --args "<NEW_ARGUMENTS>"
   ```

   예를 들어, 선택한 커널에 **noapic** 매개변수를 추가합니다.

   ```
   # grubby --update-kernel /boot/vmlinuz-5.14.0-362.8.1.el9_3.x86_64 --args "noapic"
   ```

   매개변수는 BLS 스니펫으로 전파되지만 **/etc/default/grub** 파일에는 전달되지 않습니다.

2. **/etc/default/grub** 파일에 있는 **GRUB_CMDLINE_LINUX** 값의 콘텐츠로 BLS 스니펫을 덮어씁니다.

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline
   Generating grub configuration file …
   Adding boot menu entry for UEFI Firmware Settings …
   ```

```
done
```

참고

GRUB_TIMEOUT 키 변경(**/etc/default/grub** GRUB 설정 파일에 포함)과 같은 기타 변경 사항이 기본적으로 새 **grub.cfg** 로 전파됩니다.

검증

1. 운영 체제를 재부팅합니다.
2. 매개 변수가 **/proc/cmdline** 파일에 포함되어 있는지 확인합니다.
   예를 들어 **/proc/cmdline** 에는 **noapic** 커널 매개변수가 포함되어 있습니다.

```
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-4.18.0-425.3.1.el8.x86_64 root=/dev/mapper/RHELCSB-Root ro vconsole.keymap=us crashkernel=auto rd.lvm.lv=RHELCSB/Root rd.luks.uuid=luks-d8a28c4c-96aa-4319-be26-96896272151d rhgb quiet noapic rd.luks.key=d8a28c4c-96aa-4319-be26-96896272151d=/keyfile:UUID=c47d962e-4be8-41d6-8216-8cf7a0d3b911 ipv6.disable=1
```

5장. 런타임 시 커널 매개변수 구성

시스템 관리자는 런타임 시 Red Hat Enterprise Linux 커널의 여러 가지 동작을 수정할 수 있습니다. **/etc/sysctl.d/** 및 **/proc/sys/** 디렉터리의 구성 파일을 수정하여 런타임 시 커널 매개변수를 구성합니다.

중요

프로덕션 시스템에서 커널 매개 변수를 구성하려면 신중하게 계획해야 합니다. 계획되지 않은 변경으로 인해 커널이 불안정해 시스템을 재부팅해야 합니다. 커널 값을 변경하기 전에 유효한 옵션을 사용하고 있는지 확인합니다.

5.1. 커널 매개변수 정의

커널 매개변수는 시스템이 실행되는 동안 조정할 수 있는 튜닝 가능한 값입니다. 변경 사항을 적용하려면 커널을 재부팅하거나 다시 컴파일할 필요가 없습니다.

다음을 통해 커널 매개변수를 처리할 수 있습니다.

- **sysctl** 명령
- **/proc/sys/** 디렉터리에 마운트된 가상 파일 시스템
- **/etc/sysctl.d/** 디렉터리의 구성 파일

튜닝 가능 항목은 커널 하위 시스템에 따라 클래스로 나뉩니다. Red Hat Enterprise Linux에는 다음과 같은 조정 가능한 클래스가 있습니다.

표 5.1. sysctl 클래스 테이블

튜닝 가능한 클래스	하위 시스템
abi	실행 도메인 및 개인 정보
crypto	암호화 인터페이스
debug	커널 디버깅 인터페이스
Dev	장치 별 정보
fs	글로벌 및 특정 파일 시스템 튜닝 가능 항목
kernel	글로벌 커널 튜닝 가능 항목
net	네트워크 튜닝 가능 항목
sunrpc	Sun Remote Procedure call (NFS)
user	사용자 네임스페이스 제한
vm	메모리, 버퍼 및 캐시 튜닝 및 관리

추가 리소스

- **sysctl(8)** 및 **sysctl.d(5)** 매뉴얼 페이지

5.2. SYSCTL을 사용하여 임시로 커널 매개변수 설정

sysctl 명령을 사용하여 런타임 시 커널 매개변수를 일시적으로 설정합니다. 명령은 튜닝 가능 항목을 나열하고 필터링하는 데도 유용합니다.

사전 요구 사항

- 루트 권한

절차

1. 모든 매개변수 및 해당 값을 나열합니다.

```
# sysctl -a
```

참고

 # sysctl -a 명령은 런타임 시 및 부팅 시 조정할 수 있는 커널 매개 변수를 표시합니다.

2. 일시적으로 매개변수를 구성하려면 다음을 입력합니다.

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

위의 샘플 명령은 시스템이 실행되는 동안 매개변수 값을 변경합니다. 변경 사항은 다시 시작할 필요 없이 즉시 적용됩니다.

참고

변경 사항은 시스템을 재부팅한 후 기본값으로 돌아갑니다.

추가 리소스

- **sysctl(8)** 매뉴얼 페이지
- sysctl을 사용하여 커널 매개변수 영구적으로 설정
- /etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정

5.3. SYSCTL을 사용하여 커널 매개변수 영구적으로 설정

sysctl 명령을 사용하여 커널 매개변수를 영구적으로 설정합니다.

사전 요구 사항

- 루트 권한

절차

1. 모든 매개변수를 나열합니다.

```
# sysctl -a
```

명령은 런타임 시 구성할 수 있는 모든 커널 매개 변수를 표시합니다.

2. 매개변수를 영구적으로 설정합니다.

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

샘플 명령은 조정 가능한 값을 변경하고 **/etc/sysctl.conf** 파일에 씁니다. 이 파일은 커널 매개변수의 기본값을 덮어씁니다. 변경 사항은 다시 시작할 필요 없이 즉시 영구적으로 적용됩니다.

참고

커널 매개변수를 영구적으로 수정하려면 **/etc/sysctl.d/** 디렉토리에 있는 구성 파일을 수동으로 변경할 수도 있습니다.

추가 리소스

- **sysctl(8)** 및 **sysctl.conf(5)** 매뉴얼 페이지
- /etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정

5.4. /ETC/SYSCTL.D/의 구성 파일을 사용하여 커널 매개변수 조정

커널 매개변수를 영구적으로 설정하려면 **/etc/sysctl.d/** 디렉토리의 설정 파일을 수동으로 수정합니다.

사전 요구 사항

- 루트 권한

절차

1. **/etc/sysctl.d/** 에 새 구성 파일을 만듭니다.

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

2. 커널 매개변수를 행당 하나씩 포함합니다.

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

3. 구성 파일을 저장합니다.

4. 변경 사항을 적용하려면 시스템을 재부팅합니다.

   - 또는 재부팅하지 않고 변경 사항을 적용하려면 다음을 입력합니다.

```
# sysctl -p /etc/sysctl.d/<some_file.conf>
```

명령을 사용하면 이전에 만든 구성 파일에서 값을 읽을 수 있습니다.

추가 리소스

- **sysctl(8)**, **sysctl.d(5)** 매뉴얼 페이지

5.5. /PROC/SYS/를 통해 일시적으로 커널 매개변수 구성

/proc/sys/ 가상 파일 시스템 디렉터리의 파일을 통해 커널 매개변수를 일시적으로 설정합니다.

사전 요구 사항

- 루트 권한

절차

1. 구성할 커널 매개변수를 확인합니다.

```
# ls -l /proc/sys/<TUNABLE_CLASS>/
```

   명령에서 반환한 쓰기 가능한 파일은 커널을 구성하는 데 사용할 수 있습니다. 읽기 전용 권한이 있는 파일은 현재 설정에 대한 피드백을 제공합니다.

2. 커널 매개변수에 타겟 값을 할당합니다.

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

   명령을 사용하면 시스템을 다시 시작하면 구성이 사라집니다.

3. 선택적으로 새로 설정된 커널 매개 변수의 값을 확인합니다.

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

추가 리소스

- sysctl을 사용하여 커널 매개변수 영구적으로 설정
- /etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정

5.6. 추가 리소스

- IBM DB2용 Red Hat Enterprise Linux 튜닝

6장. RHEL 시스템 역할을 사용하여 커널 매개변수를 영구적으로 구성

kernel_settings RHEL 시스템 역할을 사용하여 한 번에 여러 클라이언트에서 커널 매개변수를 구성할 수 있습니다. 이 해결 방법:

- 효율적인 입력 설정을 통해 친숙한 인터페이스를 제공합니다.
- 의도한 모든 커널 매개변수를 한 곳에 유지합니다.

제어 머신에서 **kernel_settings** 역할을 실행하면 커널 매개변수가 관리 시스템에 즉시 적용되며 재부팅 후에도 유지됩니다.

중요

RHEL 채널을 통해 제공되는 RHEL 시스템 역할은 RHEL 고객이 기본 AppStream 리포지토리에서 RPM 패키지로 사용할 수 있습니다. RHEL 시스템 역할은 Ansible Automation Hub를 통해 Ansible 서브스크립션을 통해 고객에게 컬렉션으로 제공됩니다.

6.1. KERNEL_SETTINGS RHEL 시스템 역할 소개

RHEL 시스템 역할은 여러 시스템을 원격으로 관리하는 일관된 구성 인터페이스를 제공하는 역할 집합입니다.

kernel_settings RHEL 시스템 역할을 사용하여 커널의 자동화된 구성을 위해 RHEL 시스템 역할이 도입되었습니다. **rhel-system-roles** 패키지에는 이 시스템 역할과 참조 문서가 포함되어 있습니다.

자동화된 방식으로 하나 이상의 시스템에 커널 매개 변수를 적용하려면 플레이북에서 선택한 역할 변수 중 하나 이상과 함께 **kernel_settings** 역할을 사용합니다. 플레이북은 사람이 읽을 수 있고 YAML 형식으로 작성된 하나 이상의 플레이 목록입니다.

인벤토리 파일을 사용하여 Ansible이 플레이북에 따라 구성할 시스템 세트를 정의할 수 있습니다.

kernel_settings 역할을 사용하면 다음을 구성할 수 있습니다.

- **kernel_settings_sysctl** 역할 변수를 사용하는 커널 매개변수
- **kernel_settings_sysfs** 역할 변수를 사용하는 다양한 커널 하위 시스템, 하드웨어 장치 및 장치 드라이버
- **kernel_settings_systemd_cpu_affinity** 역할 변수를 사용하여 **systemd** 서비스 관리자 및 프로세스용 CPU 선호도
- 커널 메모리 하위 시스템은 kernel_settings_transparent_hugepages 및 **kernel_settings_transparent_hugepages _defrag** 역할 변수를 사용하여 hugepages를 투명하게 합니다.

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md** 파일
- **/usr/share/doc/rhel-system-roles/kernel_settings/** directory
- 플레이북 작업

- 인벤토리를 빌드하는 방법

6.2. KERNEL_SETTINGS RHEL 시스템 역할을 사용하여 선택한 커널 매개변수 적용

다음 단계에 따라 여러 관리 운영 체제에 영향을 주므로 Ansible 플레이북을 준비하고 적용하여 커널 매개변수를 원격으로 구성합니다.

사전 요구 사항

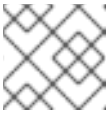
- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Configure kernel settings
  hosts: managed-node-01.example.com
  roles:
    - 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: madvise
```

   - **name**: 임의의 문자열을 레이블로 연결하고 플레이에 대한 항목을 식별하는 선택적 키입니다.
   - **hosts**: 플레이가 실행되는 호스트를 지정하는 키입니다. 이 키의 값 또는 값은 관리 호스트의 개별 이름으로 제공되거나 **인벤토리** 파일에 정의된 호스트 그룹으로 제공될 수 있습니다.
   - **vars**: 선택한 커널 매개 변수 이름 및 값을 설정해야 하는 변수 목록을 나타내는 플레이북의 섹션입니다.
   - **role**: **vars** 섹션에 언급된 매개변수 및 값을 구성하려는 RHEL 시스템 역할을 지정하는 키입니다.



참고

필요에 맞게 플레이북에서 커널 매개변수 및 해당 값을 수정할 수 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

-

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. Playbook을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

4. 관리 호스트를 재시작하고 영향을 받는 커널 매개 변수를 확인하여 변경 사항이 적용되고 재부팅 해도 지속되는지 확인합니다.

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md** 파일
- **/usr/share/doc/rhel-system-roles/kernel_settings/** directory
- 플레이북으로 작업
- 변수 사용
- 역할

7장. 커널 실시간 패치 적용

Red Hat Enterprise Linux 커널 라이브 패치 솔루션을 사용하여 프로세스를 재부팅하거나 다시 시작하지 않고도 실행 중인 커널을 패치할 수 있습니다.

이 솔루션을 사용하면 시스템 관리자가 다음을 수행할 수 있습니다.

- 커널에 중요한 보안 패치를 즉시 적용할 수 있습니다.
- 장기간 실행 중인 작업이 완료될 때까지 기다리거나, 사용자가 로그아웃하거나 예약된 다운 타임을 기다릴 필요가 없습니다.
- 시스템의 가동 시간을 더 많이 제어하고 보안이나 안정성을 포기하지 않습니다.

커널 실시간 패치 솔루션을 사용하여 모든 중요 또는 중요 CVE가 해결되는 것은 아닙니다. Red Hat의 목표는 보안 관련 패치를 완전히 제거하는 것이 아니라 필요한 재부팅을 줄이는 것입니다. 실시간 패치 범위에 대한 자세한 내용은 *고객 포털 솔루션 문서를* 참조하십시오.

주의

일부 비호환성은 커널 실시간 패치와 기타 커널 하위 구성 요소 간에 존재합니다. 관련 자료

커널 실시간 패치를 사용하기 전에 kpatch의 제한 사항.

참고

커널 라이브 패치 업데이트의 지원 주기에 대한 자세한 내용은 다음을 참조하십시오.

- 커널 라이브 패치 지원 캐싱 업데이트
- 커널 라이브 패치 라이프 사이클

7.1. KPATCH의 제한 사항

- **kpatch** 기능은 범용 커널 업그레이드 메커니즘이 아닙니다. 시스템을 재부팅할 때 간단한 보안 및 버그 수정 업데이트를 적용하는 데 사용됩니다.
- 패치를 로드하는 동안 또는 패치를 로드한 후 **SystemTap** 또는 **kprobe** 툴을 사용하지 마십시오. 이러한 프로브가 제거될 때까지 패치가 적용되지 않을 수 있습니다.

7.2. 타사 실시간 패치 지원

kpatch 유틸리티는 Red Hat 리포지토리에서 제공하는 RPM 모듈을 통해 Red Hat에서 지원하는 유일한 커널 라이브 패치 유틸리티입니다. Red Hat은 Red Hat 자체에서 제공하지 않은 실시간 패치를 지원하지 않습니다.

타사 라이브 패치에서 발생하는 문제에 대한 지원이 필요한 경우 Red Hat은 근본 원인 결정이 필요한 조사에서 실시간 패치 벤더와 함께 케이스를 열 것을 권장합니다. 이를 통해 공급 업체에서 허용하는 경우 소스 코드를 제공하고 지원 조직에서 Red Hat 지원에 대한 조사를 계산하기 전에 근본 원인 결정에 도움을

제공할 수 있습니다.

타사 라이브 패치를 사용하여 실행되는 모든 시스템의 경우 Red Hat은 Red Hat이 제공하고 지원되는 소프트웨어를 사용하여 복제하도록 요청할 수 있습니다. 이러한 문제가 발생하지 않는 경우 실시간 패치가 적용되지 않고 유사한 시스템 및 워크로드를 테스트 환경에 배포하여 동일한 동작이 관찰되었는지 확인합니다.

타사 소프트웨어 지원 정책에 대한 자세한 내용은 Red Hat 글로벌 지원 서비스가 타사 소프트웨어, 드라이버 및/또는 인증되지 않은 하드웨어/하이퍼 바이저 또는 게스트 운영 체제를 처리하는 방법을참조하십시오.

7.3. 커널 라이브 패치 액세스

커널 라이브 패치 기능은 RPM 패키지로 제공되는 커널 모듈(**k**mod)으로 구현됩니다.

모든 고객은 일반적인 채널을 통해 제공되는 커널 라이브 패치에 액세스할 수 있습니다. 그러나 연장된 지원 오퍼링에 가입하지 않은 고객은 다음 마이너 릴리스가 출시되면 현재 마이너 릴리스에 대한 새 패치에 대한 액세스 권한을 잃게 됩니다. 예를 들어, 표준 서브스크립션을 보유한 고객은 RHEL 9.2 커널이 릴리스될 때까지 RHEL 9.1 커널에 대한 패치만 사용할 수 있습니다.

7.4. 커널 실시간 패치의 구성 요소

커널 실시간 패치의 구성 요소는 다음과 같습니다.

커널 패치 모듈

- 커널 실시간 패치를 위한 전달 메커니즘.
- 커널 패치를 위해 특별히 빌드된 커널 모듈입니다.
- patch 모듈에는 커널에 대해 원하는 수정 사항 코드가 포함되어 있습니다.
- 패치 모듈은 **라이브patch** 커널 하위 시스템에 등록되고 대체 함수에 대한 해당 포인터와 함께 대체할 원래 기능에 대한 정보를 제공합니다. 커널 패치 모듈은 RPM으로 제공됩니다.
- 명명 규칙은 **kpatch_<kernel 버전>_<kpatch version>_<kpatch release>** 입니다. 이름의 "커널 버전" 부분에는 *점이 밑줄*로 교체되어 있습니다.

kpatch 유틸리티

패치 모듈을 관리하는 명령줄 유틸리티입니다.

kpatch 서비스

다중 사용자 지정에 필요한 systemd 서비스. 이 대상은 부팅 시 커널 패치 모듈을 로드합니다.

kpatch-dnf 패키지

RPM 패키지 형태로 제공되는 DNF 플러그인입니다. 이 플러그인은 커널 라이브 패치에 대한 자동 서브스크립션을 관리합니다.

7.5. 커널 실시간 패치 작동 방식

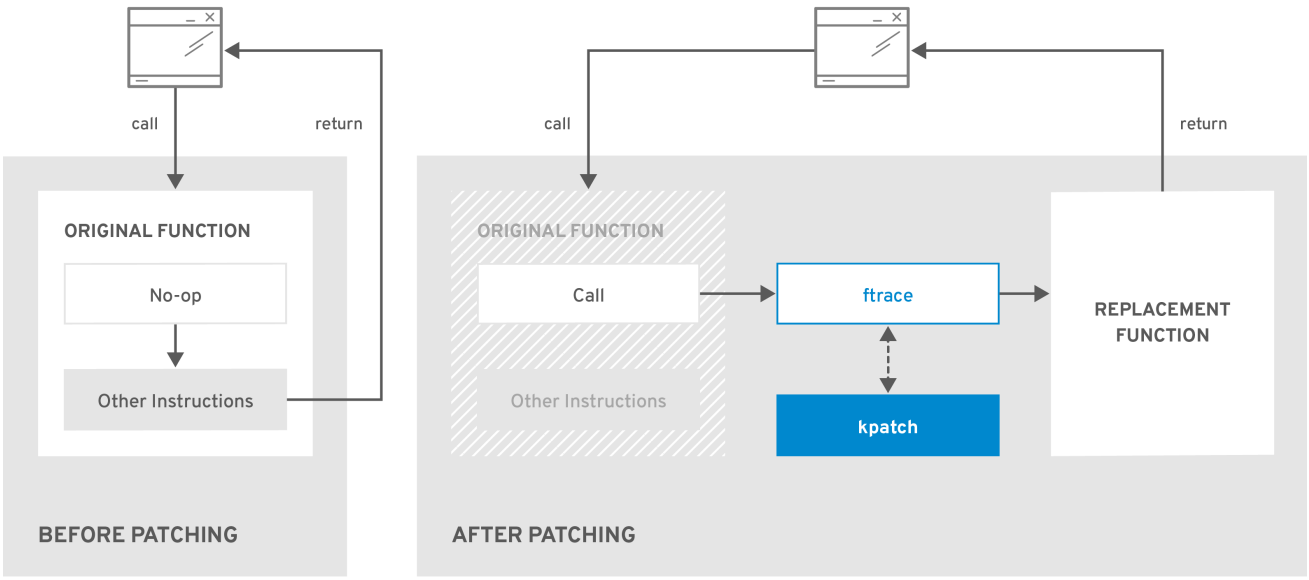
kpatch 커널 패치 솔루션은 **라이브patch** 커널 하위 시스템을 사용하여 이전 함수를 새 기능으로 리디렉션합니다. 라이브 커널 패치가 시스템에 적용되면 다음과 같은 일이 발생합니다.

1. 커널 패치 모듈은 **/var/lib/kpatch/** 디렉터리에 복사되고 다음 부팅 시 **systemd** 를 통해 커널에 다시 적용되도록 등록됩니다.

2. kpatch 모듈이 실행 중인 커널에 로드되고 새 함수가 새 코드의 메모리에 있는 위치에 대한 포인터로 **ftrace** 메커니즘에 등록됩니다.

3. 커널이 패치된 기능에 액세스하면 원래 기능을 우회하고 커널을 패치된 함수 버전으로 리디렉션하는 **ftrace** 메커니즘으로 리디렉션됩니다.

그림 7.1. 커널 실시간 패치 작동 방식



7.6. 현재 설치된 커널을 실시간 패치 스트림에 구독

커널 패치 모듈은 패치되는 커널 버전에 따라 RPM 패키지로 제공됩니다. 각 RPM 패키지는 시간이 지남에 따라 누적 업데이트됩니다.

다음 절차에서는 지정된 커널의 향후 모든 누적 실시간 패치 업데이트를 구독하는 방법을 설명합니다. 실시간 패치는 누적되므로 지정된 커널에 대해 배포되는 개별 패치를 선택할 수 없습니다.

주의

Red Hat은 Red Hat 지원 시스템에 적용된 타사의 라이브 패치를 지원하지 않습니다.

사전 요구 사항

- 루트 권한

절차

1. 선택적으로 커널 버전을 확인합니다.

```
# uname -r
5.14.0-1.el9.x86_64
```

2. 커널 버전에 해당하는 실시간 패치 패키지를 검색합니다.

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

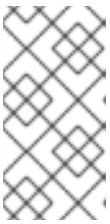
3. 실시간 패치 패키지를 설치합니다.

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

위의 명령은 해당 특정 커널에 대해 최신 누적 라이브 패치를 설치하고 적용합니다.

라이브 패치 패키지의 버전이 1-1 이상이면 패키지에 patch 모듈이 포함됩니다. 이 경우 라이브 패치 패키지를 설치하는 동안 커널이 자동으로 패치됩니다.

커널 패치 모듈은 **/var/lib/kpatch/** 디렉토리에 설치되어 나중에 재부팅할 때 **systemd** 시스템과 서비스 관리자에 의해 로드됩니다.



참고

지정된 커널에 대해 라이브 패치를 사용할 수 없는 경우 빈 실시간 패치 패키지가 설치됩니다. 빈 라이브 패치 패키지에는 *kpatch_version-kpatch_release* 가 0-0입니다(예: **kpatch-patch-5_14_0-1-0.x86_64.rpm**). 빈 RPM을 설치하면 지정된 커널의 향후 모든 실시간 패치에 시스템을 서브스크립션합니다.

검증

- 설치된 모든 커널이 패치되었는지 확인합니다.

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

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

출력에서는 커널 패치 모듈이 **kpatch-patch-5_14_0-1-0-1.el9.x86_64.rpm** 패키지의 최신 수정 사항과 함께 패치된 커널에 로드되었음을 보여줍니다.

참고

kpatch list 명령을 입력하면 빈 라이브 패치 패키지가 반환되지 않습니다. 대신 **rpm -qa | grep kpatch** 명령을 사용합니다.

```
# rpm -qa | grep kpatch
kpatch-dnf-0.4-3.el9.noarch
kpatch-0.9.7-2.el9.noarch
kpatch-patch-5_14_0-284_25_1-0-0.el9_2.x86_64
```

추가 리소스

- **kpatch(1)** 매뉴얼 페이지
- RHEL 9 콘텐츠 설치

7.7. 라이브 패치 스트림으로 향후 커널 자동 구독

kpatch-dnf DNF 플러그인을 사용하여 커널 패치 모듈 (커널 라이브 패치라고도 함)에서 제공하는 수정 사항을 시스템에 구독할 수 있습니다. 이 플러그인을 사용하면 현재 시스템에서 사용하는 모든 커널에 대해 **자동** 서브스크립션을 사용할 수 있으며 **향후 커널에서 설치되는 커널에도 사용할 수**있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. 선택적으로, 설치된 모든 커널 및 현재 실행 중인 커널을 확인합니다.

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

# uname -r
5.14.0-2.el9.x86_64
```

2. **kpatch-dnf** 플러그인을 설치합니다.

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

3. 커널 라이브 패치에 대한 자동 서브스크립션을 활성화합니다.

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

Transaction Summary
================================================
Install  2 Packages
…
```

이 명령은 현재 설치된 모든 커널을 서브스크립션하여 커널 라이브 패치를 수신합니다. 이 명령은 설치된 모든 커널에 대해 최신 누적 라이브 패치를 설치하고 적용합니다.

나중에 커널을 업데이트하면 새 커널 설치 프로세스 중에 실시간 패치가 자동으로 설치됩니다.

커널 패치 모듈은 **/var/lib/kpatch/** 디렉토리에 설치되어 나중에 재부팅할 때 **systemd** 시스템과 서비스 관리자에 의해 로드됩니다.

참고

지정된 커널에 대해 라이브 패치를 사용할 수 없는 경우 빈 실시간 패치 패키지가 설치됩니다. 라이브 패치 패키지에는 *kpatch_version-kpatch_release* 가 0-0입니다 (예: **kpatch-patch-5_14_0-1-0.el9.x86_64.rpm**). 빈 RPM을 설치하면 지정된 커널의 향후 모든 실시간 패치에 시스템을 서브스크립션합니다.

검증

- 설치된 모든 커널이 패치되었는지 확인합니다.

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

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

출력에는 실행 중인 커널과 설치된 다른 커널이 **kpatch-patch-5_14_0-1-0-1.el9.x86_64.rpm** 및 **kpatch-patch-5_14_0-2-0-1.el9.x86_64.rpm** 패키지의 수정 사항으로 패치되었습니다.

참고

kpatch list 명령을 입력하면 빈 라이브 패치 패키지가 반환되지 않습니다. 대신 **rpm -qa | grep kpatch** 명령을 사용합니다.

```
# rpm -qa | grep kpatch
kpatch-dnf-0.4-3.el9.noarch
kpatch-0.9.7-2.el9.noarch
kpatch-patch-5_14_0-284_25_1-0-0.el9_2.x86_64
```

추가 리소스

- **kpatch(1)** 및 **dnf-kpatch(8)** 매뉴얼 페이지

7.8. 라이브 패치 스트림에 대한 자동 서브스크립션 비활성화

커널 패치 모듈에서 제공하는 수정 사항에 맞게 시스템을 서브스크립션하면 서브스크립션이 자동으로 **제공됩니다**. 이 기능을 비활성화하여 **kpatch-patch** 패키지의 자동 설치를 비활성화할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. 선택적으로, 설치된 모든 커널 및 현재 실행 중인 커널을 확인합니다.

```
# dnf list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
```

```
kernel-core.x86_64          5.14.0-1.el9          @beaker-BaseOS
kernel-core.x86_64          5.14.0-2.el9          @@commandline
...

# uname -r
5.14.0-2.el9.x86_64
```

2. 커널 라이브 패치에 대한 자동 서브스크립션을 비활성화합니다.

```
# dnf kpatch manual
Updating Subscription Management repositories.
```

검증 단계

- 성공적인 결과를 확인할 수 있습니다:

```
# 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
```

추가 리소스

- **kpatch(1)** 및 **dnf-kpatch(8)** 매뉴얼 페이지

7.9. 커널 패치 모듈 업데이트

커널 패치 모듈은 RPM 패키지를 통해 전달 및 적용되므로 누적 커널 패치 모듈을 업데이트하는 것은 다른 RPM 패키지 업데이트와 같습니다.

사전 요구 사항

- 현재 설치된 커널 대체에 설명된 대로 실시간 패치 스트림에 시스템이 서브스크립션됩니다 .

절차

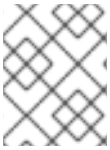
- 현재 커널에 대해 새 누적 버전으로 업데이트합니다.

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

위의 명령은 현재 실행 중인 커널에 사용 가능한 모든 업데이트를 자동으로 설치하고 적용합니다. 향후 릴리스된 누적 패치를 포함합니다.

- 또는 설치된 모든 커널 패치 모듈을 업데이트합니다.

```
# dnf update "kpatch-patch"
```



참고

시스템이 동일한 커널로 재부팅되면 **kpatch.service** systemd 서비스에서 커널이 자동으로 패치됩니다.

추가 리소스

- RHEL에서 *소프트웨어 패키지 업데이트*

7.10. 실시간 패치 패키지 제거

라이브 패치 패키지를 제거하여 Red Hat Enterprise Linux 커널 실시간 패치 솔루션을 비활성화하십시오.

사전 요구 사항

- 루트 권한
- 실시간 패치 패키지가 설치되어 있습니다.

절차

1. 라이브 패치 패키지를 선택합니다.

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

   위의 예제 출력에는 설치한 실시간 패치 패키지가 나열됩니다.

2. 라이브 패치 패키지를 제거합니다.

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

   실시간 패치 패키지가 제거되면 커널은 다음 재부팅까지 패치되지만 커널 패치 모듈은 디스크에서 제거됩니다. 향후 재부팅 시 해당 커널이 더 이상 패치되지 않습니다.

3. 시스템을 재부팅합니다.

4. 실시간 패치 패키지가 제거되었는지 확인합니다.

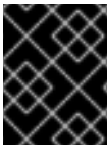
```
# dnf list installed | grep kpatch-patch
```

   패키지가 성공적으로 제거된 경우 명령은 출력을 표시하지 않습니다.

5. 필요한 경우 커널 실시간 패치 솔루션이 비활성화되어 있는지 확인합니다.

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

   예제 출력에서는 현재 로드된 패치 모듈이 없기 때문에 커널이 패치되지 않았으며 실시간 패치 솔루션이 활성화되지 않음을 보여줍니다.



중요

현재 Red Hat은 시스템을 재부팅하지 않고 실시간 패치 복원을 지원하지 않습니다. 문제가 있는 경우 Red Hat 지원팀에 문의하십시오.

추가 리소스

- **kpatch(1)** 매뉴얼 페이지
- RHEL에서 *설치된 패키지 제거*

7.11. 커널 패치 모듈 설치 제거

Red Hat Enterprise Linux 커널 실시간 패치 솔루션이 후속 부팅 시 커널 패치 모듈을 적용하지 못하도록 합니다.

사전 요구 사항

- 루트 권한
- 실시간 패치 패키지가 설치되어 있습니다.
- 커널 패치 모듈이 설치 및 로드되었습니다.

절차

1. 커널 패치 모듈을 선택합니다.

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

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

2. 선택한 커널 패치 모듈을 설치 제거합니다.

```
# kpatch uninstall kpatch_5_14_0_1_0_1
uninstalling kpatch_5_14_0_1_0_1 (5.14.0-1.el9.x86_64)
```

- 제거된 커널 패치 모듈이 계속 로드되었습니다.

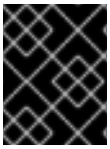
```
# kpatch list
Loaded patch modules:
kpatch_5_14_0_1_0_1 [enabled]

Installed patch modules:
<NO_RESULT>
```

선택한 모듈이 제거되면 커널은 다음 재부팅까지 패치되지만 커널 패치 모듈은 디스크에서 제거됩니다.

3. 시스템을 재부팅합니다.
4. 필요한 경우 커널 패치 모듈이 제거되었는지 확인합니다.

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

위의 예제 출력에서는 로드되거나 설치된 커널 패치 모듈이 없으므로 커널이 패치되지 않고 커널 실시간 패치 솔루션이 활성 상태가 아닙니다.



중요

현재 Red Hat은 시스템을 재부팅하지 않고 실시간 패치 복원을 지원하지 않습니다. 문제가 있는 경우 Red Hat 지원팀에 문의하십시오.

추가 리소스

- **kpatch(1)** 매뉴얼 페이지

7.12. KPATCH.SERVICE 비활성화

Red Hat Enterprise Linux 커널 실시간 패치 솔루션이 후속 부팅 시 전역적으로 모든 커널 패치 모듈을 적용하지 못하도록 합니다.

사전 요구 사항

- 루트 권한
- 실시간 패치 패키지가 설치되어 있습니다.
- 커널 패치 모듈이 설치 및 로드되었습니다.

절차

1. **kpatch.service** 가 활성화되어 있는지 확인합니다.

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

2. **kpatch.service** 를 비활성화합니다.

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

- 적용된 커널 패치 모듈이 계속 로드됩니다.

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

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

3. 시스템을 재부팅합니다.
4. 필요한 경우 **kpatch.service** 의 상태를 확인합니다.

```
# 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)
```

예제 출력은 **kpatch.service** 가 비활성화되었으며 실행되지 않음을 나타냅니다. 따라서 커널 실시간 패치 솔루션이 활성 상태가 아닙니다.

5. 커널 패치 모듈이 언로드되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:

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

위의 예제 출력에서는 커널 패치 모듈이 여전히 설치되어 있지만 커널에 패치되지 않음을 보여줍니다.

중요

현재 Red Hat은 시스템을 재부팅하지 않고 실시간 패치 복원을 지원하지 않습니다. 문제가 있는 경우 Red Hat 지원팀에 문의하십시오.

추가 리소스

- **kpatch(1)** 매뉴얼 페이지 systemd 관리

8장. 가상화된 환경에서 커널 패닉 매개변수를 비활성화 상태로 유지

RHEL 9에서 가상 머신을 구성할 때 가상 머신이 스포크한 소프트 잠금 잠금으로 어려움을 겪을 수 있으므로 **softlockup_panic** 및 **nmi_watchdog** 커널 매개변수를 활성화해서는 안 됩니다. 커널 패닉이 필요하지 않아야 합니다.

다음 섹션에서 이 조언 뒤에 있는 이유를 알아보십시오.

8.1. 소프트 잠금이란 무엇입니까?

소프트 잠금 기능은 일반적으로 버그로 인해 발생하며, 작업이 다시 예약하지 않고 CPU의 커널 공간을 실행할 때 발생합니다. 이 작업에서는 특정 CPU에서 다른 작업도 실행할 수 없습니다. 결과적으로 시스템 콘솔을 통해 사용자에게 경고가 표시됩니다. 이 문제를 소프트 잠금 기능 실행이라고도 합니다.

추가 리소스

- *CPU 소프트 잠금이란 무엇입니까?*

8.2. 커널 패닉을 제어하는 매개변수

다음 커널 매개 변수는 소프트 잠금이 탐지될 때 시스템의 동작을 제어하도록 설정할 수 있습니다.

softlockup_panic

소프트 잠금이 감지되면 커널이 패닉을 일으킬지 여부를 제어합니다.

유형	값	효과
정수	0	커널은 소프트 잠금 해제에 패닉을 초래하지 않습니다.
정수	1	소프트 잠금 장치에 커널 패닉

기본적으로 RHEL8에서는 이 값은 0입니다.

시스템은 먼저 패닉 상태가 되기 위해 하드 잠금을 감지해야 합니다. 탐지는 **nmi_watchdog** 매개변수에 의해 제어됩니다.

nmi_watchdog

잠금 감지 메커니즘(워치독)이 활성화되어 있는지 여부를 제어합니다. 이 매개 변수는 정수 유형입니다.

값	효과
0	lockup 탐지기 비활성화
1	잠금 탐지기 사용

하드 잠금 탐지기는 각 CPU를 모니터링하여 인터럽트에 응답할 수 있습니다.

watchdog_thresh

워치독 **hrtimer**, NMI 이벤트 및 소프트/하드 잠금 임계값의 빈도를 제어합니다.

기본 임계값	소프트 잠금 임계값
10초	2 * **watchdog_thresh**

이 매개변수를 0으로 설정하면 잠금 감지가 완전히 비활성화됩니다.

추가 리소스

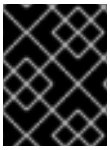
- *softlockup 탐지기 및 하드 잠금 탐지기*
- *커널 sysctl*

8.3. 가상화된 환경의 과도한 소프트 잠금 기능

소프트 잠금 장치는 일반적으로 커널 또는 하드웨어 버그를 나타냅니다. ??? 가상화 환경의 게스트 운영 체제에서 발생하는 것과 동일한 현상이 발생할 수 있습니다.

메모리와 같은 일부 특정 리소스에 대한 호스트 또는 높은 경합에 대한 작업 부하가 과도하면 일반적으로 심각한 소프트 잠금이 실행됩니다. 호스트가 게스트 CPU를 20초 이상 예약할 수 있기 때문입니다. 그런 다음 게스트 CPU가 호스트에서 실행되도록 다시 예약되면 타이머로 인해 트리거 *되는 시간* 이동이 발생합니다. 타이머에는 워치독 **hrtimer** 도 포함되어 있으므로 게스트 CPU에 소프트 잠금을 보고할 수 있습니다.

가상화된 환경의 소프트 잠금이 급증할 수 있으므로 소프트 잠금이 게스트 CPU에 보고될 때 시스템 패닉을 유발하는 커널 매개 변수를 활성화해서는 안 됩니다.



중요

게스트의 소프트 잠금을 이해하려면 호스트가 게스트를 작업으로 예약하고 게스트가 자체 작업을 예약한다는 것을 알아야 합니다.

추가 리소스

- 소프트 잠금이란 무엇입니까?
- *가상 머신 구성 요소 및 상호 작용*
- *가상 머신에서 "BUG: soft lockup"을 보고합니다.*

9장. 데이터베이스 서버에 대한 커널 매개변수 조정

특정 데이터베이스 애플리케이션의 성능에 영향을 미칠 수 있는 다양한 커널 매개변수 세트가 있습니다. 데이터베이스 서버 및 데이터베이스의 효율적인 작동을 보호하려면 그에 따라 각 커널 매개변수를 구성합니다.

9.1. 데이터베이스 서버 소개

데이터베이스 서버는 DBMS(데이터베이스 관리 시스템)의 기능을 제공하는 서비스입니다. DBMS는 데이터베이스 관리를 위한 유틸리티를 제공하고 최종 사용자, 애플리케이션 및 데이터베이스와 상호 작용합니다.

Red Hat Enterprise Linux 9는 다음과 같은 데이터베이스 관리 시스템을 제공합니다.

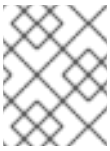
- **MariaDB 10.5**
- **MariaDB 10.11** - RHEL 9.4 이후 사용 가능
- **MySQL 8.0**
- **PostgreSQL 13**
- **PostgreSQL 15** - RHEL 9.2 이후 사용 가능
- **PostgreSQL 16** - RHEL 9.4 이후 사용 가능
- **redis 6**

9.2. 데이터베이스 애플리케이션의 성능에 영향을 주는 매개변수

다음 커널 매개 변수는 데이터베이스 애플리케이션의 성능에 영향을 미칩니다.

fs.aio-max-nr

시스템에서 서버에서 처리할 수 있는 최대 비동기 I/O 작업 수를 정의합니다.



참고

fs.aio-max-nr 매개변수를 늘리면 aio 제한을 늘리는 것 이상의 추가 변경 사항이 발생하지 않습니다.

fs.file-max

시스템이 모든 인스턴스에서 지원하는 최대 파일 처리 수(임시 파일 이름 또는 열려 있는 파일에 할당된 ID)를 정의합니다.
커널은 애플리케이션에서 파일 처리를 요청할 때마다 파일 처리를 동적으로 할당합니다. 그러나 커널은 애플리케이션에서 릴리스할 때 이러한 파일을 처리하지 않습니다. 커널은 이러한 파일 처리를 대신 재활용합니다. 즉, 현재 사용된 파일 핸들 수가 낮더라도 할당된 파일 처리의 총 수가 증가합니다.

kernel.shmall

시스템 전체에서 사용할 수 있는 총 공유 메모리 페이지 수를 정의합니다. 전체 메인 메모리를 사용하려면 **kernel.shmall** 매개 변수의 값은 전체 메인 메모리 크기여야 합니다.

kernel.shmmax

Linux 프로세스에서 가상 주소 공간에 할당할 수 있는 단일 공유 메모리 세그먼트의 최대 크기(바이트)를 정의합니다.

kernel.shmmni

데이터베이스 서버가 처리할 수 있는 최대 공유 메모리 세그먼트 수를 정의합니다.

net.ipv4.ip_local_port_range

특정 포트 번호 없이 데이터베이스 서버에 연결하려는 프로그램에 시스템에서 사용할 수 있는 포트 범위를 정의합니다.

net.core.rmem_default

TCP(Transmission Control Protocol)를 통해 기본 수신 소켓 메모리를 정의합니다.

net.core.rmem_max

TCP(Transmission Control Protocol)를 통해 최대 수신 소켓 메모리를 정의합니다.

net.core.wmem_default

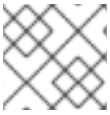
TCP(Transmission Control Protocol)를 통해 기본 전송 소켓 메모리를 정의합니다.

net.core.wmem_max

TCP(Transmission Control Protocol)를 통해 최대 전송 소켓 메모리를 정의합니다.

vm.dirty_bytes / vm.dirty_ratio

write() 함수에서 더티 데이터를 생성하는 프로세스에 더티 가능한 메모리의 백분율로 임계값을 정의합니다.



참고

한 번에 **vm.dirty_bytes** 또는 **vm.dirty_ratio** 를 지정할 수 있습니다.

vm.dirty_background_bytes / vm.dirty_background_ratio

커널이 더티 데이터를 하드 디스크에 적극적으로 작성하려고 시도하는 더티 가능한 메모리의 백분율로 임계값을 정의합니다.

참고

vm.dirty_background_bytes 또는 **vm.dirty_background_ratio** 중 하나를 지정할 수 있습니다.

vm.dirty_writeback_centisecs

더티 데이터를 하드 디스크에 작성하는 커널 스레드의 정기적인 워크업 사이에 시간 간격을 정의합니다. 이 커널 매개변수는 1초의 100번째로 측정합니다.

vm.dirty_expire_centisecs

더티 데이터가 하드 디스크에 쓸 수 있을 만큼 오래 된 후 시간을 정의합니다.
이 커널 매개변수는 1초의 100번째로 측정합니다.

추가 리소스

- *더티 페이지 캐시 쓰기 및 vm.dirty 매개변수*

10장. 커널 로깅 시작하기

로그 파일은 커널, 서비스 및 커널에서 실행되는 애플리케이션을 포함하여 시스템에 대한 메시지가 포함된 파일입니다. Red Hat Enterprise Linux의 로깅 시스템은 내장 syslog 프로토콜을 기반으로 합니다. 다양한 유틸리티에서는 이 시스템을 사용하여 이벤트를 기록하고 이를 로그 파일에 구성합니다. 이러한 파일은 운영 체제를 감사하거나 문제를 해결할 때 유용합니다.

10.1. 커널 링 버퍼란?

부팅 프로세스 중에 콘솔은 시스템 시작의 초기 단계에 대한 중요한 정보를 많이 제공합니다. 초기 메시지의 손실을 방지하기 위해 커널은 링 버퍼라는 내용을 사용합니다. 이 버퍼는 커널 코드 내에서 **printk()** 함수에 의해 생성된 부팅 메시지를 포함하여 모든 메시지를 저장합니다. 그런 다음 커널 링 버퍼의 메시지를 읽고 영구 스토리지의 로그 파일에 저장합니다(예: **syslog** 서비스에 의해).

위에서 언급한 버퍼는 크기가 고정된 순환 데이터 구조이며 커널에 하드 코딩되어 있습니다. 사용자는 **dmesg** 명령 또는 **/var/log/boot.log** 파일을 통해 커널 링 버퍼에 저장된 데이터를 표시할 수 있습니다. 링 버퍼가 가득 차면 새 데이터가 이전 데이터를 덮어씁니다.

추가 리소스

- **syslog(2)** 및 **dmesg(1)** 설명서 페이지

10.2. 로그 수준 및 커널 로깅에서 출력된 출력 역할

커널이 보고하는 각 메시지에는 메시지의 중요도를 정의하는 로그 수준이 연결되어 있습니다. 커널 링 버퍼는 What is the kernel ring buffer이며, 모든 로그 수준의 커널 메시지를 수집합니다. 이는 버퍼에서 콘솔에 출력되는 메시지를 정의하는 **kernel.printk** 매개변수입니다.

로그 수준 값은 다음 순서로 중단됩니다.

0
: 커널 긴급 상황. 시스템을 사용할 수 없습니다.

1
: 커널 경고. 즉시 조치를 취해야 합니다.

2
: 커널의 상태는 중요한 것으로 간주됩니다.

3
: 일반 커널 오류 상태입니다.

4
: 일반 커널 경고 조건.

5
: 정상이지만 중요한 상태에 대한 커널 알림입니다.

6
: 커널 정보 메시지.

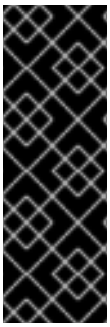
7
: 커널 디버그 수준 메시지입니다.

기본적으로 RHEL 9의 **kernel.printk** 에는 다음과 같은 네 가지 값이 포함되어 있습니다.

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

4개의 값은 순서대로 다음을 정의합니다.

1. 콘솔 로그 수준은 콘솔에 인쇄되는 메시지의 가장 낮은 우선 순위를 정의합니다.
2. 명시적 로그 수준이 첨부되지 않은 메시지의 기본 로그 수준입니다.
3. 콘솔 로그 수준에 대해 사용 가능한 가장 낮은 로그 수준 구성을 설정합니다.
4. 부팅 시 콘솔 로그 수준의 기본값을 설정합니다.
   위의 각 값은 오류 메시지를 처리하기 위한 다른 규칙을 정의합니다.



중요

기본 **7 4 1 1 printk** 값을 사용하면 커널 활동을 더 잘 디버깅할 수 있습니다. 그러나 직렬 콘솔과 연결된 경우 이 출력 설정으로 인해 RHEL 시스템이 일시적으로 응답하지 않을 수 있는 강력한 I/O 버스트가 발생할 수 있습니다. 이러한 상황을 방지하기 위해 **4 4 1 7** 의 인쇄 값을 설정하는 것은 일반적으로 작동하지만 추가 디버깅 정보가 손실되는 경우가 이에 해당합니다.

또한 **quiet** 또는 **debug** 와 같은 특정 커널 명령줄 매개 변수는 기본**kernel.printk** 값을 변경합니다.

추가 리소스

- **syslog(2)** 매뉴얼 페이지

11장. GRUB 다시 설치

GRUB 부트 로더를 다시 설치하여 일반적으로 GRUB 설치, 파일 누락 또는 손상된 시스템으로 인해 발생하는 특정 문제를 해결할 수 있습니다. 누락된 파일을 복원하고 부팅 정보를 업데이트하여 이러한 문제를 해결할 수 있습니다.

GRUB을 다시 설치하는 이유:

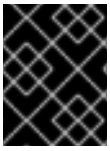
- GRUB 부트 로더 패키지 업그레이드.
- 다른 드라이브에 부팅 정보를 추가합니다.
- 사용자는 설치된 운영 체제를 제어하기 위해 GRUB 부트 로더가 필요합니다. 그러나 일부 운영 체제는 자체 부트 로더를 사용하여 설치하고 GRUB을 원하는 운영 체제에 제어 기능을 다시 설치합니다.

참고

GRUB은 파일이 손상되지 않은 경우에만 파일을 복원합니다.

11.1. BIOS 기반 시스템에 GRUB 다시 설치

BIOS 기반 시스템에 GRUB 부트 로더를 다시 설치할 수 있습니다. GRUB 패키지를 업데이트한 후 항상 GRUB을 다시 설치하십시오.



중요

기존 GRUB을 덮어쓰고 새 GRUB을 설치합니다. 설치 중에 시스템이 데이터 손상 또는 부팅 충돌을 유발하지 않는지 확인합니다.

절차

1. 설치된 장치에 GRUB을 다시 설치합니다. 예를 들어 **sda** 가 장치인 경우 다음을 수행합니다.

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

2. 변경 사항을 적용하려면 시스템을 재부팅합니다.

```
# reboot
```

추가 리소스

- **grub-install(1)** 매뉴얼 페이지

11.2. UEFI 기반 시스템에 GRUB 다시 설치

UEFI 기반 시스템에 GRUB 부트 로더를 다시 설치할 수 있습니다.

중요

설치 중에 시스템이 데이터 손상 또는 부팅 충돌을 유발하지 않는지 확인합니다.

절차

1. **grub2-efi** 및 **shim** 부트 로더 파일을 다시 설치합니다.

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

2. 변경 사항을 적용하려면 시스템을 재부팅합니다.

```
# reboot
```

11.3. IBM POWER 시스템에 GRUB 다시 설치

IBM Power 시스템의 PReP(Power PC Reference Platform) 부팅 파티션에 GRUB 부트 로더를 다시 설치할 수 있습니다. GRUB 패키지를 업데이트한 후 항상 GRUB을 다시 설치하십시오.

중요

기존 GRUB을 덮어쓰고 새 GRUB을 설치합니다. 설치 중에 시스템이 데이터 손상 또는 부팅 충돌을 유발하지 않는지 확인합니다.

절차

1. GRUB를 저장하는 디스크 파티션을 확인합니다.

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

2. 디스크 파티션에 GRUB을 다시 설치합니다.

```
# grub2-install partition
```

partition 을 이전 단계에서 찾은 GRUB 파티션 (예:**/dev/sda1**)으로 바꿉니다.

3. 변경 사항을 적용하려면 시스템을 재부팅합니다.

```
# reboot
```

추가 리소스

- **grub-install(1)** 매뉴얼 페이지

11.4. GRUB 재설정

GRUB을 재설정하면 모든 GRUB 설정 파일 및 시스템 설정이 완전히 제거되고 부트로더를 다시 설치합니다. 모든 구성 설정을 기본값으로 재설정하여 손상된 파일과 잘못된 구성으로 인한 오류를 수정할 수 있습니다.

중요

다음 절차에서는 사용자가 수행한 모든 사용자 지정을 제거합니다.

절차

1. 구성 파일을 제거합니다.

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

2. 패키지를 다시 설치합니다.
   - BIOS 기반 시스템에서 다음을 입력합니다.

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

   - UEFI 기반 시스템에서 다음을 입력합니다.

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

3. 변경 사항을 적용하려면 **grub.cfg** 파일을 다시 빌드합니다.
   - BIOS 기반 시스템에서 다음을 입력합니다.

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

   - UEFI 기반 시스템에서 다음을 입력합니다.

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

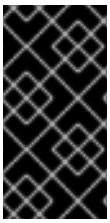
4. GRUB 설치 절차에 따라 **/boot/** 파티션에서 GRUB을 복원하십시오.

12장. KDUMP 설치

새로운 Red Hat Enterprise Linux 9 설치 버전에서 **kdump** 서비스가 기본적으로 설치 및 활성화됩니다. 제공된 정보 및 절차를 통해 **kdump** 가 무엇이며 **kdump** 가 기본적으로 활성화되어 있지 않은 경우 설치 방법을 알아보십시오.

12.1. KDUMP란?

kdump 는 크래시 덤프 메커니즘을 제공하고 크래시 덤프 또는**vmcore** 파일이라는 덤프 파일을 생성하는 서비스입니다. **vmcore** 파일에는 분석 및 문제 해결에 도움이 되는 시스템 메모리 내용이 있습니다**kdump** 는 **kexec** 시스템 호출을 사용하여 두 번째 커널로 부팅하고 재부팅 없이*캡처 커널*을 캡처한 다음 충돌된 커널의 메모리 내용을 캡처하여 파일에 저장합니다. 두 번째 커널은 시스템 메모리의 예약된 부분에서 사용할 수 있습니다.



중요

커널 크래시 덤프는 시스템 오류가 발생한 경우 사용 가능한 유일한 정보일 수 있습니다. 따라서 운영 **kdump** 는 미션 크리티컬한 환경에서 중요합니다. Red Hat은 일반 커널 업데이트 주기에서 **kexec-tools** 를 정기적으로 업데이트하고 테스트하는 것이 좋습니다. 이는 새 커널 기능을 설치할 때 특히 중요합니다.

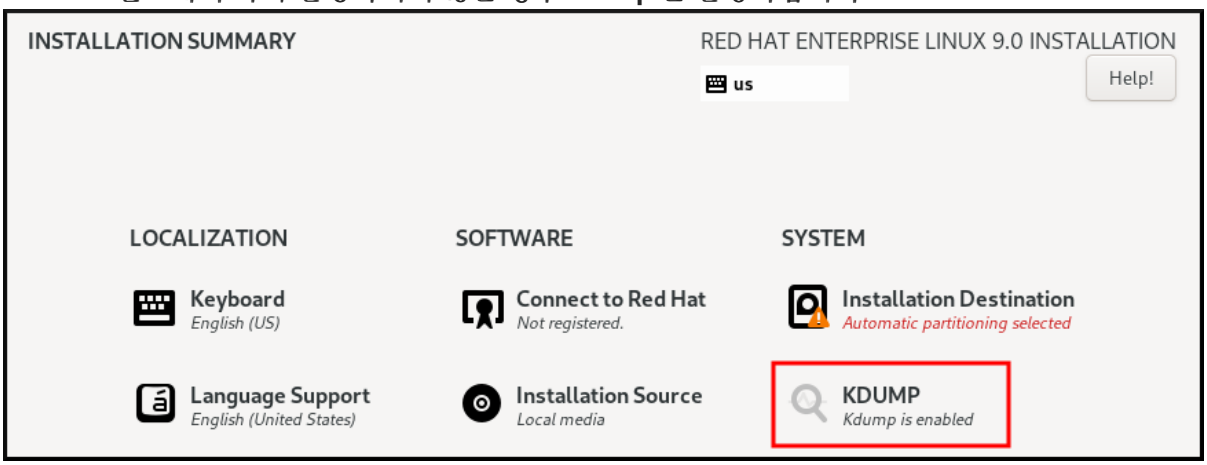
머신의 모든 설치된 커널 또는 지정된 커널의 경우에만 **kdump** 를 활성화할 수 있습니다. 이 기능은 시스템에서 여러 개의 커널이 사용되는 경우 유용합니다. 그 중 일부는 충돌할 수 있는 우려가 없을 정도로 안정적입니다. **kdump** 를 설치하면 기본**/etc/kdump.conf** 파일이 생성됩니다.**/etc/kdump.conf** 파일에는 **kdump** 설정을 사용자 지정하도록 편집할 수 있는 기본 최소 **kdump** 구성이 포함되어 있습니다.

12.2. ANACONDA를 사용하여 KDUMP 설치

Anaconda 설치 프로그램은 대화형 설치 중에**kdump** 구성에 대한 그래픽 인터페이스 화면을 제공합니다. 설치 프로그램 화면의 이름은 KDUMP 로 지정되어 있으며 기본설치 요약 화면에서 사용할 수 있습니다. **kdump** 를 활성화하고 필요한 양의 메모리를 예약할 수 있습니다.

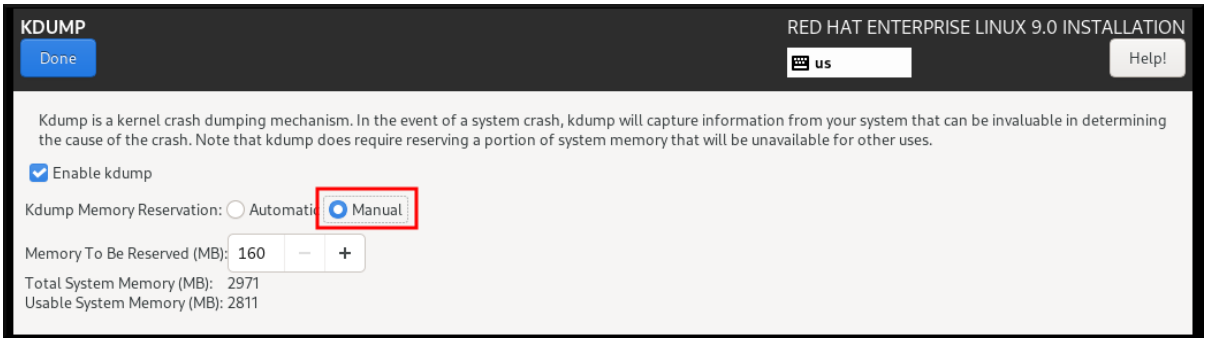
절차

1. KDUMP 필드에서 아직 활성화되지 않은 경우**kdump** 를 활성화합니다.



2. Kdump 메모리 예약 에서 메모리 예약을 사용자 지정해야 하는 경우Manual'을 선택합니다.

3. KDUMP 필드에서 KDUMP 필드에서 **kdump** 에 필요한 메모리 예약을 설정합니다.



12.3. 명령줄에 KDUMP 설치

사용자 지정 Kickstart 설치와 같은 일부 설치 옵션은 기본적으로**kdump** 를 설치하거나 활성화하지 않는 경우가 있습니다. 이 경우 아래 절차를 따르십시오.

사전 요구 사항

- 활성 RHEL 서브스크립션입니다.
- 시스템 CPU 아키텍처용 **kexec-tools** 패키지가 포함된 리포지토리입니다.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상을 참조하십시오.

절차

1. **kdump** 가 시스템에 설치되어 있는지 확인합니다.

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

패키지가 설치된 경우 출력됩니다.

```
# kexec-tools-2.0.22-13.el9.x86_64
```

패키지가 설치되지 않은 경우 출력됩니다.

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

2. 다음을 통해 **kdump** 및 기타 필요한 패키지를 설치합니다.

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

13장. 명령줄에서 KDUMP 구성

kdump 의 메모리는 시스템 부팅 중에 예약되어 있습니다. 메모리 크기는 시스템의 GRUB(GRUB) 구성 파일에서 구성됩니다. 메모리 크기는 구성 파일에 지정된 **crashkernel=** 값과 시스템의 실제 메모리에 따라 다릅니다.

13.1. KDUMP 크기 추정

kdump 환경을 계획하고 구축할 때는 크래시 덤프 파일에 필요한 공간의 양을 아는 것이 중요합니다.

makedumpfile --mem-usage 명령은 크래시 덤프 파일에 필요한 공간을 추정합니다. 메모리 사용량 보고서를 생성합니다. 이 보고서는 덤프 수준과 제외할 페이지를 결정하는 데 도움이 됩니다.

절차

- 다음 명령을 실행하여 메모리 사용량 보고서를 생성합니다.

```
# 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
```

중요

makedumpfile --mem-usage 명령은 필요한 메모리를 페이지에 보고합니다. 즉, 커널 페이지 크기에 대해 사용 중인 메모리 크기를 계산해야 합니다.

기본적으로 RHEL 커널은 IBM POWER 아키텍처에서 AMD64 및 Intel 64 CPU 아키텍처에서 4KB 크기의 페이지를 사용합니다.

13.2. RHEL 9에서 KDUMP 메모리 사용량 구성

kexec-tools 패키지는 기본 **crashkernel=** 메모리 예약 값을 유지 관리합니다.**kdump** 서비스는 기본값을 사용하여 각 커널에 대해 크래시 커널 메모리를 예약합니다. 기본값은 **crashkernel=** 값을 수동으로 설정할 때 필요한 메모리 크기를 추정하는 참조 기본 값으로도 사용할 수 있습니다. 크래시 커널의 최소 크기는 하드웨어 및 머신 사양에 따라 다를 수 있습니다.

kdump 의 자동 메모리 할당은 시스템 하드웨어 아키텍처 및 사용 가능한 메모리 크기에 따라 다릅니다. 예를 들어 AMD 및 Intel 64비트 아키텍처에서는 사용 가능한 메모리가 1GB를 초과하는 경우에만 **crashkernel=** 매개변수의 기본값이 작동합니다. **kexec-tools** 유틸리티는 AMD64 및 Intel 64비트 아키텍처에 다음과 같은 기본 메모리를 구성합니다.

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

kdumpctl 추정치를 실행하여 충돌을 트리거하지 않고 대략적인 추정 값을 쿼리할 수도 있습니다. 추정된 **crashkernel=** 값은 정확하지 않을 수 있지만 적절한**crashkernel=** 값을 설정하는 참조 역할을 할 수 있습니다.

참고

부팅 명령줄의 **crashkernel=auto** 옵션은 RHEL 9 이상 릴리스에서 더 이상 지원되지 않습니다.

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.
- 구성 및 대상에 대한 **kdump** 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상 을 참조하십시오.
- **zipl** 유틸리티가 IBM Z 시스템인 경우 설치했습니다.

절차

1. 크래시 커널의 기본값을 구성합니다.

```
# kdumpctl reset-crashkernel --kernel=ALL
```

crashkernel= 값을 구성할 때**kdump** 가 활성화된 상태로 재부팅하여 구성을 테스트합니다. **kdump** 커널이 부팅되지 않으면 메모리 크기를 점진적으로 늘려 허용 가능한 값을 설정합니다.

2. 사용자 지정 **crashkernel=** 값을 사용하려면 다음을 수행합니다.

   a. 필요한 메모리 예약을 구성합니다.

```
crashkernel=192M
```

   또는 **crashkernel= <range1>:<size*1*>,<range2>:<size2>:<size2>** *구문을 사용하여 설치된 메모리의 총 크기에* 따라 예약된 메모리 양을 변수로 설정할 수 있습니다. 예를 들면 다음과 같습니다.

```
crashkernel=1G-4G:192M,2G-64G:256M
```

   이 예제에서는 총 시스템 메모리 양이 1GB 이상이고 4GB보다 낮은 경우 192MB의 메모리를 예약합니다. 총 메모리 양이 4GB를 초과하면 **kdump** 용으로 256MB가 예약되어 있습니다.

   b. (선택 사항) 예약된 메모리를 끕니다.
   일부 시스템은 crashkernel 예약이 매우 초기이므로 특정 고정 오프셋이 있는 메모리를 예약해야 하며 특수 사용을 위해 일부 영역을 예약하려고 합니다. 오프셋이 설정되면 예약된 메모리가 여기에서 시작됩니다. 예약된 메모리를 오프셋하려면 다음 구문을 사용합니다.

```
crashkernel=192M@16M
```

   이 예제에서는 16MB(실제 주소 0x01000000)부터 192MB의 메모리를 예약합니다. 0으로 오프셋하거나 값을 지정하지 않으면 **kdump** 에서 예약된 메모리를 자동으로 오프셋합니다. 오프셋을 마지막 값으로 지정하여 변수 메모리 예약을 설정할 때 메모리를 오프셋할 수도 있습니다. 예를 들어 **crashkernel=1G-4G:192M,2G-64G:256M@16M**.

   c. 부트 로더 구성을 업데이트합니다.

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

& *lt;custom-value* >에는 크래시 커널에 대해 구성한 사용자 정의**crashkernel=** 값이 포함되어야 합니다.

3. 변경 사항을 적용하려면 재부팅합니다.

```
# reboot
```

검증

sysrq 키를 활성화하여 커널이 충돌하도록 합니다.**address-YYY-MM-DD-HH:MM:SS/vmcore** 파일은 **/etc/kdump.conf** 파일에 지정된 대로 대상 위치에 저장됩니다. 기본 대상 위치를 선택하면**vmcore** 파일은 **/var/crash/** 에 마운트된 파티션에 저장됩니다.

주의

kdump 구성을 테스트하는 명령으로 인해 커널이 데이터 손실과 충돌합니다. 신중하게 지침에 따라 kdump 구성을 테스트하는 데 활성 프로덕션 시스템을 사용하지 마십시오.

1. **kdump** 커널에 부팅하려면 **sysrq** 키를 활성화합니다.

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

명령을 사용하면 커널이 충돌하고 필요한 경우 커널을 재부팅합니다.

2. **/etc/kdump.conf** 파일을 표시하고 **vmcore** 파일이 대상 대상에 저장되는지 확인합니다.

추가 리소스

- 시스템이 부팅되기 전에 grub에서 부팅 매개 변수를 수동으로 수정하는 방법
- **grubby(8)** 도움말 페이지

13.3. KDUMP 대상 구성

크래시 덤프는 일반적으로 로컬 파일 시스템에 파일로 저장되며 장치에 직접 작성됩니다. 또는 **NFS** 또는 **SSH** 프로토콜을 사용하여 네트워크를 통해 전송되는 크래시 덤프를 설정할 수 있습니다. 크래시 덤프 파일을 유지하기 위해 이러한 옵션 중 하나만 한 번에 설정할 수 있습니다. 기본 동작은 로컬 파일 시스템의 **/var/crash/** 디렉터리에 저장하는 것입니다.

사전 요구 사항

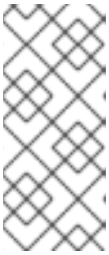
- 시스템에 대한 root 권한이 있습니다.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상을 참조하십시오.

절차

- 크래시 덤프 파일을 로컬 파일 시스템의 **/var/crash/** 디렉터리에 저장하려면 **/etc/kdump.conf** 파일을 편집하고 경로를 지정합니다.

```
path /var/crash
```

옵션 경로 **/var/crash** 는 **kdump** 가 크래시 덤프 파일을 저장하는 파일 시스템의 경로를 나타냅니다.



참고

- **/etc/kdump.conf** 파일에서 덤프 대상을 지정하면 경로는 지정된 덤프 대상을기준으로 합니다.
- **/etc/kdump.conf** 파일에 덤프 대상을 지정하지 않으면 경로는 루트 디렉터리의 절대 경로를 나타냅니다.

현재 시스템에 마운트된 항목에 따라 덤프 대상 및 조정된 덤프 경로가 자동으로 수행됩니다.

크래시 덤프 파일과 **kdump** 에서 생성된 관련 파일을 보호하려면 사용자 권한 및 SELinux 컨텍스트와 같은 대상 대상 디렉터리에 대한 적절한 속성을 설정해야 합니다. 또한 다음과 같이 **kdump.conf** 파일에서 **kdump_post.sh** 스크립트를 정의할 수 있습니다.

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

kdump_post 지시문은 **kdump** 에서 캡처하고 지정된 대상에 크래시 덤프를 저장한후 실행되는 쉘 스크립트 또는 명령을 지정합니다. 이 메커니즘을 사용하여 **kdump** 의 기능을 확장하여 파일 권한 조정을 포함하여 작업을 수행할 수 있습니다.

예 13.1. **kdump** 대상 설정

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

여기서 덤프 대상(**ext4 /dev/mapper/vg00-varcrashvol**)을 지정하므로 **/var/crash** 에 마운트됩니다. **path** 옵션도 **/var/crash** 로 설정되므로 **kdump** 는 **/var/crash/var/crash** 디렉터리에 **vmcore** 파일을 저장합니다.

- 크래시 덤프를 저장할 로컬 디렉터리를 **root** 로 변경하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
  a. **#path /var/crash** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
  b. 값을 의도된 디렉터리 경로로 바꿉니다. 예를 들면 다음과 같습니다.

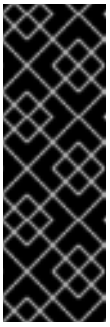
```
path /usr/local/cores
```

중요

Red Hat Enterprise Linux 9에서는 **kdump systemd** 서비스가 실패를 방지할 때 **path** 지시문을 사용하여 **kdump** 대상으로 정의된 디렉터리가 있어야 합니다. 이 동작은 서비스가 시작될 때 없는 경우 디렉터리가 자동으로 생성되는 RHEL 이전 버전과 다릅니다.

- 파일을 다른 파티션에 작성하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
  a. 선택한 항목에 따라 **#ext4** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
    - 장치 이름 (**#ext4 /dev/vg/lv_kdump** 행)
    - 파일 시스템 레이블 (**#ext4 LABEL=/boot** 행)
    - UUID (**#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** 줄)
  b. 파일 시스템 유형과 장치 이름, 레이블 또는 UUID를 필수 값으로 변경합니다. UUID 값을 지정하는 올바른 구문은 **UUID="correct-uuid"** 및 **UUID=correct-uuid** 입니다. 예를 들면 다음과 같습니다.

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



중요

LABEL= 또는 **UUID=** 를 사용하여 스토리지 장치를 지정하는 것이 좋습니다. **/dev/sda3** 과 같은 디스크 장치 이름은 재부팅 후에도 일관되게 보장되지 않습니다.

IBM Z 하드웨어에서 Direct Access Storage Device(DASD)를 사용하는 경우 **kdump** 를 진행하기 전에 덤프 장치가**/etc/dasd.conf** 에 올바르게 지정되었는지 확인합니다.

- 크래시 덤프를 장치에 직접 작성하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
  a. **#raw /dev/vg/lv_kdump** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
  b. 값을 원하는 장치 이름으로 바꿉니다. 예를 들면 다음과 같습니다.

```
raw /dev/sdb1
```

- **NFS** 프로토콜을 사용하여 크래시 덤프를 원격 시스템에 저장하려면 다음을 수행합니다.
  a. **#nfs my.server.com:/export/tmp** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
  b. 값을 유효한 호스트 이름 및 디렉터리 경로로 바꿉니다. 예를 들면 다음과 같습니다.

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

  c. 변경 사항을 적용하려면 **kdump** 서비스를 다시 시작하십시오.

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

참고

NFS 지시문을 사용하여 NFS 대상을 지정하는 경우 **kdump.service** 는 자동으로 NFS 대상을 마운트하여 디스크 공간을 확인합니다. NFS 대상을 사전에 마운트할 필요가 없습니다. **kdump.service** 가 대상을 마운트하지 않도록 kdump.conf에서 **dracut_args --mount** 지시문을 사용하여 **kdump.conf** 에서 **kdump.service** 가 **--mount** 인수를 사용하여 **dracut** 유틸리티를 호출하여 NFS 대상을 지정합니다.

- SSH 프로토콜을 사용하여 크래시 덤프를 원격 시스템에 저장하려면 다음을 수행합니다.
  a. **#ssh user@my.server.com** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
  b. 값을 유효한 사용자 이름 및 호스트 이름으로 바꿉니다.
  c. 구성에 SSH 키를 포함합니다.
    i. **#sshkey /root/.ssh/kdump_id_rsa** 줄에서 해시 기호를 제거합니다.
    ii. 덤프하려는 서버에서 유효한 키 위치로 값을 변경합니다. 예를 들면 다음과 같습니다.

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

추가 리소스

13.8절. “시스템 충돌 후 kdump에 의해 생성된 파일”

13.4. KDUMP 코어 수집기 구성

kdump 서비스는 **core_collector** 프로그램을 사용하여 크래시 덤프 이미지를 캡처합니다. RHEL에서 **makedumpfile** 유틸리티는 기본 코어 수집기입니다. 다음과 같이 덤프 파일을 줄이는 데 도움이 됩니다.

- 크래시 덤프 파일의 크기를 압축하고 다양한 덤프 수준을 사용하여 필요한 페이지만 복사합니다.
- 불필요한 크래시 덤프 페이지 제외.
- 크래시 덤프에 포함될 페이지 유형 필터링.

구문

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

옵션

- **-c , -l** 또는 **-p:, zlib** for **-c** 옵션을 사용하여 각 페이지에서 덤프 파일 형식을 압축하거나 **-l** 옵션에 대해 **lzo** 또는 **-p** 옵션을 위해 **snappy** 를 지정합니다.
- **-d (dump_level)**: 덤프 파일에 복사되지 않도록 페이지를 제외합니다.
- **--message-level** : 메시지 유형을 지정합니다. 이 옵션으로**message_level** 을 지정하여 출력된 출력을 제한할 수 있습니다. 예를 들어 7을 **message_level** 으로 지정하면 일반적인 메시지 및 오류 메시지가 출력됩니다. **message_level** 의 최대값은 31입니다.

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상 을 참조하십시오.

절차

1. **root** 로서 **/etc/kdump.conf** 구성 파일을 편집하고**#core_collector makedumpfile -l --message-level 1 -d 31** 의 시작 부분에서 해시 기호("#")를 제거합니다.
2. 크래시 덤프 파일 압축을 활성화하려면 다음을 실행합니다.

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

l 옵션은 덤프 압축 파일 형식을 지정합니다.**d** 옵션은 덤프 수준을 31로 지정합니다.**--message-level** 옵션은 메시지 수준을 1로 지정합니다.

또한 **-c** 및 **-p** 옵션을 사용하여 다음 예제를 고려하십시오.

- **-c** 를 사용하여 크래시 덤프 파일을 압축하려면 다음을 수행합니다.

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

- 크래시 덤프 파일을 압축하려면 **-p**:

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

추가 리소스

- **makedumpfile(8)** 매뉴얼 페이지
- kdump 설정 파일

13.5. KDUMP 기본 오류 응답 구성

기본적으로 **kdump** 가 구성된 대상 위치에서 크래시 덤프 파일을 생성하지 못하면 시스템이 재부팅되고 프로세스에서 덤프가 손실됩니다. 기본 실패 응답을 변경하고 코어 덤프를 기본 대상에 저장하지 못하는 경우 다른 작업을 수행하도록 **kdump** 를 구성할 수 있습니다. 추가 작업은 다음과 같습니다.

dump_to_rootfs

코어 덤프를 루트 파일 시스템에 저장합니다.

reboot

시스템을 재부팅하여 프로세스의 코어 덤프가 손실됩니다.

halt

시스템에서 시스템을 중지하고 프로세스에서 코어 덤프를 손실합니다.

poweroff

시스템의 전원을 끄고 프로세스의 코어 덤프를 끊습니다.

shell

initramfs 내에서 쉘 세션을 실행하면 코어 덤프를 수동으로 기록할 수 있습니다.

final_action

kdump 에 성공한 후 또는 shell 또는**dump_to_rootfs** 실패 작업이 완료될 때**reboot**,**halt**, **poweroff** 와 같은 추가 작업을 활성화합니다. 기본값은 **reboot** 입니다.

failure_action

커널 충돌에서 덤프가 실패할 때 수행할 작업을 지정합니다. 기본값은 **reboot** 입니다.

사전 요구 사항

- 루트 권한.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상을 참조하십시오.

절차

1. **root** 로서 **/etc/kdump.conf** 구성 파일의 **#failure_action** 행의 시작 부분에서 해시 기호(**#**)를 제거합니다.
2. 값을 원하는 작업으로 바꿉니다.

```
failure_action poweroff
```

추가 리소스

- kdump 대상 구성

13.6. KDUMP 설정 파일

kdump 커널의 설정 파일은**/etc/sysconfig/kdump** 입니다. 이 파일은**kdump** 커널 명령줄 매개 변수를 제어합니다. 대부분의 구성의 경우 기본 옵션을 사용합니다. 그러나 일부 시나리오에서는 **kdump** 커널 동작을 제어하도록 특정 매개변수를 수정해야 할 수 있습니다. 예를 들어 **KDUMP_COMMANDLINE_APPEND** 옵션을 수정하여 **kdump** 커널 명령줄을 추가하여 자세한 디버깅 출력을 얻거나**kdump** 명령줄에서 인수를 제거하도록 **KDUMP_COMMANDLINE_REMOVE** 옵션을 변경합니다.

KDUMP_COMMANDLINE_REMOVE

이 옵션은 현재 **kdump** 명령줄에서 인수를 제거합니다.**kdump** 오류 또는**kdump** 커널 부팅 오류를 일으킬 수 있는 매개변수를 제거합니다. 이러한 매개 변수는 이전 **KDUMP_COMMANDLINE** 프로세스에서 구문 분석되었거나 **/proc/cmdline** 파일에서 상속되었을 수 있습니다.
이 변수가 구성되지 않은 경우 **/proc/cmdline** 파일의 모든 값을 상속합니다. 이 옵션을 구성하면 문제를 디버깅하는 데 유용한 정보도 제공합니다.

특정 인수를 제거하려면 다음과 같이 **KDUMP_COMMANDLINE_REMOVE** 에 추가합니다.

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

KDUMP_COMMANDLINE_APPEND

이 옵션은 현재 명령줄에 인수를 추가합니다. 이러한 인수는 이전 **KDUMP_COMMANDLINE_REMOVE** 변수에서 구문 분석할 수 있습니다.
kdump 커널의 경우**mce**,**cgroup**,**numa**,**hest_disable** 와 같은 특정 모듈을 비활성화하면 커널 오류가 발생하지 않도록 할 수 있습니다. 이러한 모듈은 **kdump** 용으로 예약된 커널 메모리의 상당한 부분을 사용하거나 **kdump** 커널 부팅 오류가 발생할 수 있습니다.

kdump 커널 명령줄에서 메모리 **cgroup** 을 비활성화하려면 다음과 같이 명령을 실행합니다.

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

추가 리소스

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

13.7. KDUMP 설정 테스트

kdump 를 구성한 후 시스템 충돌을 수동으로 테스트하고**vmcore** 파일이 정의된 **kdump** 대상에 생성되었는지 확인해야 합니다. **vmcore** 파일은 새로 부팅된 커널의 컨텍스트에서 캡처되므로 커널 충돌을 디버깅하는 데 도움이 되는 중요한 정보가 있습니다.

주의

활성 프로덕션 시스템에서 **kdump** 를 테스트하지 마십시오.**kdump** 를 테스트하는 명령으로 인해 커널이 데이터 손실과 충돌합니다. 시스템 아키텍처에 따라 **kdump** 테스트에서 부팅 시간이 긴 몇 가지 재부팅이 필요할 수 있으므로 상당한 유지 관리 시간을 예약해야 합니다.

kdump 테스트 중에 **vmcore** 파일이 생성되지 않으면 **kdump** 테스트를 위해 테스트를 다시 실행하기 전에 문제를 식별하고 수정합니다.

중요

kdump 테스트에는 부팅 시간이 긴 몇 가지 재부팅이 필요할 수 있으므로 상당한 유지 관리 시간을 예약해야 합니다.

수동 시스템을 수정하는 경우 시스템 수정이 끝날 때 **kdump** 구성을 테스트해야 합니다. 예를 들어 다음 변경 사항을 수행하는 경우 **kdump** 구성에 최적의 **kdump** 성능을 테스트해야 합니다.

- 패키지 업그레이드.
- 하드웨어 수준 변경(예: 스토리지 또는 네트워킹 변경)
- 펌웨어 및 BIOS 업그레이드.
- 타사 모듈을 포함하는 새로운 설치 및 애플리케이션 업그레이드.
- 핫플러그 메커니즘을 사용하여 이 메커니즘을 지원하는 하드웨어에 메모리를 추가하는 경우.
- **/etc/kdump.conf** 또는 **/etc/sysconfig/kdump** 파일을 변경한 후

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.
- 모든 중요한 데이터를 저장했습니다. **kdump** 를 테스트하는 명령으로 인해 커널이 데이터 손실과 충돌합니다.
- 시스템 아키텍처에 따라 상당한 머신 유지 관리 시간을 예약했습니다.

절차

1. **kdump** 서비스를 활성화합니다.

```
# kdumpctl restart
```

2. **kdump** 서비스의 상태를 확인합니다. **kdumpctl** 명령을 사용하면 콘솔에서 출력을 출력할 수 있습니다.

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

또는 **systemctl** 명령을 사용하는 경우 출력은 **systemd** 저널에 출력됩니다.

3. 커널 충돌을 시작하여 **kdump** 구성을 테스트합니다. **sysrq-trigger** 키 조합을 사용하면 커널이 충돌하고 필요한 경우 시스템을 재부팅할 수 있습니다.

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

커널 재부팅 시 *주소-YYYY-MM-DD-HH:MM:SS*/vmcore 파일이 /etc/kdump.conf 파일에 지정된 위치에 생성됩니다. 기본값은 /var/crash/ 입니다.

추가 리소스

- kdump 대상 구성

13.8. 시스템 충돌 후 KDUMP에 의해 생성된 파일

시스템 충돌 후 **kdump** 서비스는 덤프 파일(vmcore)에 커널 메모리를 캡처하고 문제 해결 및 제거 후 분석을 지원하는 추가 진단 파일도 생성합니다.

kdump 에 의해 생성된 파일:

- **vmcore** - 충돌 시 시스템 메모리를 포함하는 주요 커널 메모리 덤프 파일입니다**kdump** 구성에 지정된 **core_collector** 프로그램의 구성에 따라 데이터를 포함합니다. 기본적으로 커널 데이터 구조, 프로세스 정보, 스택 추적 및 기타 진단 정보.
- **vmcore-dmesg.txt** - 패닉 상태인 기본 커널의 커널 링 버퍼 로그(**dmesg**)의 내용입니다.
- **kexec-dmesg.log** - **vmcore** 데이터를 수집하는 보조 **kexec** 커널 실행의 커널 및 시스템 로그 메시지를 포함합니다.

추가 리소스

- 커널 링 버퍼란
- kdump란 무엇입니까?

13.9. KDUMP 서비스 활성화 및 비활성화

특정 커널 또는 설치된 모든 커널에서 **kdump** 기능을 활성화하거나 비활성화하도록 구성할 수 있습니다. **kdump** 기능을 정기적으로 테스트하고 제대로 작동하는지 확인해야 합니다.

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.
- 구성 및 대상에 대한 **kdump** 요구 사항을 완료했습니다. 지원되는 kdump 구성 및 대상을 참조하십시오.
- **kdump** 설치에 대한 모든 구성은 필요에 따라 설정됩니다.

절차

- **multi-user.target** 에 대해 **kdump** 서비스를 활성화합니다.

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

- 현재 세션에서 서비스를 시작합니다.

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

- **kdump** 서비스를 중지합니다.

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

- **kdump** 서비스를 비활성화합니다.

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

주의

kptr_restrict=1 을 기본값으로 설정하는 것이 좋습니다. **kptr_restrict** 가 기본적으로 (1)로 설정된 경우 **kdumpctl** 서비스는 KASLR(커널 주소 공간 레이아웃)이 활성화되거나 활성화되지 않은 경우에도 크래시 커널을 로드합니다.

kptr_restrict 가 **1** 로 설정되지 않고 KASLR이 활성화된 경우 **/proc/kore** 파일의 콘텐츠가 모든 0으로 생성됩니다. **kdumpctl** 서비스가 **/proc/kcore** 파일에 액세스하지 못하고 크래시 커널을 로드합니다. **kexec-kdump-howto.txt** 파일에는 **kptr_restrict=1** 을 설정할 것을 권장하는 경고 메시지가 표시됩니다. **sysctl.conf** 파일에서 다음을 확인하여 **kdumpctl** 서비스가 크래시 커널을 로드하는지 확인합니다.

- **sysctl.conf** 파일의 커널 **kptr_restrict=1**.

13.10. 커널 드라이버가 KDUMP에 대한 로드되지 않음

/etc/sysconfig/kdump 구성 파일에 **KDUMP_COMMANDLINE_APPEND=** 변수를 추가하여 캡처 커널이

특정 커널 드라이버를 로드하지 않도록 제어할 수 있습니다. 이 방법을 사용하면 **kdump** 초기 RAM 디스크 이미지 **initramfs** 가 지정된 커널 모듈을 로드하지 못하도록 할 수 있습니다. 이렇게 하면 OOM(메모리 부족) 중단 오류 또는 기타 크래시 커널 실패를 방지할 수 있습니다.

다음 구성 옵션 중 하나를 사용하여 **KDUMP_COMMANDLINE_APPEND=** 변수를 추가할 수 있습니다.

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

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.

절차

1. 현재 실행 중인 커널에 로드된 모듈 목록을 표시합니다. 로드에서 차단할 커널 모듈을 선택합니다.

```
$ lsmod

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

2. **/etc/sysconfig/kdump** 파일에서 **KDUMP_COMMANDLINE_APPEND=** 변수를 업데이트합니다. 예를 들면 다음과 같습니다.

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

또한 **modprobe.blacklist= <*modules*>** 구성 옵션을 사용하는 다음 예제도 고려하십시오.

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

3. **kdump** 서비스를 다시 시작하십시오.

```
# systemctl restart kdump
```

추가 리소스

- **dracut.cmdline** 도움말 페이지

13.11. 암호화된 디스크가 있는 시스템에서 KDUMP 실행

LUKS 암호화된 파티션을 실행할 때 시스템에 특정 양의 사용 가능한 메모리가 필요합니다. 시스템에 필요한 양의 메모리보다 적은 경우 **cryptsetup** 유틸리티에서 파티션을 마운트하지 못합니다. 결과적으로 두 번째 커널(capture 커널)에서 **vmcore** 파일을 암호화된 대상 위치로 캡처할 수 없습니다.

kdumpctl estimate 명령은 kdump .**kdump** ctl 추정에 필요한 메모리 양을 추정하는 데 도움이 됩니다.**kdumpctl** 추정은 **kdump** 에 필요한 메모리 크기에 가장 적합한크래시커널 값을 출력합니다.

권장되는 **crashkernel** 값은 현재 커널 크기, 커널 모듈, initramfs 및 LUKS 암호화된 대상 메모리 요구 사항을 기반으로 계산됩니다.

사용자 정의 **crashkernel=** 옵션을 사용하는 경우**kdumpctl** 추정은 **LUKS** 필요한 크기 값을 출력합니다. 값은 LUKS 암호화된 대상에 필요한 메모리 크기입니다.

절차

1. 추정치 **crashkernel=** 값을 출력합니다.

```
# *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. **crashkernel=** 값을 늘려 필요한 메모리 양을 구성합니다.

3. 시스템을 재부팅합니다.

참고

kdump 서비스가 여전히 덤프 파일을 암호화된 대상에 저장하지 못하는 경우 필요에 따라 **crashkernel=** 값을 늘립니다.

14장. KDUMP 활성화

Red Hat Enterprise Linux 9 시스템의 경우 특정 커널 또는 설치된 모든 커널에서 **kdump** 기능을 활성화하거나 비활성화하도록 구성할 수 있습니다. 그러나 **kdump** 기능을 정기적으로 테스트하고 제대로 작동하는지 확인해야 합니다.

14.1. 설치된 모든 커널에 KDUMP 활성화

kdump 서비스는 **kexec** 툴이 설치된 후 **kdump.service** 를 활성화하여 시작합니다. 머신에 설치된 모든 커널에 대해 **kdump** 서비스를 활성화하고 시작할 수 있습니다.

사전 요구 사항

- 관리자 권한이 있습니다.

절차

1. **crashkernel=** 명령줄 매개 변수를 설치된 모든 커널에 추가합니다.

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

   XXM 은 필요한 메모리(MB)입니다.

2. **kdump** 서비스를 활성화합니다.

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

검증

- **kdump** 서비스가 실행 중인지 확인합니다.

```
# 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)
```

14.2. 설치된 특정 커널에 대해 KDUMP 활성화

머신에서 특정 커널에 대해 **kdump** 서비스를 활성화할 수 있습니다.

사전 요구 사항

- 관리자 권한이 있습니다.

절차

1. 시스템에 설치된 커널을 나열합니다.

```
# 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. 시스템의 GRUB(GRUB) 설정에 특정 **kdump** 커널을 추가합니다.
예를 들면 다음과 같습니다.

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

XXM 은 필요한 메모리 예약(MB)입니다.

3. **kdump** 서비스를 활성화합니다.

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

검증

- **kdump** 서비스가 실행 중인지 확인합니다.

```
# 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)
```

14.3. KDUMP 서비스 비활성화

kdump.service 를 중지하고 Red Hat Enterprise Linux 9 시스템에서 서비스가 시작되지 않도록 비활성화할 수 있습니다.

사전 요구 사항

- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은지원되는 kdump 구성 및 대상을 참조하십시오.
- **kdump** 설치의 모든 구성은 필요에 따라 설정됩니다. 자세한 내용은kdump 설치를 참조하십시오.

절차

1. 현재 세션에서 **kdump** 서비스를 중지하려면 다음을 수행합니다.

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

2. **kdump** 서비스를 비활성화하려면 다음을 수행합니다.

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

주의

kptr_restrict=1 을 기본값으로 설정하는 것이 좋습니다.**kptr_restrict** 를 기본값으로 설정하면 **kdumpctl** 서비스가 커널 주소 공간 레이아웃(**KASLR**)이 활성화되거나 활성화되지 않은 경우에도 크래시 커널을 로드합니다.

kptr_restrict 가 **1** 로 설정되지 않고 **KASLR** 이 활성화된 경우 **/proc/kore** 파일의 콘텐츠가 모든 0으로 생성됩니다. **kdumpctl** 서비스가 **/proc/kcore** 파일에 액세스하지 못하고 크래시 커널을 로드합니다. **kexec-kdump-howto.txt** 파일에는 **kptr_restrict=1** 을 설정할 것을 권장하는 경고 메시지가 표시됩니다. **sysctl.conf** 파일에서 다음을 확인하여 **kdumpctl** 서비스가 크래시 커널을 로드하는지 확인합니다.

- **sysctl.conf** 파일의 커널 **kptr_restrict=1**.

추가 리소스

- systemd 관리

15장. 지원되는 KDUMP 구성 및 대상

kdump 메커니즘은 커널 충돌 발생 시 크래시 덤프 파일을 생성하는 Linux 커널의 기능입니다. 커널 덤프 파일에는 커널 충돌의 근본 원인을 분석하고 결정하는 데 도움이 되는 중요한 정보가 있습니다. 충돌은 다양한 요인, 하드웨어 문제 또는 타사 커널 모듈 문제로 인해 몇 가지 이름을 지정할 수 있습니다.

제공된 정보와 절차를 통해 Red Hat Enterprise Linux 9 시스템에서 지원되는 구성 및 대상을 이해하고 **kdump** 를 올바르게 구성하고 작동하는지 검증할 수 있습니다.

15.1. KDUMP의 메모리 요구 사항

kdump 에서 커널 크래시 덤프를 캡처하고 추가 분석을 위해 저장하려면 시스템 메모리의 일부를 캡처 커널에 영구적으로 예약해야 합니다. 예약된 경우 시스템 메모리의 이 부분을 기본 커널에서 사용할 수 없습니다.

메모리 요구 사항은 특정 시스템 매개변수에 따라 다릅니다. 가장 중요한 요소 중 하나는 시스템의 하드웨어 아키텍처입니다. 정확한 머신 아키텍처 (예: Intel 64 및 AMD64, x86_64라고도 함)를 찾아서 표준 출력으로 출력하려면 다음 명령을 사용하십시오.

```
$ uname -m
```

명시된 최소 메모리 요구 사항 목록을 사용하여 사용 가능한 최신 버전에서 **kdump** 의 메모리를 자동으로 예약하도록 적절한 메모리 크기를 설정할 수 있습니다. 메모리 크기는 시스템의 아키텍처 및 사용 가능한 실제 메모리에 따라 다릅니다.

표 15.1. kdump에 필요한 최소 예약된 메모리 양

아키텍처	사용 가능한 메모리	예약된 최소 메모리
AMD64 and Intel 64 (**x86_64**)	1GB에서 4GB	192MB RAM
	4GB ~ 64GB	256MB RAM
	64GB 이상	512MB RAM
64비트 ARM (4k 페이지)	1GB에서 4GB	256MB RAM
	4GB에서 64GB	320MB의 RAM
	64GB 이상	576MB의 RAM
64비트 ARM(64k 페이지)	1GB에서 4GB	356MB의 RAM
	4GB에서 64GB	420MB의 RAM
	64GB 이상	676MB의 RAM
IBM Power Systems(**ppc64le**)	2GB에서 4GB	384MB의 RAM
	4GB ~ 16GB	512MB RAM

아키텍처	사용 가능한 메모리	예약된 최소 메모리
	16GB ~ 64GB	1GB RAM
	64GB ~ 128GB	2GB RAM
	128GB 이상	4GB RAM
IBM Z (**s390x**)	1GB에서 4GB	192MB RAM
	4GB ~ 64GB	256MB RAM
	64GB 이상	512MB RAM

많은 시스템에서 kdump 는 필요한 메모리 양을 추정하고 자동으로 예약할 수 있습니다. 이 동작은 기본적으로 활성화되어 있지만 시스템 아키텍처에 따라 다른 특정 양의 사용 가능한 메모리가 있는 시스템에서만 작동합니다.

중요

시스템의 총 메모리 양을 기반으로 예약된 메모리의 자동 구성은 최선의 노력 추정입니다. 실제 필요한 메모리는 I/O 장치와 같은 다른 요인에 따라 다를 수 있습니다. 커널 패닉 시 디버그 커널이 캡처 커널로 부팅되지 않을 수 있는 메모리가 충분하지 않을 수 있습니다. 이 문제를 방지하려면 크래시 커널 메모리를 충분히 늘립니다.

추가 리소스

- 기술 기능 및 제한 테이블

15.2. 자동 메모리 예약의 최소 임계값

kexec-tools 유틸리티는 기본적으로 crashkernel 명령줄 매개변수를 구성하고 kdump 에 대해 특정 양의 메모리를 예약합니다. 그러나 일부 시스템에서는 부트 로더 구성 파일에서 crashkernel=auto 매개 변수를 사용하거나 그래픽 구성 유틸리티에서 이 옵션을 활성화하여 kdump 에 메모리를 할당할 수 있습니다. 이 자동 예약이 작동하려면 시스템에서 특정 양의 총 메모리를 사용할 수 있어야 합니다. 메모리 요구 사항은 시스템의 아키텍처에 따라 다릅니다. 시스템 메모리가 지정된 임계값보다 작으면 메모리를 수동으로 구성해야 합니다.

표 15.2. 메모리 예약에 필요한 최소 메모리 양

아키텍처	필수 메모리
AMD64 and Intel 64 (**x86_64**)	1GB
IBM Power Systems(**ppc64le**)	2GB
IBM Z (**s390x**)	1GB
64비트 ARM	1GB

참고

부팅 명령줄의 crashkernel=auto 옵션은 RHEL 9 이상 릴리스에서 더 이상 지원되지 않습니다.

15.3. 지원되는 KDUMP 대상

커널 충돌이 발생하면 운영 체제는 구성된 또는 기본 대상 위치에 덤프 파일을 저장합니다. 덤프 파일을 장치에 직접 저장하거나 로컬 파일 시스템에 파일로 저장하거나 네트워크를 통해 덤프 파일을 보낼 수 있습니다. 다음 덤프 대상 목록을 사용하면 kdump 에서 현재 지원되거나 지원되지 않는 대상을 알 수 있습니다.

표 15.3. RHEL 9의 kdump 대상

대상 유형	지원되는 대상	지원되지 않는 대상
물리적 스토리지	• LVM(Logical Volume Manager). • 씬 프로비저닝 볼륨. • **qla2xxx**,**lpfc**,**bnx2fc**, **bfa** 와 같은 파이버 채널 (FC) 디스크. • 네트워크로 연결된 스토리지 서버의 **iSCSI** 소프트웨어 구성 논리 장치. • 소프트웨어 **RAID** 솔루션인 **mdraid** 하위 시스템. • **smartpqi**,**hp** said ,**megaraid**,**mpt3sas**,**aacraid** 및 **mpi3mr** 와 같은 하드웨어 RAID • **SCSI** 및 **SATA** 디스크. • **iSCSI** 및 **HBA** 오프로드. • **qla2xxx** 및 **lpfc** 와 같은 하드웨어 **FCoE**. • **bnx2fc** 와 같은 소프트웨어 **FCoE**. 소프트웨어 **FCoE** 가 작동하려면 추가 메모리 구성이 필요할 수 있습니다.	• BIOS RAID. • **iBFT** 를 사용하는 소프트웨어 **iSCSI**. 현재 지원되는 전송은 **bnx2i**,**cxgb3i** 및 **cxgb4i** 입니다. • **be2iscsi** 과 같은 하이브리드 장치 드라이버가 있는 소프트웨어 iSCSI. • **FCoE**(Fibre Channel over Ethernet). • 레거시 IDE. • **GlusterFS** 서버. • **Cryostat2** 파일 시스템. • 클러스터형 논리 볼륨 관리자(CLVM). • HA-LVM(고가용성 LVM 볼륨).

대상 유형	지원되는 대상	지원되지 않는 대상
네트워크	• **igb**,**ixgbe**,**i40e**,**e1000e**, **igc**,**tg3**, bnx2x ,**bnx2x**,**qede**,**cxgb4** 와 같은 커널 모듈을 사용하는 하드웨어 **be2net**,**enic, enic ,mlx4_en**,**mlx5_core**,**r 8169**,**atlantic**,**nfp** 및 **nicvf 64** 비트 **ARM** 아키텍처에서만.	• **sfc SRIOV**,**cxgb4vf** 및 **pch_gbe** 와 같은 커널 모듈을 사용하는 하드웨어. • IPv6 프로토콜. • clevis 연결입니다. • InfiniBand 네트워크 • 브리지 및 팀을 통한 VLAN 네트워크.
하이퍼바이저	• KVM(커널 기반 가상 시스템). • 특정 구성의 Cryostat 하이퍼바이저만 사용할 수 있습니다. • ESXi 6.6, 6.7, 7.0. • RHEL Cryostat1 UP 게스트의 Hyper-V 2012 R2 이상 버전만 사용할 수 있습니다.	
파일 시스템	**ext[234]fs**,**XFS**,**virtiofs** 및 NFS 파일 시스템.	**Btrfs** 파일 시스템입니다.
펌웨어	• BIOS 기반 시스템. • UEFI Secure Boot.	

추가 리소스

- kdump 대상 구성

15.4. 지원되는 KDUMP 필터링 수준

kdump 는 덤프 파일의 크기를 줄이기 위해makedumpfile 코어 수집기를 사용하여 데이터를 압축하고 원하지 않는 정보를 제외합니다. 예를 들어 -8 수준을 사용하여 hugepages 및 hugetlbfs 페이지를 제거할 수 있습니다. 현재 dumpfile 에서 지원하는 수준은*'kdump'에 대한 수준 필터링을 위해*표에서 확인할 수 있습니다.

표 15.4. kdump에 대한 수준 필터링

옵션	설명
1	페이지 0개
2	캐시 페이지
4	캐시 개인 정보
8	사용자 페이지
16	무료 페이지

추가 리소스

- 코어 수집기 구성

15.5. 지원되는 기본 오류 응답

기본적으로 **kdump** 가 코어 덤프를 생성하지 못하면 운영 체제가 재부팅됩니다. 그러나 코어 덤프를 기본 대상에 저장하지 못하는 경우 다른 작업을 수행하도록 **kdump** 를 구성할 수 있습니다.

dump_to_rootfs

코어 덤프를 루트 파일 시스템에 저장해 보십시오. 이 옵션은 네트워크 대상과 조합할 때 특히 유용합니다. 네트워크 대상에 연결할 수 없는 경우 이 옵션은 코어 덤프를 로컬에 저장하도록 kdump를 구성합니다. 이후 시스템이 재부팅됩니다.

reboot

시스템을 재부팅하여 프로세스의 코어 덤프를 손실합니다.

halt

시스템을 중지하고 프로세스의 코어 덤프를 손실합니다.

poweroff

시스템의 전원을 끄고 프로세스의 코어 덤프를 손실합니다.

shell

initramfs 내에서 쉘 세션을 실행하여 사용자가 코어 덤프를 수동으로 기록할 수 있습니다.

final_action

kdump 가 성공한 후 **reboot**, stop , **poweroff** 작업 또는 쉘 또는 **dump_to_rootfs** 오류 작업이 완료되면 추가 작업을 활성화합니다. 기본 **final_action** 옵션은 재부팅 입니다.

failure_action

커널 충돌 시 덤프가 실패할 때 수행할 작업을 지정합니다. 기본 **failure_action** 옵션은 재부팅 입니다.

추가 리소스

- kdump 기본 실패 응답 구성

15.6. FINAL_ACTION 매개변수 사용

kdump 가 성공하거나 kdump 가 구성된 대상에 vmcore 파일을 저장하지 못하는 경우, final_action 매개변수를 사용하여 reboot,halt, poweroff 와 같은 추가 작업을 수행할 수 있습니다. final_action 매개변수를 지정하지 않으면 재부팅이 기본 응답입니다.

절차

1. final_action 을 구성하려면 /etc/kdump.conf 파일을 편집하고 다음 옵션 중 하나를 추가합니다.

   - final_action reboot
   - final_action halt
   - final_action poweroff

2. 변경 사항을 적용하려면 kdump 서비스를 다시 시작하십시오.

```
# kdumpctl restart
```

15.7. FAILURE_ACTION 매개변수 사용

failure_action 매개변수는 커널 충돌 시 덤프가 실패할 때 수행할 작업을 지정합니다. failure_action 의 기본 작업은 시스템을 재부팅 하는 ;을(를) 재부팅합니다.

매개변수는 수행할 다음 작업을 인식합니다.

reboot

덤프 실패 후 시스템을 재부팅합니다.

dump_to_rootfs

루트가 아닌 덤프 대상이 구성된 경우 덤프 파일을 루트 파일 시스템에 저장합니다.

halt

시스템을 중지합니다.

poweroff

시스템에서 실행 중인 작업을 중지합니다.

shell

initramfs 내에서 쉘 세션을 시작하여 추가 복구 작업을 수동으로 수행할 수 있습니다.

절차:

1. 덤프가 실패하는 경우 수행할 작업을 구성하려면 **/etc/kdump.conf** 파일을 편집하고 **failure_action** 옵션 중 하나를 지정합니다.

   - **failure_action reboot**
   - **failure_action halt**
   - **failure_action poweroff**
   - **failure_action 쉘**
   - **failure_action dump_to_rootfs**

2. 변경 사항을 적용하려면 **kdump** 서비스를 다시 시작하십시오.

```
# kdumpctl restart
```

16장. 펌웨어에서 덤프 메커니즘 지원

펌웨어 지원 덤프(fadump)는 IBM POWER 시스템의 kdump 메커니즘 대신 제공되는 덤프 캡처 메커니즘입니다. kexec 및 kdump 메커니즘은 AMD64 및 Intel 64 시스템에서 코어 덤프를 캡처하는 데 유용합니다. 그러나 미니 시스템 및 메인프레임 컴퓨터와 같은 일부 하드웨어는 온보드 펌웨어를 활용하여 메모리 영역을 분리하고 충돌 분석에 중요한 데이터의 실수로 덮어 쓰기를 방지합니다. fadump 유틸리티는 fadump 메커니즘과 IBM POWER 시스템의 RHEL과의 통합에 최적화되어 있습니다.

16.1. IBM POWERPC 하드웨어에서 지원되는 펌웨어

fadump 유틸리티는 PCI 및 I/O 장치가 있는 완전 재설정 시스템에서 vmcore 파일을 캡처합니다. 이 메커니즘은 펌웨어를 사용하여 충돌 중에 메모리 영역을 보존한 다음 kdump 사용자 공간 스크립트를 사용하여 vmcore 파일을 저장합니다. 메모리 영역은 부팅 메모리, 시스템 레지스터 및 하드웨어 페이지 테이블 항목(PTE)을 제외한 모든 시스템 메모리 콘텐츠로 구성됩니다.

fadump 메커니즘은 파티션을 재부팅하고 새 커널을 사용하여 이전 커널 충돌의 데이터를 덤프하여 기존 덤프 유형보다 향상된 안정성을 제공합니다. fadump 에는 IBM POWER6 프로세서 기반 버전 하드웨어 플랫폼이 필요합니다.

하드웨어 재설정의 PowerPC 특정 방법을 포함하여 fadump 메커니즘에 대한 자세한 내용은 /usr/share/doc/kexec-tools/fadump-howto.txt 파일을 참조하십시오.

참고

부팅 메모리라는 보존되지 않은 메모리 영역은 충돌 이벤트 후 커널을 성공적으로 부팅하는 데 필요한 RAM의 양입니다. 기본적으로 부팅 메모리 크기는 총 시스템 RAM의 256MB 또는 5 %이며 이는 더 큽니다.

kexec-initiated 이벤트와 달리 fadump 메커니즘은 프로덕션 커널을 사용하여 크래시 덤프를 복구합니다. 충돌 후 부팅 시 PowerPC 하드웨어는 장치 노드 /proc/device-tree/rtas/ibm.kernel-dump 를 proc 파일 시스템(procfs)에서 사용할 수 있도록 합니다. fadump 인식 kdump 스크립트는 저장된 vmcore 를 확인한 다음 시스템 재부팅을 정상적으로 완료합니다.

16.2. 펌웨어 지원 덤프 메커니즘 활성화

펌웨어 지원 덤프(fadump) 메커니즘을 활성화하여 IBM POWER 시스템의 크래시 덤프 기능을 개선할 수 있습니다.

Secure Boot 환경에서 GRUB 부트 로더는 RMA(Real Mode Area)라고 하는 부팅 메모리 영역을 할당합니다. RMA의 크기는 부팅 구성 요소로 나누어지며, 구성 요소가 크기 할당을 초과하면 OOM(메모리 부족) 오류와 함께 GRUB이 실패합니다.

주의

RHEL 9.1 및 이전 버전의 Secure Boot 환경에서 펌웨어 지원 덤프(fadump) 메커니즘을 활성화하지 마십시오. GRUB 부트 로더는 다음과 같은 오류와 함께 실패합니다.

```
error: ../../grub-core/kern/mm.c:376:out of memory.
Press any key to continue…
```

fadump 구성으로 인해 기본 initramfs 크기를 늘리는 경우에만 시스템을 복구할 수 있습니다.

시스템을 복구하는 해결 방법에 대한 자세한 내용은 GRUB OOM(메모리 부족) 문서에서 시스템 부팅 을 참조하십시오.

사전 요구 사항

- 시스템에 대한 root 권한이 있습니다.

절차

1. kexec-tools 패키지를 설치합니다.

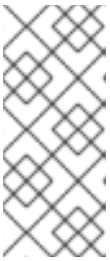
2. crashkernel 의 기본값을 구성합니다.

```
# kdumpctl reset-crashkernel --fadump=on --kernel=ALL
```

3. (선택 사항) 기본값 대신 부팅 메모리를 예약합니다.

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

XXM 은 필요한 메모리 크기(**MB**)입니다.



참고

부팅 구성 옵션을 지정할 때 **kdump** 가 활성화된 커널을 재부팅하여 구성을 테스트합니다. **kdump** 커널을 부팅하지 못하면 **crashkernel** 값을 점진적으로 늘려 적절한 값을 설정합니다.

4. 변경 사항을 적용하려면 재부팅합니다.

```
# reboot
```

16.3. IBM Z 하드웨어에서 펌웨어 지원 덤프 메커니즘

IBM Z 시스템은 다음과 같은 펌웨어 지원 덤프 메커니즘을 지원합니다.

- 독립 실행형 덤프(**sadump**)
- **VMDUMP**

kdump 인프라는 **IBM Z** 시스템에서 지원 및 활용됩니다. 그러나 **IBM Z**에 대한 펌웨어 지원 덤프(**fadump**) 방법 중 하나를 사용하면 다음과 같은 다양한 이점을 제공할 수 있습니다.

- **sadump** 메커니즘은 시스템 콘솔에서 시작 및 제어되며 **IPL** 부팅 가능 장치에 저장됩니다.
- **VMDUMP** 메커니즘은 **sadump** 와 유사합니다. 이 툴은 시스템 콘솔에서도 시작되지만 하드웨어에서 결과 덤프를 검색하여 분석을 위해 시스템에 복사합니다.
- 이러한 방법(다른 하드웨어 기반 덤프 메커니즘과 동일)은 **kdump** 서비스가 시작되기 전에 초기 부팅 단계에서 시스템의 상태를 캡처할 수 있습니다.
- **VMDUMP** 에는 덤프 파일을 **Red Hat Enterprise Linux** 시스템으로 수신하는 메커니즘이 포함되어 있지만 **VMDUMP** 의 구성 및 제어는 **IBM Z Hardware** 콘솔에서 관리됩니다.

추가 리소스

- Red Hat Enterprise Linux 8.5에서 덤프 툴 사용
- 독립형 덤프
- VMDUMP를 사용하여 z/VM에서 덤프 생성

16.4. FUJITSU PRIMEQUEST 시스템에서 SADUMP 사용

Fujitsu sadump 메커니즘은 **kdump** 가 성공적으로 완료할 수 없는 경우 이벤트에 대체 덤프 캡처를 제공하도록 설계되었습니다. **sadump** 메커니즘은 시스템 관리 보드(MMB) 인터페이스에서 수동으로 호출됩니다. **MMB**를 사용하여 **Intel 64** 또는 **AMD 64** 서버의 경우와 같이 **kdump** 를 구성한 다음 **sadump** 를 활성화합니다.

절차

1. **/etc/sysctl.conf** 파일에 다음 행을 추가하거나 편집하여 **kdump** 가 **sadump** 에 대해 예상대로 시작되는지 확인합니다.

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

주의

특히 **kdump** 후 시스템이 재부팅되지 않는지 확인하십시오. **kdump** 가 **vmcore** 파일을 저장하지 못한 후 시스템이 재부팅되면 **sadump** 를 호출할 수 없습니다.

2. 중지 또는 쉘 과 같이 **/etc/kdump.conf** 에서 **failure_action** 매개변수를 적절하게 설정합니다.

```
failure_action shell
```

추가 리소스

- **FUJITSU Server PRIMEQUEST 2000 설치 설명서**

17장. 코어 덤프 분석

시스템 충돌 원인을 확인하기 위해 크래시 유틸리티를 사용하면 **GNU Debugger(GDB)**와 매우 유사한 대화형 프롬프트를 제공할 수 있습니다. 이 유틸리티를 사용하면 **kdump,netdump,diskdump** 또는 **xendump** 및 실행 중인 **Linux** 시스템에서 생성한 코어 덤프를 대화형으로 분석할 수 있습니다. 또는 **Kernel Oops Analyzer** 또는 **Kdump Helper** 툴을 사용할 수 있는 옵션이 있습니다.

17.1. 크래시 유틸리티 설치

제공된 정보를 사용하여 필요한 패키지 및 크래시 유틸리티를 설치하는 절차를 이해하십시오. 크래시 유틸리티는 **Red Hat Enterprise Linux 9** 시스템에 기본적으로 설치되지 않을 수 있습니다. 크래시 기능은 실행 중이거나 커널 충돌이 발생하고 코어 덤프 파일이 생성되는 동안 시스템의 상태를 대화형으로 분석하는 툴입니다. 코어 덤프 파일은 **vmcore** 파일이라고도 합니다.

절차

1. 관련 리포지토리를 활성화합니다.

```
# subscription-manager repos --enable baseos repository
# subscription-manager repos --enable appstream repository
# subscription-manager repos --enable rhel-9-for-x86_64-baseos-debug-rpms
```

2. 크 래시 패키지를 설치합니다.

```
# dnf install crash
```

3. **kernel-debuginfo** 패키지를 설치합니다.

```
# dnf install kernel-debuginfo
```

kernel-debuginfo 패키지는 실행 중인 커널에 대응하고 덤프 분석에 필요한 데이터를 제공합니다.

17.2. 크래시 유틸리티 실행 및 종료

제공된 정보를 사용하여 필요한 매개 변수와 크래시 유틸리티를 실행하고 종료하는 절차를 파악합니다. 크래시 는 커널 크래시가 발생하거나 코어 덤프 파일이 생성되는 동안 시스템의 상태를 대화형으로 분

석하는 툴입니다. 코어 덤프 파일은 **vmcore** 파일이라고도 합니다.

사전 요구 사항

- 현재 실행 중인 커널(예: **5.14.0-1.el9.x86_64**)을 확인합니다.

절차

1. 크래시 유틸리티를 시작하려면 다음 두 가지 필수 매개 변수를 명령에 전달해야 합니다.

   - **debug-info**(연결 해제된 **vmlinuz** 이미지)(예: **/usr/lib/debug/lib/modules/5.14.0-1.el9.x86_64/vmlinux**)는 특정 **kernel-debuginfo** 패키지를 통해 제공됩니다.

   - 실제 **vmcore** 파일(예: **/var/crash/127.0.0.1-2021-09-13-14:05:33/vmcore**)

     결과 크래시 명령은 다음과 같습니다.

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

     kdump 가 캡처한 것과 동일한 ***<kernel>*** 버전을 사용합니다.

예 17.1. 크래시 유틸리티 실행

다음 예제에서는 **5.14.0-1.el9.x86_64** 커널을 사용하여 **2021**년 **9**월 **13**일 오후 **14:05** 에 생성된 코어 덤프를 분석하는 방법을 보여줍니다.

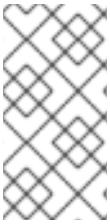
```
...
WARNING: kernel relocated [202MB]: patching 90160 gdb minimal_symbol values

      KERNEL: /usr/lib/debug/lib/modules/5.14.0-1.el9.x86_64/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2021-09-13-14:05:33/vmcore  [PARTIAL DUMP]
        CPUS: 2
        DATE: Mon Sep 13 14:05:16 2021
      UPTIME: 01:03:57
LOAD AVERAGE: 0.00, 0.00, 0.00
       TASKS: 586
    NODENAME: localhost.localdomain
     RELEASE: 5.14.0-1.el9.x86_64
     VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
     MACHINE: x86_64  (2904 Mhz)
```

```
      MEMORY: 2.9 GB
       PANIC: "sysrq: SysRq : Trigger a crash"
         PID: 10635
     COMMAND: "bash"
        TASK: ffff8d6c84271800  [THREAD_INFO: ffff8d6c84271800]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)

crash>
```

2. 대화형 프롬프트를 종료하고 크래시 를 중지하려면 **exit** 또는 **q** 를 입력합니다.

예 17.2. 크래시 유틸리티 종료

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



참고

크래시 명령은 라이브 시스템을 디버깅하기 위한 강력한 도구로도 사용할 수 있습니다. 그러나 시스템을 중단하지 않도록 주의해서 사용하십시오.

추가 리소스

- 예기치 않은 시스템 재시작 가이드

17.3. 크래시 유틸리티에 다양한 지표 표시

크래시 유틸리티를 사용하여 커널 메시지 버퍼, 역추적, 프로세스 상태, 가상 메모리 정보 및 파일 열기와 같은 다양한 지표를 표시합니다.

메시지 버퍼 표시

- 커널 메시지 버퍼를 표시하려면 대화형 프롬프트에서 **log** 명령을 입력합니다.

```
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> ffffffb 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
```

명령 사용에 대한 자세한 내용은 도움말 로그 를 입력합니다.

참고

커널 메시지 버퍼는 시스템 충돌에 대한 가장 중요한 정보를 포함하며, 따라서 항상 먼저 vmcore-dmesg.txt 파일에 먼저 덤프됩니다. 이는 예를 들어 대상 위치에 공간이 없기 때문에 전체 vmcore 파일을 가져오려고 할 때 유용합니다. 기본적으로 vmcore-dmesg.txt 는 /var/crash/ 디렉터리에 있습니다.

역추적 표시

- **커널 스택 추적을 표시하려면 bt 명령을 사용합니다.**

```
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
```

bt < *pid* >를 입력하여 특정 프로세스의 역추적 또는 bt 사용량에 대한 자세한 정보를 보려면 help bt 를 입력합니다.

프로세스 상태 표시

- **시스템에서 프로세스 상태를 표시하려면 ps 명령을 사용합니다.**

```
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
```

ps *<pid>* 를 사용하여 단일 특정 프로세스의 상태를 표시합니다. ps 사용에 대한 자세한 내용은 *help ps* 를 사용하십시오.

가상 메모리 정보 표시

- **기본 가상 메모리 정보를 표시하려면 대화형 프롬프트에서 vm 명령을 입력합니다.**

```
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 ...
```

vm & *lt;pid >*를 사용하여 단일 특정 프로세스에 대한 정보를 표시하거나 vm 사용량에 대한 자세한 내용은 help vm 를 사용합니다.

열려 있는 파일 표시

- **열려 있는 파일에 대한 정보를 표시하려면 files 명령을 사용합니다.**

```
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
```

파일 & *lt;pid >*를 사용하여 선택한 프로세스로 열린 파일을 표시하거나 파일 사용에 대한 자세한 내용은 도움말 파일을 사용합니다.

17.4. 커널 OOPS ANALYZER 사용

Kernel Oops Analyzer 툴은 oops 메시지를 기술 자료의 알려진 문제와 비교하여 크래시 덤프를 분석합니다.

사전 요구 사항

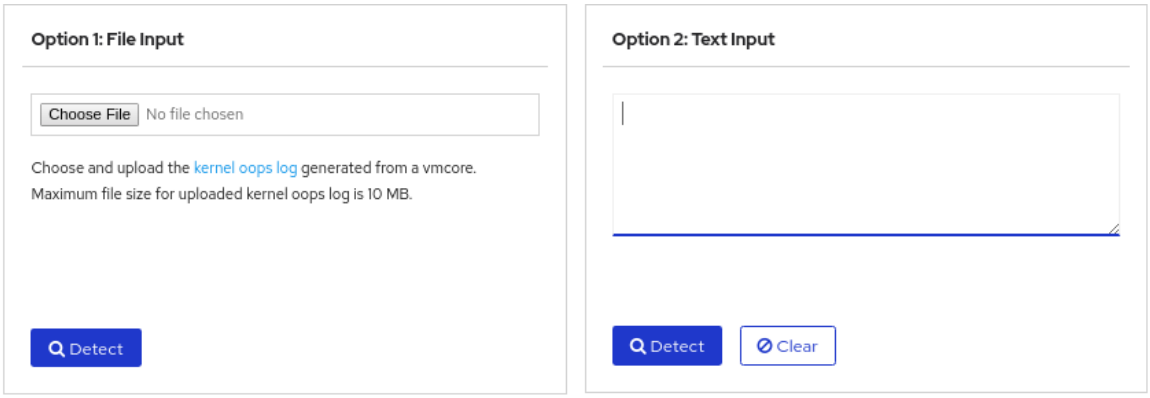
- **oops 메시지를 보호하여 Kernel Oops Analyzer를 피드합니다.**

절차

1. 커널 **Oops Analyzer** 툴에 액세스합니다.

2. 커널 충돌 문제를 진단하려면 **vmcore** 에서 생성된 커널 **oops** 로그를 업로드합니다.

   - 또는 텍스트 메시지 또는 **vmcore-dmesg.txt** 를 입력으로 제공하여 커널 충돌 문제를 진단할 수도 있습니다.



3. **DETECT** 를 클릭하여 **makedumpfile** 의 정보를 기반으로 하는 **oops** 메시지를 알려진 솔루션과 비교합니다.

추가 리소스

- 커널 Oops 분석기 문서
- 예기치 않은 시스템 재시작 가이드

17.5. KDUMP HELPER 툴

Kdump Helper 툴은 제공된 정보를 사용하여 **kdump** 를 설정하는 데 도움이 됩니다. **kdump Helper**는 기본 설정에 따라 설정 스크립트를 생성합니다. 서버에서 스크립트를 시작하고 실행하는 경우 **kdump** 서비스가 설정됩니다.

추가 리소스

- **kdump Helper**

18장. 초기 KDUMP를 사용하여 부팅 시간 충돌 캡처

early kdump는 시스템 서비스가 시작되기 전에 부팅 프로세스의 초기 단계에서 시스템 또는 커널 충돌이 발생하는 경우 vmcore 파일을 캡처하는 kdump 메커니즘의 기능입니다. early kdump는 크래시 커널과 크래시 커널의 initramfs 를 훨씬 일찍 로드합니다.

18.1. 초기 KDUMP란 무엇입니까?

커널 충돌은 kdump 서비스가 시작되기 전에 초기 부팅 단계에서 발생할 수 있으며 충돌하는 커널 메모리의 내용을 캡처하고 저장할 수 있습니다. 따라서 문제 해결에 중요한 정보가 손실되는 충돌과 관련된 중요한 정보가 손실됩니다. 이 문제를 해결하려면 kdump 서비스의 일부인 초기 kdump 기능을 사용할 수 있습니다.

18.2. 초기 KDUMP 활성화

early kdump 기능은 크래시 커널과 초기 RAM 디스크 이미지(initramfs)를 설정하여 조기 충돌을 위해 vmcore 정보를 캡처할 수 있을 만큼 조기에 로드되도록 설정합니다. 이를 통해 초기 부팅 커널 충돌에 대한 정보가 손실될 위험이 제거됩니다.

사전 요구 사항

- 활성 RHEL 서브스크립션입니다.
- 시스템 CPU 아키텍처용 kexec-tools 패키지가 포함된 리포지토리입니다.
- kdump 구성 및 대상 요구 사항을 충족했습니다. 자세한 내용은 지원되는 kdump 구성 및 대상 을 참조하십시오.

절차

1. kdump 서비스가 활성화되어 활성화되어 있는지 확인합니다.

```
# systemctl is-enabled kdump.service && systemctl is-active kdump.service
enabled
active
```

kdump 가 활성화되어 실행되지 않은 경우 필요한 구성을 모두 설정하고 kdump 서비스가 활성화되어 있는지 확인합니다.

2. 초기 **kdump** 기능을 사용하여 부팅 커널의 **initramfs** 이미지를 다시 빌드합니다.

```
# dracut -f --add earlykdump
```

3. **rd.earlykdump** 커널 명령줄 매개변수를 추가합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="rd.earlykdump"
```

4. 변경 사항을 반영하도록 시스템 재부팅

```
# reboot
```

검증 단계

- **rd.earlykdump** 가 성공적으로 추가되었고 **early kdump** 기능이 활성화되었는지 확인합니다.

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.14.0-1.el9.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
Sep 13 15:46:11 redhat dracut-cmdline[304]: early-kdump is enabled.
Sep 13 15:46:12 redhat dracut-cmdline[304]: kexec: loaded early-kdump kernel
```

추가 리소스

- **/usr/share/doc/kexec-tools/early-kdump-howto.txt** 파일
- 초기 **kdump** 지원이란 무엇이며 어떻게 구성합니까?
- **kdump** 활성화

19장. 커널 및 SECURE BOOT에 대한 모듈 서명

서명된 커널 및 서명된 커널 모듈을 사용하여 시스템의 보안을 강화할 수 있습니다. Secure Boot가 활성화된 UEFI 기반 빌드 시스템에서는 개인 빌드 커널 또는 커널 모듈을 자체 서명할 수 있습니다. 또한 커널 또는 커널 모듈을 배포하려는 대상 시스템으로 공개 키를 가져올 수 있습니다.

Secure Boot가 활성화된 경우 다음 구성 요소를 모두 개인 키로 서명하고 해당 공개 키로 인증해야 합니다.

- UEFI 운영 체제 부트 로더
- Red Hat Enterprise Linux 커널
- 모든 커널 모듈

이러한 구성 요소 중 하나라도 서명 및 인증되지 않은 경우 시스템에서 부팅 프로세스를 완료할 수 없습니다.

Red Hat Enterprise Linux 9에는 다음이 포함됩니다.

- 서명된 부트 로더
- 서명된 커널
- 서명된 커널 모듈

또한 서명된 first-stage 부트 로더와 서명된 커널에는 Red Hat 공개 키가 포함되어 있습니다. 이러한 서명된 실행 바이너리 및 포함된 키를 사용하면 Red Hat Enterprise Linux 9가 UEFI Secure Boot 부팅을 지원하는 시스템의 UEFI 펌웨어에서 제공하는 Microsoft UEFI Secure Boot 인증 기관 키를 설치, 부팅 및 실행할 수 있습니다.

참고

- 일부 **UEFI** 기반 시스템에서 **Secure Boot**에 대한 지원이 포함된 것은 아닙니다.
- 커널 모듈을 빌드하고 서명하는 빌드 시스템은 **UEFI Secure Boot**를 사용할 필요가 없으며 **UEFI** 기반 시스템도 필요하지 않습니다.

19.1. 사전 요구 사항

- 외부에서 빌드된 커널 모듈에 서명하려면 다음 패키지에서 유틸리티를 설치합니다.

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

표 19.1. 필수 유틸리티

utility	패키지에서 제공	사용 중	목적
efikeygen	**pesign**	빌드 시스템	공개 및 개인 X.509 키 쌍 생성
openssl	**openssl**	빌드 시스템	암호화되지 않은 개인 키 내보내기
sign-file	**kernel-devel**	빌드 시스템	개인 키를 사용하여 커널 모듈에 서명하는 데 사용되는 실행 파일
mokutil	**mokutil**	대상 시스템	공개 키를 수동으로 등록하는 데 사용되는 선택적 유틸리티
keyctl	**keyutils**	대상 시스템	시스템 인증 키에 공개 키를 표시하는 데 사용되는 선택적 유틸리티

19.2. UEFI SECURE BOOT

UEFI(***Unified Extensible Firmware Interface*** **) Secure Boot** 기술을 사용하면 신뢰할 수 있는 키로 서명되지 않은 커널 공간 코드의 실행을 방지할 수 있습니다. 시스템 부트 로더는 암호화 키를 사용하여 서명됩니다. 펌웨어에 포함된 공개 키의 데이터베이스는 서명 키를 인증합니다. 나중에 다음 단계 부트 로더 및 커널에서 서명을 확인할 수 있습니다.

UEFI Secure Boot는 다음과 같이 펌웨어에서 서명된 드라이버 및 커널 모듈로 신뢰 체인을 설정합니다.

- **UEFI** 개인 키 서명과 공개 키는 **shim** 첫 번째 단계 부트 로더를 인증합니다. *인증 기관* (**CA**) 은 공개 키에 서명합니다. **CA**는 펌웨어 데이터베이스에 저장됩니다.

- **shim** 파일에는 **GRUB** 부트 로더와 커널을 인증하는 **Red Hat** 공개 키 **Red Hat Secure Boot (CA** 키 **1)** 가 포함되어 있습니다.

- 커널에 차례로 드라이버와 모듈을 인증하는 공개 키가 포함되어 있습니다.

Secure Boot는 **UEFI** 사양의 부팅 경로 검증 구성 요소입니다. 사양은 다음을 정의합니다.

- 비휘발성 스토리지에서 암호로 보호되는 **UEFI** 변수를 위한 프로그래밍 인터페이스입니다.

- **UEFI** 변수에 신뢰할 수 있는 **X.509** 루트 인증서를 저장합니다.

- 부트 로더 및 드라이버와 같은 **UEFI** 애플리케이션 검증.

- 알려진**-bad** 인증서 및 애플리케이션 해시를 취소하는 절차입니다.

UEFI Secure Boot는 무단 변경 사항을 감지하는 데 도움이 되지만 다음과 같은 것은 아닙니다.

- 두 번째 단계 부트 로더의 설치 또는 제거를 방지합니다.

- 이러한 변경에 대한 명확한 사용자 확인이 필요합니다.

- 부팅 경로 조작을 중지합니다. 부트 로더가 설치 또는 업데이트될 때 아닌 부팅 중에 서명을 확인합니다.

부트 로더 또는 커널이 시스템 신뢰할 수 있는 키로 서명되지 않은 경우 **Secure Boot**가 시작되지 않습니다.

19.3. UEFI SECURE BOOT 지원

커널과 로드된 모든 드라이버가 신뢰할 수 있는 키로 서명된 경우 **UEFI Secure Boot**가 활성화된 시스템에서 **Red Hat Enterprise Linux 9**를 설치하고 실행할 수 있습니다. **Red Hat**은 관련 **Red Hat** 키로 서명 및 인증된 커널 및 드라이버를 제공합니다.

외부에서 빌드된 커널 또는 드라이버를 로드하려면 이를 서명해야 합니다.

UEFI Secure Boot에 의한 제한 사항

- 시스템이 올바르게 인증된 후 커널 모드 코드만 실행합니다.
- **GRUB** 모듈 로드는 **GRUB** 모듈에 서명 및 검증을 위한 인프라가 없기 때문에 비활성화됩니다. 이를 로드할 수 있도록 허용하면 **Secure Boot**에서 정의하는 보안 **gateway** 내부의 신뢰할 수 없는 코드를 실행하도록 구성됩니다.
- **Red Hat**은 **Red Hat Enterprise Linux 9**에서 지원되는 모든 모듈이 포함된 서명된 **GRUB** 바이너리를 제공합니다.

추가 리소스

- UEFI Secure Boot에서 설정한 제한 사항

19.4. X.509 키를 사용하여 커널 모듈을 인증하기 위한 요구사항

Red Hat Enterprise Linux 9에서 커널 모듈이 로드되면 커널은 커널 시스템 키링 (**.builtin_trusted_keys**) 및 커널 플랫폼 키링(**.platform**)에서 공용 **X.509** 키에 대해 모듈의 서명을 확인합니다. **.platform** 인증 키에는 타사 플랫폼 공급자의 키와 사용자 지정 공개 키가 포함되어 있습니다. 커널 시스템 **.blacklist** 인증 키의 키는 검증에서 제외됩니다.

UEFI Secure Boot 기능이 활성화된 시스템에서 커널 모듈을 로드하려면 특정 조건을 충족해야 합니다.

- **UEFI Secure Boot**가 활성화되어 있거나 **module.sig_enforce** 커널 매개변수가 지정된 경우:
  -

서명 된 커널 모듈만 시스템 인증 키 (. builtin_trusted_keys) 및 플랫폼 인증 키 (.platform.platform)의 키에 대해 인증 된 커널 모듈 만 로드할 수 있습니다.

- 공개 키는 시스템에 해지된 키 키(.blacklist)에 없어야 합니다.

- UEFI Secure Boot가 비활성화되고 module.sig_enforce 커널 매개 변수가 지정되지 않은 경우:
  - 공개 키 없이 서명되지 않은 커널 모듈 및 서명된 커널 모듈을 로드할 수 있습니다.

- 시스템이 UEFI 기반이 아니거나 UEFI Secure Boot가 비활성화된 경우:
  - 커널에 포함된 키만 .builtin_trusted_keys 및 .platform 에 로드됩니다.
  - 커널을 다시 빌드하지 않고도 해당 키 세트를 보강할 수 없습니다.

표 19.2. 로드를 위한 커널 모듈 인증 요구 사항

모듈 서명	공개 키 및 서명 유효한 공개 키	UEFI Secure Boot 상태	sig_enforce	모듈 로드	커널 테인트
서명되지 않음	-	활성화되지 않음	활성화되지 않음	succeeds	있음
		활성화되지 않음	enabled	실패	-
		enabled	-	실패	-
signed	없음	활성화되지 않음	활성화되지 않음	succeeds	있음
		활성화되지 않음	enabled	실패	-
		enabled	-	실패	-
signed	있음	활성화되지 않음	활성화되지 않음	succeeds	없음
		활성화되지 않음	enabled	succeeds	없음
		enabled	-	succeeds	없음

19.5. 공개 키 소스

부팅 중에 커널은 영구 키 저장소 집합에서 **X.509** 키를 다음 인증 키로 로드합니다.

- 시스템 인증 키 (. **builtin_trusted_keys**)
- **.platform** 인증
- 시스템 **.blacklist** 인증 키

표 **19.3.** 시스템 인증 키의 소스

X.509 키 소스	사용자가 키를 추가할 수 있습니다.	UEFI Secure Boot 상태	부팅 중 로드된 키
커널에 삽입	없음	-	**.builtin_trusted_keys**
UEFI **db**	제한됨	활성화되지 않음	없음
		enabled	**.platform**
shim 부트 로더에 포함	없음	활성화되지 않음	없음
		enabled	**.platform**
머신 소유자 키(MOK) 목록	있음	활성화되지 않음	없음
		enabled	**.platform**

.builtin_trusted_keys

- 부팅 시 빌드된 인증 키
- 신뢰할 수 있는 공개 키 포함
- 키를 보려면 루트 권한이 필요합니다.

.platform

- 부팅 시 빌드된 인증 키
- 타사 플랫폼 공급자의 키 및 사용자 지정 공개 키 포함
- 키를 보려면 루트 권한이 필요합니다.

.blacklist

- 취소된 **X.509** 키가 있는 인증 키
- 공개 키가 **.builtin_trusted_keys**에 있는 경우에도 **.blacklist** 의 키로 서명된 모듈이 인증이 실패합니다.

UEFI Secure Boot db

- 서명 데이터베이스
- **UEFI** 애플리케이션, **UEFI** 드라이버 및 부트 로더의 키(해시) 저장
- 키를 머신에 로드할 수 있습니다.

UEFI Secure Boot dbx

- 취소된 서명 데이터베이스
- 키가 로드되지 않도록 합니다.
- 이 데이터베이스에서 해지된 키는 **.blacklist** 키에 추가됩니다.

19.6. 공개 및 개인 키 쌍 생성

Secure Boot 지원 시스템에서 사용자 지정 커널 또는 사용자 지정 커널 모듈을 사용하려면 공개 및 개

인 **X.509** 키 쌍을 생성해야 합니다. 생성된 개인 키를 사용하여 커널 또는 커널 모듈에 서명할 수 있습니다. **Secure Boot**의 **MOK(Machine Owner Key)**에 해당 공개 키를 추가하여 서명된 커널 또는 커널 모듈을 검증할 수도 있습니다.

주의

강력한 보안 조치 및 액세스 정책을 적용하여 개인 키의 콘텐츠를 보호합니다. 잘못된 핸드에서는 키를 사용하여 해당 공개 키로 인증된 시스템을 손상시킬 수 있습니다.

절차

- **X.509** 공개 및 개인 키 쌍을 생성합니다.
  - 사용자 정의 커널 모듈 만 서명하려는 경우 :

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

  - 사용자 정의 *커널에* 서명하려면 다음을 수행합니다.

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

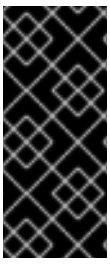
  - **RHEL** 시스템이 **FIPS** 모드를 실행하는 경우:

```
# efikeygen --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'
```

참고

FIPS 모드에서는 **efikeygen** 이 **PKI** 데이터베이스에서 기본 **"NSS Certificate DB"** 토큰을 찾도록 **--token** 옵션을 사용해야 합니다.

공개 키와 개인 키는 이제 **/etc/pki/pesign/** 디렉토리에 저장됩니다.



중요

서명 키의 유효 기간 내에 커널 및 커널 모듈에 서명하는 것이 좋습니다. 그러나 서명 파일 유틸리티는 경고하지 않으며 유효 날짜에 관계없이 **Red Hat Enterprise Linux 9**에서 키를 사용할 수 있습니다.

추가 리소스

- **OpenSSL(1)** 매뉴얼 페이지
- *RHEL 보안 가이드*
- **MOK** 목록에 공개 키를 추가하여 대상 시스템에서 공개 키 등록

19.7. 시스템 인증 키의 출력 예

key utils 패키지의 **keyctl** 유틸리티를 사용하여 시스템 인증 키 의 키에 대한 정보를 표시할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.
- **keyutils** 패키지에서 **keyctl** 유틸리티를 설치했습니다.

> **예 19.1.** 인증 키 출력
>
> 다음은 **UEFI Secure Boot**가 활성화된 **Red Hat Enterprise Linux 9** 시스템에서

.builtin_trusted_keys,.platform, .blacklist 키링의 단축된 예제 출력입니다.

```
# 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: cc6fafe72...
...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...
```

예제의 .builtin_trusted_keys 키는 UEFI Secure Boot db 키와 shim 부트 로더에 포함된 Red Hat Secure Boot(CA 키 1) 의 두 개의 키를 추가하는 방법을 보여줍니다.

예 19.2. 커널 콘솔 출력

다음 예제에서는 커널 콘솔 출력을 보여줍니다. 메시지는 UEFI Secure Boot 관련 소스를 사용하여 키를 식별합니다. 여기에는 UEFI Secure Boot db, embedded shim, MOK 목록이 포함됩니다.

```
# 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...
```

추가 리소스

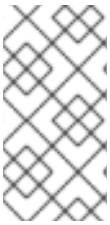
- keyctl(1), dmesg(1) 매뉴얼 페이지

19.8. MOK 목록에 공개 키를 추가하여 대상 시스템에 공개 키 등록

커널 또는 커널 모듈을 인증하고 로드하려는 모든 시스템에 공개 키를 등록해야 합니다. 플랫폼 키링(.platform)이 공개 키를 사용하여 커널 또는 커널 모듈을 인증할 수 있도록 대상 시스템의 공개 키를 다양한 방법으로 가져올 수 있습니다.

RHEL 9가 Secure Boot가 활성화된 UEFI 기반 시스템에서 부팅되면 커널은 Secure Boot db 키 데이터베이스에 있는 플랫폼 키링(.platform)에 로드됩니다. 동시에 커널은 취소된 키의 dbx 데이터베이스에서 키를 제외합니다.

MOK(Machine Owner Key) 기능 기능을 사용하여 UEFI Secure Boot 키 데이터베이스를 확장할 수 있습니다. Secure Boot가 활성화된 UEFI 지원 시스템에서 RHEL 9가 부팅되면 키 데이터베이스의 키 외에 MOK 목록의 키도 플랫폼 인증 키링(.platform)에 추가됩니다. MOK 목록 키도 지속적으로 저장되고 안전하게 Secure Boot 데이터베이스 키와 동일한 방식으로 저장되지만 이는 두 가지 개별 기능입니다. MOK 기능은 shim,MokManager,GRUB 및 mokutil 유틸리티에서 지원됩니다.



참고

시스템에서 커널 모듈을 쉽게 인증하려면 시스템 공급 업체에 공개 키를 팩토리 펌웨어 이미지의 UEFI Secure Boot 키 데이터베이스에 통합하는 것이 좋습니다.

사전 요구 사항

- 공개 키 및 개인 키 쌍을 생성하고 공개 키의 유효 날짜를 알고 있습니다. 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.

절차

1. 공개 키를 sb_cert.cer 파일로 내보냅니다.

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

2. 공개 키를 MOK 목록으로 가져옵니다.

-

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

3. 이 MOK 등록 요청에 대한 새 암호를 입력합니다.

4. 시스템을 재부팅합니다.

   shim 부트 로더는 보류 중인 MOK 키 등록 요청을 통지하고 MokManager.efi 를 시작하여 UEFI 콘솔의 등록을 완료할 수 있습니다.

5. Enroll MOK 를 선택하고, 메시지가 표시되면 이 요청과 관련된 암호를 입력하고 등록을 확인합니다.

   공개 키는 지속적인 MOK 목록에 추가됩니다.

   키가 MOK 목록에 있으면 이 키로 자동으로 .platform 키링으로 전파되고 UEFI Secure Boot 가 활성화되면 후속 부팅이 수행됩니다.

19.9. 개인 키로 커널 서명

UEFI Secure Boot 메커니즘이 활성화된 경우 서명된 커널을 로드하여 시스템에서 향상된 보안 이점을 얻을 수 있습니다.

사전 요구 사항

- 공개 키 및 개인 키 쌍을 생성하고 공개 키의 유효 날짜를 알고 있습니다. 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.
- 대상 시스템에 공개 키를 등록했습니다. 자세한 내용은 공개 키를 MOK 목록에 추가하여 대상 시스템에서 공개 키 등록을 참조하십시오.
- 서명할 수 있는 ELF 형식의 커널 이미지가 있습니다.

절차

- x64 아키텍처의 경우:

a.

서명된 이미지를 생성합니다.

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

version **을** **vmlinuz** **파일의 버전 접미사로 바꾸고** ***Custom Secure Boot 키*** **를 이전에 선택한 이름으로 교체합니다.**

b.

선택 사항: 서명 확인:

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

c.

서명되지 않은 이미지를 서명된 이미지로 덮어씁니다.

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

- **64비트** **ARM** **아키텍처에서는 다음을 수행합니다.**

a.

vmlinuz **파일의 압축을 풉니다.**

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

b.

서명된 이미지를 생성합니다.

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

c.

선택 사항: 서명 확인:

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

d. **vmlinux** 파일을 압축합니다.

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

e. 압축되지 않은 **vmlinux** 파일을 제거합니다.

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

19.10. 개인 키를 사용하여 GRUB 빌드 서명

UEFI Secure Boot 메커니즘이 활성화된 시스템에서 사용자 지정 기존 개인 키를 사용하여 **GRUB** 빌드에 서명할 수 있습니다. 사용자 지정 **GRUB** 빌드를 사용하거나 시스템에서 **Microsoft** 신뢰 앵커를 제거한 경우 이 작업을 수행해야 합니다.

사전 요구 사항

- 공개 키 및 개인 키 쌍을 생성하고 공개 키의 유효 날짜를 알고 있습니다. 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.
- 대상 시스템에 공개 키를 등록했습니다. 자세한 내용은 공개 키를 **MOK** 목록에 추가하여 대상 시스템에서 공개 키 등록을 참조하십시오.
- 서명할 수 있는 **GRUB EFI** 바이너리가 있습니다.

절차

- **x64** 아키텍처의 경우:

a. 서명된 **GRUB EFI** 바이너리를 만듭니다.

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

Custom Secure Boot key 를 이전에 선택한 이름으로 교체합니다.

b.
선택 사항: 서명 확인:

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

c.
서명되지 않은 바이너리를 서명된 바이너리로 덮어씁니다.

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

- 64비트 ARM 아키텍처에서는 다음을 수행합니다.

a.
서명된 GRUB EFI 바이너리를 만듭니다.

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

Custom Secure Boot 키 를 이전에 선택한 이름으로 교체합니다.

b.
선택 사항: 서명 확인:

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

c.
서명되지 않은 바이너리를 서명된 바이너리로 덮어씁니다.

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

19.11. 개인 키를 사용하여 커널 모듈에 서명

UEFI Secure Boot 메커니즘이 활성화된 경우 서명된 커널 모듈을 로드하여 시스템의 보안을 향상시킬 수 있습니다.

서명된 커널 모듈은 UEFI Secure Boot가 비활성화된 시스템이나 UEFI가 아닌 시스템에서도 로드할

수 있습니다. 따라서 서명된 커널 모듈과 서명되지 않은 버전의 커널 모듈을 모두 제공할 필요가 없습니다.

사전 요구 사항

- 공개 키 및 개인 키 쌍을 생성하고 공개 키의 유효 날짜를 알고 있습니다. 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.
- 대상 시스템에 공개 키를 등록했습니다. 자세한 내용은 공개 키를 **MOK** 목록에 추가하여 대상 시스템에서 공개 키 등록을 참조하십시오.
- **ELF** 이미지 형식의 커널 모듈이 서명에 사용할 수 있습니다.

절차

1. 공개 키를 **sb_cert.cer** 파일로 내보냅니다.

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

2. **NSS** 데이터베이스에서 **PKCS #12** 파일로 키를 추출합니다.

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

3. 이전 명령을 실행하면 개인 키를 암호화하는 새 암호를 입력합니다.

4. 암호화되지 않은 개인 키를 내보냅니다.

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

중요

암호화되지 않은 개인 키를 처리합니다.

5. 커널 모듈에 서명합니다. 다음 명령은 커널 모듈 파일의 **ELF** 이미지에 서명을 직접 추가합니다.

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

이제 커널 모듈을 로드할 준비가 되었습니다.

중요

Red Hat Enterprise Linux 9에서 키 쌍의 유효 날짜는 중요합니다. 키는 만료되지 않지만 서명 키의 유효 기간 내에 커널 모듈에 서명해야 합니다. **sign-file** 유틸리티는 이에 대해 경고하지 않습니다. 예를 들어 **2021**에서만 유효한 키는 해당 키로 **2021**년에 서명된 커널 모듈을 인증하는 데 사용할 수 있습니다. 그러나 사용자는 **2022**년 커널 모듈에 서명하기 위해 해당 키를 사용할 수 없습니다.

검증

1. 커널 모듈 서명에 대한 정보를 표시합니다.

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

서명에 생성 중 입력한 이름이 나열되어 있는지 확인합니다.

참고

첨부된 서명은 **ELF** 이미지에 포함되어 있지 않으며 **ELF** 이미지의 공식적인 부분이 아닙니다. 따라서 **readelf** 와 같은 유틸리티에서는 커널 모듈에 서명을 표시할 수 없습니다.

2. 모듈을 로드합니다.

```
# insmod my_module.ko
```

3. 모듈을 제거(미로드)합니다.

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

추가 리소스

- 커널 모듈에 대한 정보 표시

19.12. 서명된 커널 모듈 로드

공개 키가 시스템 키링(**.builtin_trusted_keys**) 및 **MOK** 목록에 등록되면, 각 커널 모듈에 개인 키로 서명한 후 **modprobe** 명령을 사용하여 서명된 커널 모듈을 로드할 수 있습니다.

사전 요구 사항

- 공개 및 개인 키 쌍을 생성했습니다. 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.
- 공개 키를 시스템 인증 키에 등록했습니다. 자세한 내용은 공개 키를 **MOK** 목록에 추가하여 대상 시스템에서 공개 키 등록을 참조하십시오.
- 개인 키를 사용하여 커널 모듈을 서명했습니다. 자세한 내용은 개인 키를 사용하여 커널 모듈 서명을 참조하십시오.
- **/lib/modules/$(uname -r)/extra/** 디렉터리를 생성하는 **kernel-modules-extra** 패키지를 설치합니다.

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

절차

1. 공개 키가 시스템 인증 키에 있는지 확인합니다.

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

2. 커널 모듈을 원하는 커널의 **extra/** 디렉터리에 복사합니다.

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

3. 모듈식 종속성 목록을 업데이트합니다.

```
# depmod -a
```

4. 커널 모듈을 로드합니다.

```
# modprobe -v my_module
```

5. 선택적으로 부팅 시 모듈을 로드하려면 **/etc/modules-loaded.d/*my_module*.conf** 파일에 추가합니다.

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

검증

- 모듈이 성공적으로 로드되었는지 확인합니다.

```
# lsmod | grep my_module
```

추가 리소스

- 커널 모듈 관리

20장. SECURE BOOT REVOCATION LIST 업데이트

Secure Boot에서 알려진 보안 문제가 있는 소프트웨어를 식별하고 부팅 프로세스가 손상되지 않도록 시스템에서 UEFI Secure Boot Revocation 목록을 업데이트할 수 있습니다.

20.1. 사전 요구 사항

- 시스템에서 Secure Boot가 활성화됩니다.

20.2. UEFI SECURE BOOT

UEFI(*Unified Extensible Firmware Interface*) Secure Boot 기술을 사용하면 신뢰할 수 있는 키로 서명되지 않은 커널 공간 코드의 실행을 방지할 수 있습니다. 시스템 부트 로더는 암호화 키를 사용하여 서명됩니다. 펌웨어에 포함된 공개 키의 데이터베이스는 서명 키를 인증합니다. 나중에 다음 단계 부트 로더 및 커널에서 서명을 확인할 수 있습니다.

UEFI Secure Boot는 다음과 같이 펌웨어에서 서명된 드라이버 및 커널 모듈로 신뢰 체인을 설정합니다.

- UEFI 개인 키 서명과 공개 키는 shim 첫 번째 단계 부트 로더를 인증합니다. *인증 기관* (CA)은 공개 키에 서명합니다. CA는 펌웨어 데이터베이스에 저장됩니다.
- shim 파일에는 GRUB 부트 로더와 커널을 인증하는 Red Hat 공개 키 Red Hat Secure Boot (CA 키 1)가 포함되어 있습니다.
- 커널에 차례로 드라이버와 모듈을 인증하는 공개 키가 포함되어 있습니다.

Secure Boot는 UEFI 사양의 부팅 경로 검증 구성 요소입니다. 사양은 다음을 정의합니다.

- 비휘발성 스토리지에서 암호로 보호되는 UEFI 변수를 위한 프로그래밍 인터페이스입니다.
- UEFI 변수에 신뢰할 수 있는 X.509 루트 인증서를 저장합니다.

- 부트 로더 및 드라이버와 같은 **UEFI** 애플리케이션 검증.
- 알려진**-bad** 인증서 및 애플리케이션 해시를 취소하는 절차입니다.

UEFI Secure Boot는 무단 변경 사항을 감지하는 데 도움이 되지만 다음과 같은 것은 아닙니다.

- 두 번째 단계 부트 로더의 설치 또는 제거를 방지합니다.
- 이러한 변경에 대한 명확한 사용자 확인이 필요합니다.
- 부팅 경로 조작을 중지합니다. 부트 로더가 설치 또는 업데이트될 때 아닌 부팅 중에 서명을 확인합니다.

부트 로더 또는 커널이 시스템 신뢰할 수 있는 키로 서명되지 않은 경우 **Secure Boot**가 시작되지 않습니다.

20.3. SECURE BOOT REVOCATION LIST

UEFI Secure Boot Revocation List 또는 **Secure Boot Forbidden Signature Database (dbx)**는 **Secure Boot**에서 더 이상 실행할 수 없는 소프트웨어를 식별하는 목록입니다.

GRUB 부트 로더와 같이 **Secure Boot**와 상호 작용하는 소프트웨어에서 보안 문제 또는 안정성 문제가 발견되면 **Revocation List**에서 해시 서명을 저장합니다. 이러한 인식된 서명이 있는 소프트웨어는 부팅 중에 실행할 수 없으며 시스템 부팅 시 시스템을 손상시키지 못합니다.

예를 들어 특정 버전의 **GRUB**에는 공격자가 **Secure Boot** 메커니즘을 우회할 수 있는 보안 문제가 포함될 수 있습니다. 문제가 발견되면 **Revocation List**는 문제가 포함된 모든 **GRUB** 버전의 해시 서명을 추가합니다. 결과적으로 보안 **GRUB** 버전만 시스템에서 부팅할 수 있습니다.

취소 목록을 사용하려면 새로 발견된 문제를 인식하기 위해 정기적으로 업데이트해야 합니다. **Revocation List**를 업데이트할 때 현재 설치된 시스템이 더 이상 부팅되지 않도록 하는 안전한 업데이트 방법을 사용해야 합니다.

20.4. 온라인 해지 목록 업데이트 적용

Secure Boot에서 알려진 보안 문제를 방지하도록 시스템의 Secure Boot Revocation List를 업데이트할 수 있습니다. 이 절차는 안전하며 업데이트가 시스템이 부팅되지 않도록 합니다.

사전 요구 사항

- 시스템에서 업데이트를 위해 인터넷에 액세스할 수 있습니다.

절차

1. Revocation List의 현재 버전을 확인합니다.

```
# fwupdmgr get-devices
```

UEFI dbx 아래의 현재 버전 필드를 참조하십시오.

2. LVFS 해지 목록 리포지토리를 활성화합니다.

```
# fwupdmgr enable-remote lvfs
```

3. 리포지토리 메타데이터를 새로 고칩니다.

```
# fwupdmgr refresh
```

4. 취소 목록 업데이트를 적용합니다.

- 명령줄에서 다음을 수행합니다.

```
# fwupdmgr update
```

- 그래픽 인터페이스에서 다음을 수행합니다.

i. 소프트웨어 애플리케이션 열기

ii. 업데이트 탭으로 이동합니다.

iii. **Secure Boot dbx Configuration Update** 항목을 찾습니다.

iv. 업데이트를 클릭합니다.

5. 업데이트가 끝나면 **fwupdmgr** 또는 **Software** 가 시스템을 재부팅하도록 요청합니다. 재부팅을 확인합니다.

검증

- 재부팅 후 현재 버전의 취소 목록을 다시 확인합니다.

```
# fwupdmgr get-devices
```

20.5. 오프라인 해지 목록 업데이트 적용

인터넷에 연결되지 않은 시스템에서는 **Secure Boot Revocation List**를 **RHEL**에서 업데이트하여 알려진 보안 문제를 방지할 수 있습니다. 이 절차는 안전하며 업데이트가 시스템이 부팅되지 않도록 합니다.

절차

1. **Revocation List**의 현재 버전을 확인합니다.

```
# fwupdmgr get-devices
```

UEFI dbx 아래의 현재 버전 필드를 참조하십시오.

2. **RHEL**에서 사용 가능한 업데이트를 나열합니다.

```
# ls /usr/share/dbxtool/
```

3. 아키텍처의 최신 업데이트 파일을 선택합니다. 파일 이름은 다음 형식을 사용합니다.

```
DBXUpdate-date-architecture.cab
```

4. 선택한 업데이트 파일을 설치합니다.

```
# fwupdmgr install /usr/share/dbxtool/DBXUpdate-date-architecture.cab
```

5. 업데이트가 끝나면 **fwupdmgr** 에서 시스템을 재부팅하도록 요청합니다. 재부팅을 확인합니다.

검증

- 재부팅 후 현재 버전의 취소 목록을 다시 확인합니다.

```
# fwupdmgr get-devices
```

21장. 커널 무결성 하위 시스템으로 보안 강화

커널 무결성 하위 시스템의 구성 요소를 사용하여 시스템 보호를 개선할 수 있습니다. 관련 구성 요소 및 해당 구성에 대해 자세히 알아보십시오.

참고

커널 키링 시스템에는 **Red Hat** 서명 키의 인증서만 포함하므로 이 기능을 **Red Hat** 제품용 암호화 서명과 함께 사용할 수 있습니다. 다른 해시 기능을 사용하면 변조가 불완전합니다.

21.1. 커널 무결성 하위 시스템

무결성 하위 시스템은 시스템 데이터의 전체 무결성을 유지 관리하는 커널의 일부입니다. 이 하위 시스템은 시스템 상태를 빌드한 시점과 동일하게 유지하는 데 도움이 됩니다. 이 하위 시스템을 사용하면 특정 시스템 파일의 바람직하지 않은 수정을 방지할 수 있습니다.

커널 무결성 하위 시스템은 다음 두 가지 주요 구성 요소로 구성됩니다.

무결성 측정 아키텍처 (IMA)

- **IMA**는 암호화 방식으로 해시 또는 암호화 키를 사용하여 서명을 통해 실행되거나 열 때마다 파일 콘텐츠를 측정합니다. 키는 커널 키링 하위 시스템에 저장됩니다.
- **IMA**는 측정된 값을 커널의 메모리 공간에 배치합니다. 이렇게 하면 시스템의 사용자가 측정 값을 수정하지 못하도록 합니다.
- **IMA**를 사용하면 로컬 및 원격 당사자가 측정 값을 확인할 수 있습니다.
- **IMA**는 커널 메모리 내의 측정 목록에 이전에 저장된 값에 대해 현재 파일 콘텐츠에 대한 로컬 유효성 검사를 제공합니다. 이 확장은 현재 및 이전 측정값이 일치하지 않는 경우 특정 파일에서 모든 작업을 수행하는 것을 금지합니다.

EVM(Extended Verification Module)

- **EVM**은 **IMA** 측정 및 **SELinux** 속성과 같은 시스템 보안과 관련된 파일의 확장된 속성(*xattr*라고도 함)을 보호합니다. **EVM**은 해당 값을 암호화 방식으로 해시하거나 암호화 키를 사

용하여 서명합니다. 키는 커널 키링 하위 시스템에 저장됩니다.

커널 무결성 하위 시스템은 신뢰할 수 있는 플랫폼 모듈(TPM)을 사용하여 시스템 보안을 강화할 수 있습니다.

TPM은 중요한 암호화 기능을 위해 신뢰할 수 있는 컴퓨팅 그룹(TCG)의 TPM 사양에 따라 빌드되는 통합 암호화 키가 있는 하드웨어, 펌웨어 또는 가상 구성 요소입니다. TPMS는 일반적으로 플랫폼의 마더보드에 연결된 전용 하드웨어로 구축됩니다. 하드웨어 칩의 보호 및 변조 방지 영역에서 암호화 기능을 제공하여 TPM은 소프트웨어 기반 공격으로부터 보호됩니다. TPMS는 다음 기능을 제공합니다.

- 난수 생성기
- 암호화 키를 위한 생성기 및 보안 스토리지
- 해시 생성기
- 원격 인증

추가 리소스

- 보안 강화
- SELinux(Security- Enhanced Linux)의 기본 및 고급 구성

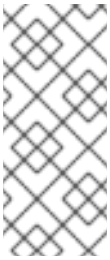
21.2. 신뢰할 수 있는 암호화된 키

*신뢰할 수 있는 키와 암호화된 키*는 시스템 보안을 강화하는 데 중요한 부분입니다.

신뢰할 수 있고 암호화된 키는 커널 키링 서비스를 사용하는 커널에서 생성되는 변수 길이 대칭 키입니다. 키의 무결성을 확인할 수 있습니다. 즉, 실행 중인 시스템의 무결성을 확인하고 확인하기 위해 확장 검증 모듈(EVM)에서 사용할 수 있습니다. 사용자 수준 프로그램은 암호화된 ***Blob*** 형식의 키만 액세스할 수 있습니다.

신뢰할 수 있는 키

신뢰할 수 있는 키에는 키를 생성하고 암호화(seal)하는 데 사용되는 신뢰할 수 있는 플랫폼 모듈(TPM) 칩이 필요합니다. 각 TPM에는 TPM 자체에 저장된 스토리지 루트 키라는 마스터 래핑 키가 있습니다.



참고

RHEL 9에서는 TPM 2.0만 지원합니다. TPM 1.2를 사용해야 하는 경우 RHEL 8을 사용합니다. 자세한 내용은 Red Hat에서 지원하는 Is Trusted Platform Module (TPM) 솔루션을 참조하십시오.

다음 명령을 입력하여 TPM 2.0 칩이 활성화되어 있는지 확인할 수 있습니다.

```
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

TPM 2.0 칩을 활성화하고 머신 펌웨어의 설정을 통해 TPM 2.0 장치를 관리할 수도 있습니다.

또한 신뢰할 수 있는 키를 TPM의 *플랫폼 구성 레지스터* (PCR) 값 세트로 봉인할 수 있습니다. PCR에는 펌웨어, 부트 로더 및 운영 체제를 반영하는 무결성 관리 값 세트가 포함되어 있습니다. 즉, PCR-sealed 키는 암호화 된 동일한 시스템에서 TPM에 의해서만 해독 될 수 있습니다. 그러나 PCRsealed 신뢰할 수 있는 키가 로드되고(인증 키에 추가) 연결된 PCR 값이 확인되면 새 커널(예: 새 커널)을 사용하도록 새 (또는 향후) PCR 값으로 업데이트할 수 있습니다. 단일 키를 각각 다른 PCR 값을 가진 여러 Blob으로 저장할 수 있습니다.

암호화된 키

암호화된 키에는 AES(커널 고급 암호화 표준)를 사용하므로 TPM이 필요하지 않으므로 신뢰할 수 있는 키보다 더 빠릅니다. 암호화된 키는 커널 생성 임의 숫자를 사용하여 생성되며, 사용자 공간 Blob으로 내보낼 때 *마스터 키로* 암호화됩니다.

마스터 키는 신뢰할 수 있는 키 또는 사용자 키입니다. 마스터 키가 신뢰할 수 없는 경우 암호화 키는 암호화에 사용되는 사용자 키만큼만 안전합니다.

21.3. 신뢰할 수 있는 키로 작업

keyctl 유틸리티를 사용하여 신뢰할 수 있는 키를 생성, 내보내기, 로드 및 업데이트할 수 있습니다.

사전 요구 사항

- 신뢰할 수 있는 플랫폼 모듈(TPM)이 활성화되어 활성화되어 있습니다. 커널 무결성 하위 시스템 및 신뢰할 수 있고 암호화된 키를 참조하십시오.

  tpm2_pcrread 명령을 입력하여 시스템에 **TPM**이 있는지 확인할 수 있습니다. 이 명령의 출력에 여러 해시가 표시되면 **TPM**이 있습니다.

절차

1. 다음 유틸리티 중 하나를 사용하여 영구 처리(예: ***81000001***)가 있는 **SHA-256** 기본 스토리지 키로 **2048**비트 **RSA** 키를 생성합니다.

   a. **tss2** 패키지를 사용하여 다음을 수행합니다.

   ```
   # TPM_DEVICE=/dev/tpm0 tsscreateprimary -hi o -st
   Handle 80000000
   # TPM_DEVICE=/dev/tpm0 tssevictcontrol -hi o -ho 80000000 -hp 81000001
   ```

   b. **tpm2-tools** 패키지를 사용하여 다음을 수행합니다.

   ```
   # tpm2_createprimary --key-algorithm=rsa2048 --key-context=key.ctxt
   name-alg:
     value: sha256
     raw: 0xb
   …
   sym-keybits: 128
   rsa: xxxxxx…

   # tpm2_evictcontrol -c key.ctxt 0x81000001
   persistentHandle: 0x81000001
   action: persisted
   ```

2. **key ctl add trusted <** ***NAME*** **> "new <** ***KEY_LENGTH*** **> keyhandle= <** ***PERSISTENT-HANDLE*** **> [options]" <** ***KEYRING*** **>**과 함께 **TPM 2.0**을 사용하여 신뢰할 수 있는 키를 만듭니다. 이 예에서 영구 처리는 ***81000001*** 입니다.

   ```
   # keyctl add trusted kmk "new 32 keyhandle=0x81000001" @u
   642500861
   ```

   이 명령은 **32** 바이트(**256**비트) 길이로 **kmk** 라는 신뢰할 수 있는 키를 생성하여 사용자 인증 키(**@u**)에 배치합니다. 키 길이는 **32~128**바이트(**256~24**비트)일 수 있습니다.

3. 커널 인증 키의 현재 구조를 나열합니다.

```
# keyctl show
Session Keyring
      -3 --alswrv    500   500  keyring: ses 97833714 --alswrv 500 -1 \ keyring: uid.1000
642500861 --alswrv 500 500 \ trusted: kmk
```

4. 신뢰할 수 있는 키의 일련 번호를 사용하여 사용자 공간 Blob으로 키를 내보냅니다.

```
# keyctl pipe 642500861 > kmk.blob
```

명령은 pipe 하위 명령과 kmk 의 일련 번호를 사용합니다.

5. 사용자 공간 Blob에서 신뢰할 수 있는 키를 로드합니다.

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

6. TPM-sealed 신뢰할 수 있는 키(kmk)를 사용하는 안전한 암호화된 키를 만듭니다. 다음 구문을 따르십시오. *keyctl add encrypted <NAME> "new [FORMAT] <KEY_TYPE>:<PRI Cryostat_KEY_NAME> <KEY_LENGTH>" <KEYRING >* :

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

추가 리소스

- keyctl(1) 매뉴얼 페이지
- 신뢰할 수 있는 암호화된 키
- 커널 키 보존 서비스
- 커널 무결성 하위 시스템

21.4. 암호화된 키 작업

암호화된 키를 관리하여 신뢰할 수 있는 플랫폼 모듈(TPM)을 사용할 수 없는 시스템에서 시스템 보안을 개선할 수 있습니다.

절차

1. 임의의 숫자 시퀀스를 사용하여 사용자 키를 생성합니다.

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

이 명령은 *기본 키 역할을 하고 실제 암호화된 키*를 봉인하는 데 사용되는 **kmk-user** 라는 사용자 키를 생성합니다.

2. 이전 단계의 기본 키를 사용하여 암호화된 키를 생성합니다.

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

3. 선택적으로 지정된 사용자 인증 키의 모든 키를 나열합니다.

```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv  1000  1000 user: kmk-user
1012412758: --alswrv  1000  1000 encrypted: encr-key
```

중요

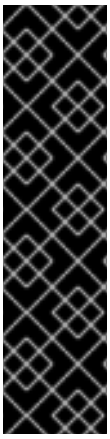
신뢰할 수 있는 기본 키로 봉인되지 않은 암호화된 키는 암호화에 사용된 사용자 기본 키(임의 숫자 키)만큼 안전합니다. 따라서 기본 사용자 키를 최대한 안전하게 로드하고 부팅 프로세스 중 조기에 로드됩니다.

추가 리소스

- **keyctl(1)** 매뉴얼 페이지
- 커널 키 보존 서비스

21.5. IMA 및 EVM 활성화

무결성 측정 아키텍처(IMA) 및 EVM(확장 확인 모듈)을 활성화하고 구성하여 운영 체제의 보안을 개선할 수 있습니다.



중요

항상 IMA와 함께 EVM을 활성화합니다.

EVM만 활성화할 수 있지만 EVM 평가는 IMA 평가 규칙에 의해서만 트리거됩니다. 따라서 EVM은 SELinux 속성과 같은 파일 메타데이터를 보호하지 않습니다. 파일 메타데이터가 오프라인으로 변조되는 경우 EVM은 파일 메타데이터 변경만 방지할 수 있습니다. 파일 실행과 같은 파일 액세스를 차단하지 않습니다.

사전 요구 사항

- Secure Boot는 일시적으로 비활성화되어 있습니다.

참고

Secure Boot가 활성화되면 ima_appraise=fix 커널 명령줄 매개변수가 작동하지 않습니다.

- securityfs 파일 시스템은 /sys/kernel/security/ 디렉터리에 마운트되고 /sys/kernel/security/integrity/ima/ 디렉터리가 있습니다. mount 명령을 사용하여 securityfs 가 마운트된 위치를 확인할 수 있습니다.

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

- systemd 서비스 관리자는 부팅 시 IMA 및 EVM을 지원하도록 패치됩니다. 다음 명령을 사용하여 확인합니다.

```
# grep <options> pattern <files>
```

예를 들면 다음과 같습니다.

```
# dmesg | grep -i -e EVM -e IMA -w
[ 0.943873] ima: No TPM chip found, activating TPM-bypass!
[ 0.944566] ima: Allocated hash algorithm: sha256
[ 0.944579] ima: No architecture policies found
[ 0.944601] evm: Initialising EVM extended attributes:
[ 0.944602] evm: security.selinux
[ 0.944604] evm: security.SMACK64 (disabled)
[ 0.944605] evm: security.SMACK64EXEC (disabled)
[ 0.944607] evm: security.SMACK64TRANSMUTE (disabled)
[ 0.944608] evm: security.SMACK64MMAP (disabled)
[ 0.944609] evm: security.apparmor (disabled)
[ 0.944611] evm: security.ima
[ 0.944612] evm: security.capability
[ 0.944613] evm: HMAC attrs: 0x1
[ 1.314520] systemd[1]: systemd 252-18.el9 running in system mode (+PAM +AUDIT
+SELINUX -APPARMOR +IMA +SMACK +SECCOMP +GCRYPT +GNUTLS +OPENSSL
+ACL +BLKID +CURL +ELFUTILS -FIDO2 +IDN2 -IDN -IPTC +KMOD +LIBCRYPTSETUP
+LIBFDISK +PCRE2 -PWQUALITY +P11KIT -QRENCODE +TPM2 +BZIP2 +LZ4 +XZ
+ZLIB +ZSTD -BPF_FRAMEWORK +XKBCOMMON +UTMP +SYSVINIT default-
hierarchy=unified)
[ 1.717675] device-mapper: core: CONFIG_IMA_DISABLE_HTABLE is disabled.
Duplicate IMA measurements will not be recorded in the IMA log.
[ 4.799436] systemd[1]: systemd 252-18.el9 running in system mode (+PAM +AUDIT
+SELINUX -APPARMOR +IMA +SMACK +SECCOMP +GCRYPT +GNUTLS +OPENSSL
+ACL +BLKID +CURL +ELFUTILS -FIDO2 +IDN2 -IDN -IPTC +KMOD +LIBCRYPTSETUP
+LIBFDISK +PCRE2 -PWQUALITY +P11KIT -QRENCODE +TPM2 +BZIP2 +LZ4 +XZ
+ZLIB +ZSTD -BPF_FRAMEWORK +XKBCOMMON +UTMP +SYSVINIT default-
hierarchy=unified)
```

절차

1. 현재 부팅 항목에 대한 *수정* 모드에서 **IMA** 및 **EVM**을 활성화하고 사용자가 다음 커널 명령줄 매개변수를 추가하여 **IMA** 측정값을 수집하고 업데이트할 수 있습니다.

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

이 명령을 사용하면 현재 부팅 항목에 대한 *수정* 모드에서 **IMA** 및 **EVM**을 활성화하고 사용자가 **IMA** 측정을 수집하고 업데이트할 수 있습니다.

ima_policy=appraise_tcb 커널 명령줄 매개 변수를 사용하면 커널이 기본 **trusted Computing Base(ECDHEB)** 측정 정책 및 평가 단계를 사용합니다. 평가 단계는 이전 및 현재 측정값이 일치하지 않는 파일에 대한 액세스를 금지합니다.

2. 재부팅하여 변경 사항을 적용합니다.

3. 선택 사항: 매개변수가 커널 명령줄에 추가되었는지 확인합니다.

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.14.0-1.el9.x86_64 root=/dev/mapper/rhel-root
ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M 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. EVM 키를 보호하기 위해 커널 마스터 키를 생성합니다.

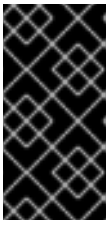
```
# keyctl add user kmk "$(dd if=/dev/urandom bs=1 count=32 2> /dev/null)" @u
748544121
```

kmk 는 커널 공간 메모리에 전적으로 유지됩니다. kmk 의 32바이트 긴 값은 /dev/urandom 파일에서 임의의 바이트에서 생성되고 사용자(@u) 인증 키에 배치됩니다. 키 일련 번호는 이전 출력의 첫 번째 행에 있습니다.

5. kmk 를 기반으로 암호화된 EVM 키를 만듭니다.

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

명령은 kmk 를 사용하여 64바이트 긴 사용자 키(evm-key)를 생성하고 암호화하여 사용자(@u) 인증 키에 배치합니다. 키 일련 번호는 이전 출력의 첫 번째 행에 있습니다.



중요

EVM 하위 시스템 이름이 예상하고 있으므로 사용자 키 이름을 evm-key 로 지정해야 합니다.

6. 내보낸 키의 디렉터리를 만듭니다.

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

7. kmk 를 검색하고 암호화되지 않은 값을 새 디렉터리로 내보냅니다.

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

8. **evm-key** 를 검색하고 암호화된 값을 새 디렉터리로 내보냅니다.

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

evm-key 는 이전에 커널 마스터 키로 암호화되었습니다.

9. 선택 사항: 새로 생성된 키를 확인합니다.

```
# 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. 선택 사항: 예를 들어 시스템을 재부팅한 후 인증 키에서 키가 제거된 경우 새 키를 생성하는 대신 이미 내보낸 **kmk** 및 **evm-key** 를 가져올 수 있습니다.

a. **kmk** 를 가져옵니다.

```
# keyctl add user kmk "$(cat /etc/keys/kmk)" @u
451342217
```

b. **evm-key** 를 가져옵니다.

```
# keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
924537557
```

11. **EVM**을 활성화합니다.

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

12. 전체 시스템의 레이블을 다시 지정합니다.

```
# find / -fstype xfs -type f -uid 0 -exec head -n 1 '{}' >/dev/null \;
```

주의

시스템의 레이블을 다시 지정하지 않고 **IMA** 및 **EVM**을 활성화하면 시스템의 대부분의 파일에 액세스할 수 없게 될 수 있습니다.

검증

- **EVM**이 초기화되었는지 확인합니다.

```
# dmesg | tail -1
[…] evm: key initialized
```

추가 리소스

- grep(1) 도움말 페이지
- 커널 무결성 하위 시스템
- 신뢰할 수 있는 암호화된 키

21.6. 무결성 측정 아키텍처를 사용하여 파일 해시 수집

측정 단계에서는 파일 해시를 생성하여 해당 파일의 확장 속성(***xattrs***)으로 저장할 수 있습니다. 파일 해시를 사용하면 **RSA** 기반 디지털 서명 또는 **HMAC-SHA1**(HMAC-SHA1)을 생성하여 확장된 속성에 대한 오프라인 변조 공격을 방지할 수 있습니다.

사전 요구 사항

- **IMA** 및 **EVM**이 활성화됩니다. 자세한 내용은 무결성 측정 아키텍처 활성화 및 확장 검증 모듈을 참조하십시오.

- 신뢰할 수 있는 유효한 키 또는 암호화된 키는 커널 인증 키에 저장됩니다.

- ima-evm-utils,attr 및 keyutils 패키지가 설치됩니다.

절차

1. 테스트 파일을 생성합니다.

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

IMA 및 EVM은 test_file 예제 파일에 확장 속성으로 저장된 해시 값을 할당했는지 확인합니다.

2. 파일의 확장 속성을 검사합니다.

```
# getfattr -m . -d test_file
# file: test_file
security.evm=0sAnDIy4VPA0HArpPO/EqiutnNyBql
security.ima=0sAQOEDeuUnWzwwKYk+n66h/vby3eD
```

예제 출력에는 IMA 및 EVM 해시 값과 SELinux 컨텍스트가 포함된 확장된 속성이 표시되어 있습니다. EVM은 다른 특성과 관련된 security.evm extended 속성을 추가합니다. 이 시점에서 security.evm 에서 evmctl 유틸리티를 사용하여 RSA 기반 디지털 서명 또는 HMAC-SHA1(Hash-based Message Authentication Code)을 생성할 수 있습니다.

추가 리소스

- 보안 강화

21.7. 패키지 파일에 IMA 서명 추가

커널, Keylime, fapolicyd, debuginfo 패키지를 허용하려면 RPM 파일에 IMA 서명을 추가하여 무결성 검사를 수행해야 합니다. rpm-plugin-ima 플러그인을 설치한 후 새로 설치된 RPM 파일에 자동으로 IMA 서명이 security.ima 확장 파일 속성에 배치됩니다. 그러나 IMA 서명을 얻으려면 기존 패키지를 다시 설치해야 합니다.

절차

1. rpm-plugin-ima 플러그인을 설치하려면 다음을 실행합니다.

```
# dnf install rpm-plugin-ima -y
```

2. 모든 패키지를 다시 설치하려면 다음을 실행합니다.

```
# dnf reinstall “*” -y
```

검증

1. 다시 설치한 패키지 파일에 유효한 IMA 서명이 있는지 확인합니다. 예를 들어 /usr/bin/bash 파일의 IMA 서명을 확인하려면 다음을 실행합니다.

```
# getfattr -m security.ima -d /usr/bin/bash
security.ima=0sAwIE0zIESQBnMGUCMFhf0iBeM7NjjhCCHVt4/ORx1eCegjrWSHzFbJM
CsAhR9bYU2hNGjiWUYT2IlqWaaAIxALFGUkqGP5vDLuxQXibO9g7HFcfyZzRBY4rbKP
sXcAlZRtDHVS5dQBZqM3hyS5v1MA==
```

2. 지정된 인증서를 사용하여 IMA 서명이 있는지 확인합니다. IMA 코드 서명 키는 /usr/share/doc/kernel-keys/$(uname -r)/ima.cer 에서 액세스할 수 있습니다.

```
# evmctl ima_verify -k /usr/share/doc/kernel-keys/$(uname -r)/ima.cer /usr/bin/bash
key 1: d3320449 /usr/share/doc/kernel-keys/5.14.0-359.el9.x86-64/ima.cer
/usr/bin/bash: verification is OK
```

21.8. 커널 런타임 무결성 모니터링 활성화

IMA 평가에서 제공하는 커널 런타임 무결성 모니터링을 활성화할 수 있습니다.

사전 요구 사항

- 시스템에 설치된 커널에 는 5.14.0-359 이상의 버전이 있습니다.
- dracut 패키지에는 057-43.git20230816 버전이 있습니다.
- keyutils 패키지가 설치되어 있습니다.

- ima-evm-utils 패키지가 설치되어 있습니다.

- 정책에서 다루는 파일에는 유효한 서명이 있습니다. 자세한 내용은 패키지 파일에 IMA 서명 추가를 참조하십시오.

절차

1. Red Hat IMA 코드 서명 키를 /etc/ima/keys 파일에 복사하려면 다음을 실행합니다.

```
$ mkdir -p /etc/keys/ima
$ cp /usr/share/doc/kernel-keys/$(uname -r)/ima.cer /etc/ima/keys
```

2. IMA 코드 서명 키를 .ima 인증 키에 추가하려면 다음을 실행합니다.

```
# keyctl padd asymmetric RedHat-IMA %:.ima < /etc/ima/keys/ima.cer
```

3. 위협 모델에 따라 /etc/sysconfig/ima-policy 파일에 IMA 정책을 정의합니다. 예를 들어 다음 IMA 정책은 실행 파일과 관련 메모리 매핑 라이브러리 파일의 무결성을 확인합니다.

```
# PROC_SUPER_MAGIC = 0x9fa0
dont_appraise fsmagic=0x9fa0
# SYSFS_MAGIC = 0x62656572
dont_appraise fsmagic=0x62656572
# DEBUGFS_MAGIC = 0x64626720
dont_appraise fsmagic=0x64626720
# TMPFS_MAGIC = 0x01021994
dont_appraise fsmagic=0x1021994
# RAMFS_MAGIC
dont_appraise fsmagic=0x858458f6
# DEVPTS_SUPER_MAGIC=0x1cd1
dont_appraise fsmagic=0x1cd1
# BINFMTFS_MAGIC=0x42494e4d
dont_appraise fsmagic=0x42494e4d
# SECURITYFS_MAGIC=0x73636673
dont_appraise fsmagic=0x73636673
# SELINUX_MAGIC=0xf97cff8c
dont_appraise fsmagic=0xf97cff8c
# SMACK_MAGIC=0x43415d53
dont_appraise fsmagic=0x43415d53
# NSFS_MAGIC=0x6e736673
dont_appraise fsmagic=0x6e736673
# EFIVARFS_MAGIC
dont_appraise fsmagic=0xde5e81e4
# CGROUP_SUPER_MAGIC=0x27e0eb
dont_appraise fsmagic=0x27e0eb
```

```
# CGROUP2_SUPER_MAGIC=0x63677270
dont_appraise fsmagic=0x63677270
appraise func=BPRM_CHECK
appraise func=FILE_MMAP mask=MAY_EXEC
```

4. 커널이 이 IMA 정책을 수락하는지 확인하기 위해 IMA 정책을 로드하려면 다음을 실행합니다.

```
# echo /etc/sysconfig/ima-policy > /sys/kernel/security/ima/policy
# echo $?
0
```

5. dracut 무결성 모듈이 IMA 코드 서명 키와 IMA 정책을 자동으로 로드하도록 활성화하려면 다음을 실행합니다.

```
# echo 'add_dracutmodules+=" integrity "' > /etc/dracut.conf.d/98-integrity.conf
# dracut -f
```

21.9. OPENSSL을 사용하여 사용자 정의 IMA 키 생성

OpenSSL 을 사용하여 코드를 보호하기 위해 디지털 인증서에 대한 CSR을 생성할 수 있습니다.

커널은 IMA 서명을 확인하기 위해 코드 서명 키를 .ima 인증 키를 검색합니다. .ima 인증 키에 코드 서명 키를 추가하기 전에 IMA CA 키가 .builtin_trusted_keys 또는 .secondary_trusted_keys 키 링에서 이 키를 서명했는지 확인해야 합니다.

사전 요구 사항

- 사용자 정의 IMA CA 키에는 다음과 같은 확장 기능이 있습니다.
  - CA 부울이 선언된 기본 제약 조건 확장입니다.
  - CertSign 비트가 선언되었지만 Digital Signature 가 선언되지 않은 키 Usage 확장입니다.
- 사용자 정의 IMA 코드 서명 키는 다음 기준에 따릅니다.
  - IMA CA 키는 이 사용자 정의 IMA 코드 서명 키에 서명했습니다.

- 사용자 지정 키에는 **subjectKeyIdentifier** 확장이 포함됩니다.

절차

1. 사용자 정의 **IMA CA** 키 쌍을 생성하려면 다음을 실행합니다.

```
# openssl req -new -x509 -utf8 -sha256 -days 3650 -batch -config ima_ca.conf -outform
DER -out custom_ima_ca.der -keyout custom_ima_ca.priv
```

2. *선택 사항*: **ima_ca.conf** 파일의 내용을 확인하려면 다음을 실행합니다.

```
# cat ima_ca.conf
[ req ]
default_bits = 2048
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = ca

[ req_distinguished_name ]
O = YOUR_ORG
CN =  YOUR_COMMON_NAME IMA CA
emailAddress = YOUR_EMAIL

[ ca ]
basicConstraints=critical,CA:TRUE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
keyUsage=critical,keyCertSign,cRLSign
```

3. **IMA** 코드 서명 키에 대한 개인 키 및 **CSR**(인증서 서명 요청)을 생성하려면 다음을 실행합니다.

```
# openssl req -new -utf8 -sha256 -days 365 -batch -config ima.conf -out
custom_ima.csr -keyout custom_ima.priv
```

4. *선택 사항*: **ima.conf** 파일의 내용을 확인하려면 다음을 실행합니다.

```
# cat ima.conf
[ req ]
default_bits = 2048
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = code_signing
```

```
[ req_distinguished_name ]
O = YOUR_ORG
CN = YOUR_COMMON_NAME IMA signing key
emailAddress = YOUR_EMAIL

[ code_signing ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
```

5. IMA CA 개인 키를 사용하여 CSR에 서명하여 IMA 코드 서명 인증서를 생성합니다.

```
# openssl x509 -req -in custom_ima.csr -days 365 -extfile ima.conf -extensions
code_signing -CA custom_ima_ca.der -CAkey custom_ima_ca.priv -CAcreateserial -
outform DER -out ima.der
```

21.10. UEFI 시스템을 위한 사용자 정의 서명된 IMA 정책 배포

Secure Boot 환경에서는 사용자 정의 IMA 키에서 서명한 서명된 IMA 정책만 로드해야 할 수 있습니다.

사전 요구 사항

- MOK 목록에는 사용자 지정 IMA 키가 포함되어 있습니다. 자세한 내용은 MOK 목록에 공개 키를 추가하여 대상 시스템에서 공개 키 등록을 참조하십시오.
- 시스템에 설치된 커널에는 5.14.0-335 버전이 있습니다.

절차

1. Secure Boot를 활성화합니다.
2. ima_policy=secure_boot 커널 매개변수를 영구적으로 추가합니다.

   자세한 내용은 sysctl을 사용하여 커널 매개변수 구성을 참조하십시오.
3. 명령을 실행하여 IMA 정책을 준비합니다.

```
# evmctl ima_sign /etc/sysconfig/ima-policy -k
<PATH_TO_YOUR_CUSTOM_IMA_KEY>
Place your public certificate under /etc/keys/ima/ and add it to the .ima keyring
```

4. 명령을 실행하여 사용자 정의 **IMA** 코드 서명 키를 사용하여 정책에 서명합니다.

```
# keyctl padd asymmetric CUSTOM_IMA1 %:.ima < /etc/ima/keys/my_ima.cer
```

5. 명령을 실행하여 **IMA** 정책을 로드합니다.

```
# echo /etc/sysconfig/ima-policy > /sys/kernel/security/ima/policy
# echo $?
0
```

22장. SYSTEMD를 사용하여 애플리케이션에서 사용하는 리소스 관리

RHEL 9는 cgroup 계층 구조의 시스템에 systemd 장치 트리를 바인딩하여 리소스 관리 설정을 프로세스 수준에서 애플리케이션 수준으로 이동합니다. 따라서 systemctl 명령을 사용하거나 systemd 장치 파일을 수정하여 시스템 리소스를 관리할 수 있습니다.

이를 위해 systemd 는 유닛 파일의 다양한 구성 옵션을 사용하거나 systemctl 명령을 통해 직접 사용합니다. 그런 다음 systemd 는 cgroup 및 네임스페이스 와 같은 Linux 커널 시스템 호출 및 기능을 사용하여 해당 옵션을 특정 프로세스 그룹에 적용합니다.

참고

다음 도움말 페이지에서 systemd 에 대한 전체 구성 옵션 세트를 검토할 수 있습니다.

- systemd.resource-control(5)
- systemd.exec(5)

22.1. 리소스 관리에서 SYSTEMD의 역할

systemd 의 핵심 기능은 서비스 관리 및 감독입니다. systemd 시스템 및 서비스 관리자:

- 부팅 프로세스 중에 관리 서비스가 적시에 올바른 순서로 시작되는지 확인합니다.
- 관리 서비스가 원활하게 실행되도록 기본 하드웨어 플랫폼을 최적으로 사용할 수 있습니다.
- 리소스 관리 정책을 정의하는 기능을 제공합니다.
- 서비스 성능을 향상시킬 수 있는 다양한 옵션을 조정하는 기능을 제공합니다.

중요

일반적으로 **Red Hat**은 시스템 리소스 사용량을 제어하기 위해 **systemd** 를 사용할 것을 권장합니다. 특수한 경우에만 **cgroup** 가상 파일 시스템을 수동으로 구성해야 합니다. 예를 들어 **cgroup-v2** 계층 구조에 동등한 권한이 없는 **cgroup-v1** 컨트롤러를 사용해야 하는 경우입니다.

22.2. 시스템 소스의 배포 모델

시스템 리소스 배포를 수정하려면 다음 배포 모델 중 하나 이상을 적용할 수 있습니다.

weights

모든 하위 그룹의 가중치를 추가하고 각 하위 그룹에 합계와 일치하는 부분을 지정하여 리소스를 배포할 수 있습니다.

예를 들어 **10**개의 **cgroups**가 있고 각각 값이 **100**인 경우 합계는 **1000**입니다. 각 **cgroup**은 **10**번째 리소스 **1**개를 수신합니다.

weight는 일반적으로 상태 비저장 리소스를 배포하는 데 사용됩니다. 예를 들어 ***CPUWeight=*** 옵션은 이 리소스 배포 모델의 구현입니다.

제한

cgroup은 구성된 리소스 양까지 사용할 수 있습니다. 하위 그룹 제한 합계는 상위 **cgroup** 제한을 초과할 수 있습니다. 따라서 이 모델에서 리소스를 과다 할당할 수 있습니다.

예를 들어 ***MemoryMax=*** 옵션은 이 리소스 배포 모델의 구현입니다.

보호

cgroup에 대해 보호된 리소스 양을 설정할 수 있습니다. 리소스 사용량이 보호 경계 아래에 있는 경우 커널은 동일한 리소스에 경쟁하는 다른 **cgroups**에 대해 이 **cgroup**에 페달하지 않도록 합니다. 오버 커밋도 가능합니다.

예를 들어 ***MemoryLow=*** 옵션은 이 리소스 배포 모델의 구현입니다.

할당

유한 리소스의 절대 할당입니다. 오버 커밋이 불가능합니다. **Linux**의 이 리소스 유형의 예로는 실시간 예산이 있습니다.

유닛 파일 옵션

리소스 제어 구성 설정입니다.

예를 들어 *CPUAccounting=*, *CPUQuota=* 과 같은 옵션을 사용하여 CPU 리소스를 구성할 수 있습니다. 마찬가지로 *AllowedMemoryNodes=* 및 *IOAccounting=* 과 같은 옵션을 사용하여 메모리 또는 I/O 리소스를 구성할 수 있습니다.

22.3. SYSTEMD를 사용하여 시스템 리소스 할당

절차

서비스의 단위 파일 옵션의 필요한 값을 변경하려면 단위 파일에서 값을 조정하거나 systemctl 명령을 사용하면 됩니다.

1. 선택한 서비스에 할당된 값을 확인합니다.

```
# systemctl show --property <unit file option> <service name>
```

2. CPU 시간 할당 정책 옵션의 필요한 값을 설정합니다.

```
# systemctl set-property <service name> <unit file option>=<value>
```

검증 단계

- 선택한 서비스에 대해 새로 할당된 값을 확인합니다.

```
# systemctl show --property <unit file option> <service name>
```

추가 리소스

- systemd.resource-control(5) 및 systemd.exec(5) 도움말 페이지

22.4. CGROUP의 SYSTEMD 계층 구조 개요

백엔드에서 systemd 시스템 및 서비스 관리자는 슬라이스, 범위 및 서비스 단위를 사용하여 제어 그룹의 프로세스를 구성하고 구성합니다. 사용자 지정 유닛 파일을 만들거나 systemctl 명령을 사용하여 이

계층 구조를 추가로 수정할 수 있습니다. 또한 **systemd** 는 **/sys/fs/cgroup/** 디렉터리에 중요한 커널 리소스 컨트롤러의 계층을 자동으로 마운트합니다.

리소스 제어의 경우 다음 세 가지 **systemd** 장치 유형을 사용할 수 있습니다.

Service

systemd 가 장치 구성 파일에 따라 시작된 프로세스 또는 프로세스 그룹입니다.

서비스는 지정된 프로세스를 하나의 세트로 시작하고 중지할 수 있도록 캡슐화합니다. 서비스의 이름은 다음과 같습니다.

> ***<name>*.service**

범위

외부에서 생성된 프로세스 그룹입니다. 범위는 **fork()** 함수를 통해 임의의 프로세스에서 시작 및 중지된 프로세스를 캡슐화한 다음 런타임 시 **systemd** 에 의해 등록됩니다. 예를 들어 사용자 세션, 컨테이너 및 가상 머신은 범위로 처리됩니다. 범위는 다음과 같이 이름이 지정됩니다.

> ***<name>*.scope**

slice

계층적으로 구성된 단위 그룹입니다. 슬라이스에서는 범위 및 서비스가 배치되는 계층 구조를 구성합니다.

실제 프로세스는 범위 또는 서비스에 포함됩니다. 슬라이스 유닛의 모든 이름은 계층 구조의 위치에 대한 경로에 해당합니다.

대시(**-**) 문자는 **-.slice** 루트 슬라이스에서 슬라이스에 대한 경로 구성 요소의 구분 기호 역할을 합니다. 다음 예에서 다음을 수행합니다.

> ***<parent-name>*.slice**

parent-name.slice 는 **- .slice** 루트 슬라이스의 하위 라이센스입니다. **parent-name.slice** 는 **parent-name-name2.slice** 라는 자체 하위 라이센스를 가질 수 있습니다.

서비스, 범위, 슬라이스 단위는 제어 그룹 계층 구조의 개체에 직접 매핑됩니다. 이러한 장치가 활성화

되면 장치 이름에서 빌드된 그룹 경로를 제어하기 위해 직접 매핑됩니다.

다음은 제어 그룹 계층 구조의 축약된 예입니다.

```
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
  ...
```

위의 예제에서는 서비스 및 범위에 프로세스가 포함되어 있으며 자체 프로세스가 포함되지 않은 슬라이스에 배치되어 있음을 보여줍니다.

추가 리소스

- Red Hat Enterprise Linux에서 *systemctl을 사용하여 시스템 서비스 관리*

- 커널 리소스 컨트롤러의 정의
- systemd.resource-control(5), systemd.exec(5), cgroups(7), fork(), fork(2) 매뉴얼 페이지
- cgroups 이해

22.5. SYSTEMD 장치 나열

systemd 시스템 및 서비스 관리자를 사용하여 단위를 나열합니다.

절차

- systemctl 유틸리티를 사용하여 시스템의 활성 단위를 모두 나열합니다. 터미널은 다음 예와 유사한 출력을 반환합니다.

```
# 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

컨트롤 그룹 계층 구조의 단위 위치를 반영하는 단위의 이름입니다. 리소스 제어와 관련된 단위는 *슬라이스*, *범위* 및 *서비스* 입니다.

LOAD

단위 구성 파일이 올바르게 로드되었는지 여부를 나타냅니다. 단위 파일을 로드하지 못하는 경우 필드에 로드 되지 않고 상태 *오류가* 포함됩니다. 기타 단위 로드 상태는 ***stub,merged*** 및 ***masked*** 입니다.

활성 상태

SUB 의 일반화인 고급 단위 활성화 상태입니다.

SUB

낮은 수준의 유닛 활성화 상태입니다. 가능한 값의 범위는 단위 유형에 따라 다릅니다.

DESCRIPTION

단위 콘텐츠 및 기능에 대한 설명입니다.

- 활성 및 비활성 유닛을 모두 나열합니다.

```
# systemctl --all
```

- 출력 정보 양을 제한합니다.

```
# systemctl --type service,masked
```

--type 옵션에는 *서비스* 및 *슬라이스* 와 같은 쉼표로 구분된 장치 유형 목록 또는 로드 및 *마스크* 와 같은 단위 로드 상태가 필요합니다.

추가 리소스

- RHEL에서 ***systemctl*을 사용하여 시스템 서비스 관리**
- systemd.resource-control(5), systemd.exec(5) 매뉴얼 페이지

22.6. SYSTEMD CGROUPS 계층 구조 보기

특정**cgroups**에서 실행되는 제어 그룹**(cgroup)** 계층 및 프로세스를 표시합니다.

절차

- **systemd-cgls** 명령을 사용하여 시스템의 전체 **cgroup** 계층 구조를 표시합니다.

```
# 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
  …
```

예제 출력은 *슬라이스* 에 의해 가장 높은 수준의 전체 **cgroup** 계층을 반환합니다.

- **systemd-cgls <*resource_controller*>** 명령을 사용하여 리소스 컨트롤러에서 필터링한 **cgroup** 계층을 표시합니다.

```
# 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
  …
```

예제 출력에는 선택한 컨트롤러와 상호 작용하는 서비스가 나열됩니다.

- **systemctl status <*system_unit*>** 명령을 사용하여 특정 단위 및 **cgroups** 계층 구조의 일부에 대한 자세한 정보를 표시합니다.

```
# 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
```

추가 리소스

- 커널 리소스 컨트롤러의 정의
- **systemd.resource-control(5)** 및 **cgroups(7)** 도움말 페이지

22.7. 프로세스의 CGROUPS 보기

프로세스가 속한 *제어 그룹* (cgroup)을 확인할 수 있습니다. 그런 다음 **cgroup** 을 확인하여 사용하는 컨트롤러 및 컨트롤러별 구성을 찾을 수 있습니다.

절차

1. 프로세스가 속하는 **cgroup** 을 보려면 **# cat proc/<*PID*>/cgroup** 명령을 실행합니다.

```
# cat /proc/2467/cgroup
0::/system.slice/example.service
```

예제 출력은 관심 있는 프로세스와 관련이 있습니다. 이 경우 **PID 2467** 로 식별되는 프로세스입니다. 이 프로세스는 **example.service** 유닛에 속합니다. **systemd** 장치 파일 사양에 정의된 대로 프로세스가 올바른 제어 그룹에 배치되었는지 여부를 확인할 수 있습니다.

2. **cgroup**에서 사용하는 컨트롤러 및 해당 구성 파일을 표시하려면 **cgroup** 디렉터리를 확인합니다.

```
# cat /sys/fs/cgroup/system.slice/example.service/cgroup.controllers
memory pids

# ls /sys/fs/cgroup/system.slice/example.service/
cgroup.controllers
cgroup.events
...
cpu.pressure
cpu.stat
io.pressure
memory.current
memory.events
...
pids.current
pids.events
pids.max
```

참고

cgroup 의 버전 **1** 계층 구조는 컨트롤러 단위 모델을 사용합니다. 따라서 **/proc/*PID*/cgroup** 파일의 출력에는 **PID**가 속하는 각 컨트롤러 아래의 **cgroup** 이 표시됩니다. 각 **cgroup**은 **/sys/fs/ cgroup / <*controller_name*> /** 에서 확인할 수 있습니다.

추가 리소스

- **cgroups(7)** 매뉴얼 페이지
- 커널 리소스 컨트롤러의 정의
- **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/admin-guide/cgroup-v2.rst** 파일에 있는 문서(**kernel-doc** 패키지 설치 후)

22.8. 리소스 사용 모니터링

현재 실행 중인 제어 그룹(cgroup) 목록과 해당 리소스 소비 목록을 실시간으로 확인합니다.

절차

1. systemd-cgtop 명령을 사용하여 현재 cgroups 의 동적 계정을 표시합니다.

```
# 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        -        -
...
```

예제 출력에는 현재 리소스 사용량(CPU, 메모리, 디스크 I/O 로드)에 의해 주문되는 cgroup 이 실행 중으로 표시됩니다. 이 목록은 기본적으로 1초마다 새로 고쳐집니다. 따라서 각 제어 그룹의 실제 리소스 사용에 대한 동적 인 통찰력을 제공합니다.

추가 리소스

- systemd-cgtop(1) 매뉴얼 페이지

22.9. SYSTEMD 장치 파일을 사용하여 애플리케이션 제한 설정

systemd 서비스 관리자는 기존 또는 실행 중인 각 유닛을 감독하고 이를 위한 제어 그룹을 생성합니다. 단위에는 /usr/lib/systemd/system/ 디렉터리에 구성 파일이 있습니다.

단위 파일을 다음과 같이 수동으로 수정할 수 있습니다.

- 제한 설정.
- 우선순위를 지정합니다.
- 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제어합니다.

사전 요구 사항

- root 권한이 있습니다.

절차

1. /usr/lib/systemd/system/example.service 파일을 편집하여 서비스의 메모리 사용량을 제한합니다.

```
...
[Service]
MemoryMax=1500K
...
```

구성은 제어 그룹의 프로세스가 초과할 수 없는 최대 메모리를 제한합니다. example.service 서비스는 제한 사항이 적용된 이러한 제어 그룹의 일부입니다. K, M, G 또는 T 접미사를 사용하여 Kilo바이트, 메가바이트, 기가바이트 또는 Terabyte를 측정 단위로 확인할 수 있습니다.

2. 모든 장치 구성 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

3. 서비스를 다시 시작하십시오.

```
# systemctl restart example.service
```

검증

1. 변경 사항이 적용되는지 확인합니다.

```
# cat /sys/fs/cgroup/system.slice/example.service/memory.max
1536000
```

예제 출력에서는 메모리 사용량이 약 1,500KB로 제한되었음을 보여줍니다.

추가 리소스

- cgroups 이해
- Red Hat Enterprise Linux에서 *systemctl을 사용하여 시스템 서비스 관리*
- systemd.resource-control(5), systemd.exec(5) 및 cgroups(7) 도움말 페이지

22.10. SYSTEMCTL 명령을 사용하여 애플리케이션에 제한 설정

CPU 선호도 설정을 사용하면 특정 프로세스의 액세스를 일부 CPU로 제한할 수 있습니다. 효과적으로 CPU 스케줄러는 프로세스의 선호도 마스크에 없는 CPU에서 실행되도록 프로세스를 예약하지 않습니다.

기본 CPU 선호도 마스크는 systemd 에서 관리하는 모든 서비스에 적용됩니다.

특정 systemd 서비스에 대한 CPU 선호도 마스크를 구성하기 위해 systemd 는 다음과 같이 CPUAffinity= 를 제공합니다.

- 단위 파일 옵션.
- /etc/systemd/system.conf 파일의 [Manager] 섹션에 있는 구성 옵션입니다.

CPUAffinity= 단위 파일 옵션은 유사성 마스크로 병합되어 사용되는 CPU 또는 CPU 범위 목록을 설정합니다.

절차

CPUAffinity 장치 파일 옵션을 사용하여 특정 systemd 서비스의 CPU 선호도 마스크를 설정하려면 다음을 수행합니다.

1. 선택한 서비스에서 **CPUAffinity** 유닛 파일 옵션 값을 확인합니다.

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. **root** 사용자로 선호도 마스크로 사용되는 **CPU** 범위에 대한 **CPUAffinity** 장치 파일 옵션의 필요한 값을 설정합니다.

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. 서비스를 다시 시작하여 변경 사항을 적용합니다.

```
# systemctl restart <service name>
```

추가 리소스

- **systemd.resource-control(5), systemd.exec(5), cgroups(7)** 도움말 페이지

22.11. 관리자 구성을 통해 글로벌 기본 CPU 선호도 설정

/etc/systemd/system.conf 파일의 **CPUAffinity** 옵션은 **PID(**프로세스 식별 번호**) 1**에 대한 선호도 마스크와 **PID1**에서 분기된 모든 프로세스를 정의합니다. 그런 다음 서비스별로 **CPUAffinity** 를 덮어쓸 수 있습니다.

/etc/systemd/system.conf 파일을 사용하여 모든 **systemd** 서비스의 기본 **CPU** 선호도 마스크를 설정하려면 다음을 수행합니다.

1. **/etc/systemd/system.conf** 파일의 **[Manager]** 섹션에서 **CPUAffinity=** 옵션의 **CPU** 번호를 설정합니다.

2. 편집한 파일을 저장하고 **systemd** 서비스를 다시 로드합니다.

```
# systemctl daemon-reload
```

3. 서버를 재부팅하여 변경 사항을 적용합니다.

추가 리소스

- systemd.resource-control(5) 및 systemd.exec(5) 도움말 페이지.

22.12. SYSTEMD를 사용하여 NUMA 정책 구성

NUMA(Non-Uniform Memory Access)는 메모리 액세스 시간이 프로세서와 관련된 실제 메모리 위치에 따라 달라지는 컴퓨터 메모리 하위 시스템 설계입니다.

CPU에 가까운 메모리는 다른 CPU(정규 메모리)에 대해 로컬 메모리보다 대기 시간(로컬 메모리)이 낮거나 CPU 세트 간에 공유됩니다.

Linux 커널 측면에서 NUMA 정책은 커널이 프로세스에 물리적 메모리 페이지를 할당하는 위치(예: NUMA 노드)를 관리합니다.

systemd 는 유닛 파일 옵션 NUMAPolicy 및 NUMAMask 를 제공하여 서비스의 메모리 할당 정책을 제어합니다.

절차

NUMAPolicy 단위 파일 옵션을 통해 NUMA 메모리 정책을 설정하려면 다음을 수행합니다.

1. 선택한 서비스에서 NUMAPolicy 장치 파일 옵션 값을 확인합니다.

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. root로 NUMAPolicy 장치 파일 옵션의 필요한 정책 유형을 설정합니다.

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. 서비스를 다시 시작하여 변경 사항을 적용합니다.

```
# systemctl restart <service name>
```

[Manager] 구성 옵션을 사용하여 글로벌 NUMAPolicy 설정을 설정하려면 다음을 수행합니다.

1. /etc/systemd/system.conf 파일에서 파일의 [Manager] 섹션에서 NUMAPolicy 옵션을 검색합니다.

2. 정책 유형을 편집하고 파일을 저장합니다.

3. systemd 구성을 다시 로드합니다.

```
# systemd daemon-reload
```

4. 서버를 재부팅합니다.

중요

엄격한 NUMA 정책을 구성할 때(예: 바인딩) CPUAffinity= 장치 파일 옵션을 적절하게 설정해야 합니다.

추가 리소스

- systemctl 명령을 사용하여 애플리케이션에 제한 설정
- systemd.resource-control(5), systemd.exec(5) 및 set_mempolicy(2) 도움말 페이지.

22.13. SYSTEMD에 대한 NUMA 정책 구성 옵션

systemd 는 NUMA 정책을 구성하는 다음 옵션을 제공합니다.

NUMAPolicy

실행된 프로세스의 NUMA 메모리 정책을 제어합니다. 이러한 정책 유형을 사용할 수 있습니다.

- default
- 기본 설정

- **bind**
- **interleave**
- **지역**

NUMAMask

선택한 NUMA 정책과 연결된 NUMA 노드 목록을 제어합니다.

다음 정책에 대해 NUMAMask 옵션을 지정할 필요가 없습니다.

- **default**
- **지역**

기본 정책의 경우 목록은 단일 NUMA 노드만 지정합니다.

추가 리소스

- **systemd.resource-control(5), systemd.exec(5), set_mempolicy(2) 도움말 페이지**

22.14. SYSTEMD-RUN 명령을 사용하여 일시적인 CGROUP 생성

일시적인 cgroups 는 런타임 중 단위(서비스 또는 범위)에서 사용하는 리소스를 제한합니다.

절차

- **임시 제어 그룹을 생성하려면 다음 형식으로 systemd-run 명령을 사용합니다.**

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

이 명령은 일시적인 서비스 또는 범위 단위를 생성하고 시작하고 이러한 단위로 사용자 지정 명령을 실행합니다.

- **--unit=<name>** 옵션은 유닛의 이름을 제공합니다. **--unit** 을 지정하지 않으면 이름이 자동으로 생성됩니다.

- **--slice=<*name*>.slice** 옵션을 사용하면 서비스 또는 범위 단위를 지정된 슬라이스의 멤버로 만듭니다. **<*name*>.slice** 를 기존 슬라이스의 이름으로 바꾸거나(**systemctl -t** 슬라이스의 출력에 표시됨) 고유한 이름을 전달하여 새 슬라이스를 생성합니다. 기본적으로 서비스 및 범위는 **system.slice** 의 멤버로 생성됩니다.

- **& lt;*command* >**를 서비스 또는 범위 단위에 입력하려는 명령으로 바꿉니다.

다음 메시지가 표시되어 서비스 또는 범위를 성공적으로 생성하고 시작했는지 확인합니다.

```
# Running as unit <name>.service
```

- *선택 사항*: 런타임 정보를 수집하기 위해 프로세스가 완료된 후에도 장치를 계속 실행합니다.

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

명령은 일시적인 서비스 장치를 생성 및 시작하고 단위에서 사용자 지정 명령을 실행합니다. **--remain-after-exit** 옵션을 사용하면 프로세스가 완료된 후에도 서비스가 계속 실행됩니다.

추가 리소스

- 제어 그룹 정의

- RHEL에서 *systemd 관리*

- **systemd-run(1)** 매뉴얼 페이지

22.15. 임시 제어 그룹 제거

더 이상 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제한, 우선 순위 또는 제어할 필요가 없는

경우 **systemd** 시스템 및 서비스 관리자를 사용하여 일시적인 제어 그룹(**cgroups**)을 제거할 수 있습니다.

임시 **cgroup** 은 서비스 또는 범위 단위에 포함된 모든 프로세스가 완료되면 자동으로 릴리스됩니다.

절차

- 모든 프로세스가 포함된 서비스 장치를 중지하려면 다음을 입력합니다.

```
# systemctl stop <name>.service
```

- 하나 이상의 단위 프로세스를 종료하려면 다음을 입력합니다.

```
# systemctl kill <name>.service --kill-who=PID,... --signal=<signal>
```

명령은 **--kill-who** 옵션을 사용하여 종료하려는 제어 그룹에서 프로세스를 선택합니다. 동시에 여러 프로세스를 종료하려면 쉼표로 구분된 **PID** 목록을 전달합니다. **--signal** 옵션은 지정된 프로세스로 전송할 **POSIX** 신호의 유형을 결정합니다. 기본 신호는 ***SIGTERM*** 입니다.

추가 리소스

- 제어 그룹 정의
- 커널 리소스 컨트롤러의 정의
- **systemd.resource-control(5)** 및 **cgroups(7)** 도움말 페이지
- 제어 그룹 이해
- **RHEL**에서 ***systemd 관리***

23장. 제어 그룹 이해

제어 그룹(cgroup) 커널 기능을 사용하면 애플리케이션의 리소스 사용량을 제어하여 보다 효율적으로 사용할 수 있습니다.

다음 작업에 cgroup 을 사용할 수 있습니다.

- 시스템 리소스 할당에 대한 제한 설정.
- 특정 프로세스에 대한 하드웨어 리소스 할당 우선 순위를 지정합니다.
- 특정 프로세스에서 하드웨어 리소스를 가져오지 못하도록 격리합니다.

23.1. 제어 그룹 소개

제어 그룹 Linux 커널 기능을 사용하여 프로세스를 계층적으로 정렬된 그룹인 cgroup 으로 구성할 수 있습니다. /sys/fs/cgroup/ 디렉터리에 기본적으로 마운트된 cgroups 가상 파일 시스템에 구조를 제공하여 계층 구조(제어 그룹 트리)를 정의합니다.

systemd 서비스 관리자는 cgroup 을 사용하여 관리하는 모든 장치 및 서비스를 구성합니다. 수동으로 /sys/fs/cgroup/ 디렉터리에 하위 디렉터리를 생성하고 제거하여 cgroup 의 계층 구조를 관리할 수 있습니다.

그런 다음 커널의 리소스 컨트롤러는 해당 프로세스의 시스템 리소스를 제한, 우선 지정 또는 할당하여 cgroup 의 프로세스 동작을 수정합니다. 이러한 리소스에는 다음이 포함됩니다.

- CPU 시간
- 메모리
- 네트워크 대역폭

- 이러한 리소스의 조합

cgroup 의 주요 사용 사례는 시스템 프로세스를 집계하고 애플리케이션 및 사용자 간에 하드웨어 리소스를 분할하는 것입니다. 이를 통해 환경의 효율성, 안정성 및 보안을 강화할 수 있습니다.

컨트롤 그룹 버전 1

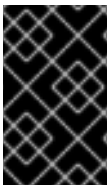
제어 그룹 버전 1 (**cgroups-v1**)은 리소스별 컨트롤러 계층을 제공합니다. 즉, 각 리소스(예: **CPU**, 메모리 또는 **I/O**)에는 자체 제어 그룹 계층 구조가 있습니다. 하나의 컨트롤러에서 각각의 리소스를 관리할 때 다른 컨트롤러와 조정할 수 있는 방식으로 서로 다른 제어 그룹 계층 구조를 결합할 수 있습니다. 그러나 두 컨트롤러가 서로 다른 프로세스 계층에 속하는 경우 적절한 조정이 제한됩니다.

cgroup-v1 컨트롤러는 대규모 기간 동안 개발되었으며 이로 인해 제어 파일의 동작 및 이름 지정이 일정하지 않습니다.

제어 그룹 버전 2

제어 그룹 버전 2 (**cgroups-v2**)는 모든 리소스 컨트롤러가 마운트된 단일 제어 그룹 계층 구조를 제공합니다.

제어 파일 동작 및 이름 지정은 서로 다른 컨트롤러에서 일관되게 유지됩니다.



중요

RHEL 9는 기본적으로 **cgroups-v2** 를 마운트 및 사용합니다.

추가 리소스

- 커널 리소스 컨트롤러 소개
- **cgroups(7)** 매뉴얼 페이지
- **cgroups-v1**
- **cgroups-v2**

23.2. 커널 리소스 컨트롤러 소개

커널 리소스 컨트롤러를 사용하면 제어 그룹의 기능을 사용할 수 있습니다. RHEL 9는 *제어 그룹 버전 1* (cgroups-v1) 및 *제어 그룹 버전 2* (cgroups-v2)에 대해 다양한 컨트롤러를 지원합니다.

제어 그룹 하위 시스템이라고도 하는 리소스 컨트롤러는 CPU 시간, 메모리, 네트워크 대역폭 또는 디스크 I/O와 같은 단일 리소스를 나타내는 커널 하위 시스템입니다. Linux 커널은 systemd 서비스 관리자가 자동으로 마운트하는 다양한 리소스 컨트롤러를 제공합니다. 현재 마운트된 리소스 컨트롤러 목록은 /proc/cgroups 파일에서 찾을 수 있습니다.

cgroups-v1 에서 사용 가능한 컨트롤러:

blkio

블록 장치에 대한 입력/출력 액세스 제한을 설정합니다.

cpu

제어 그룹의 작업에 대한 CFS(Completely Fair Scheduler)의 매개변수를 조정합니다. cpu 컨트롤러는 동일한 마운트에 cpuacct 컨트롤러와 함께 마운트됩니다.

cpuacct

제어 그룹의 작업에서 사용하는 CPU 리소스에 대한 자동 보고서를 생성합니다. cpuacct 컨트롤러는 동일한 마운트에 cpu 컨트롤러와 함께 마운트됩니다.

cpuset

CPU의 지정된 하위 집합에서만 실행되도록 제어 그룹 작업을 제한하고 지정된 메모리 노드에서만 메모리를 사용하도록 작업에 지시합니다.

devices

제어 그룹의 작업에 대한 장치에 대한 액세스를 제어합니다.

freezer

제어 그룹에서 작업을 일시 중지하거나 재개합니다.

memory

제어 그룹의 작업에서 메모리 사용량에 대한 제한을 설정하고 해당 작업에서 사용하는 메모리 리소스에 대한 자동 보고서를 생성합니다.

net_cls

Linux 트래픽 컨트롤러(tc 명령)를 활성화하여 특정 제어 그룹 작업에서 시작된 패킷을 식별하는 클래스 식별자(classid)가 있는 네트워크 패킷을 태그합니다. net_cls 의 하위 시스템인 net_filter (iptables)도 이 태그를 사용하여 이러한 패킷에 대한 작업을 수행할 수 있습니다. net_filter 는 Linux 방화벽에서 특정 제어 그룹 작업에서 시작된 패킷을 식별할 수 있는 방화벽 식별자(fwid)로 네트워크 소켓을 태그합니다(iptables 명령을 사용하여).

net_prio

네트워크 트래픽의 우선 순위를 설정합니다.

pids

제어 그룹에서 여러 프로세스 및 해당 하위 항목에 대한 제한을 설정합니다.

perf_event

perf 성능 모니터링 및 보고 유틸리티를 통한 모니터링을 위한 작업을 그룹화합니다.

rdma

제어 그룹의 Remote Direct Memory Access/InfiniBand 특정 리소스에 대한 제한을 설정합니다.

hugetlb

제어 그룹의 작업으로 큰 크기의 가상 메모리 페이지 사용을 제한하는 데 사용할 수 있습니다.

cgroups-v2 에서 사용 가능한 컨트롤러:

io

블록 장치에 대한 입력/출력 액세스 제한을 설정합니다.

memory

제어 그룹의 작업에서 메모리 사용량에 대한 제한을 설정하고 해당 작업에서 사용하는 메모리 리소스에 대한 자동 보고서를 생성합니다.

pids

제어 그룹에서 여러 프로세스 및 해당 하위 항목에 대한 제한을 설정합니다.

rdma

제어 그룹의 Remote Direct Memory Access/InfiniBand 특정 리소스에 대한 제한을 설정합니다.

cpu

제어 그룹의 작업에 대한 CFS(Completely Fair Scheduler) 매개변수를 조정하고 제어 그룹의 작

업에서 사용하는 **CPU** 리소스에 대한 자동 보고서를 생성합니다.

cpuset

CPU의 지정된 하위 집합에서만 실행되도록 제어 그룹 작업을 제한하고 지정된 메모리 노드에서만 메모리를 사용하도록 작업에 지시합니다. 새 파티션 기능이 있는 코어 기능**(cpus{,.effective}, mems{,.effective})**만 지원합니다.

perf_event

perf 성능 모니터링 및 보고 유틸리티에 의한 모니터링을 위한 그룹 작업입니다. **perf_event** 는 **v2** 계층에서 자동으로 활성화됩니다.

중요

리소스 컨트롤러는 **cgroups-v1** 계층 구조 또는 **cgroups-v2** 계층에서 둘 다 동시에 사용할 수 있습니다.

추가 리소스

- **cgroups(7)** 매뉴얼 페이지
- **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** 디렉터리에 있는 문서**(kernel-doc** 패키지 설치 후**).**

23.3. 네임스페이스 소개

네임스페이스는 소프트웨어 오브젝트를 구성하고 식별하는 가장 중요한 방법 중 하나입니다.

네임스페이스는 글로벌 리소스(예**:** 마운트 지점**,** 네트워크 장치 또는 호스트 이름)를 추상화로 래핑하여 글로벌 리소스의 자체 격리된 인스턴스가 있는 네임스페이스 내의 프로세스에 나타납니다. 네임스페이스를 사용하는 가장 일반적인 기술 중 하나는 컨테이너입니다.

특정 글로벌 리소스에 대한 변경 사항은 해당 네임스페이스의 프로세스에만 표시되고 시스템 또는 기타 네임스페이스의 나머지 부분에는 영향을 미치지 않습니다.

프로세스가 속한 네임스페이스를 검사하려면 **/proc/<*PID*>/ns/** 디렉토리에서 심볼릭 링크를 확인할 수 있습니다.

표 23.1. 격리할 수 있는 지원되는 네임스페이스 및 리소스:

네임스페이스	격리
Mount	마운트 지점
UTS	호스트 이름 및 NIS 도메인 이름
IPC	System V IPC, POSIX 메시지 대기열
PID	프로세스 ID
네트워크	네트워크 장치, 스택, 포트 등
사용자	사용자 및 그룹 ID
제어 그룹	제어 그룹 루트 디렉터리

추가 리소스

- **네임스페이스(7) 및 cgroup_namespaces(7) 매뉴얼 페이지**
- 제어 그룹 소개

24장. CGROUPFS를 사용하여 CGROUP을 수동으로 관리

cgroup fs 가상 파일 시스템에 디렉터리를 생성하여 시스템의 **cgroup** 계층을 관리할 수 있습니다. 파일 시스템은 기본적으로 **/sys/fs/cgroup/** 디렉토리에 마운트되며 전용 제어 파일에 원하는 구성을 지정할 수 있습니다.

중요

일반적으로 **Red Hat**은 시스템 리소스 사용량을 제어하기 위해 **systemd** 를 사용할 것을 권장합니다. 특수한 경우에만 **cgroup** 가상 파일 시스템을 수동으로 구성해야 합니다. 예를 들어 **cgroup-v2** 계층 구조에 동등한 권한이 없는 **cgroup-v1** 컨트롤러를 사용해야 하는 경우입니다.

24.1. CGROUP 생성 및 CGROUP-V2 파일 시스템에서 컨트롤러 활성화

디렉토리를 만들거나 제거하고**cgroup**가상 파일 시스템의 파일에 작성하여 *제어 그룹* (**cgroups**)을 관리할 수 있습니다. 파일 시스템은 기본적으로 **/sys/fs/cgroup/** 디렉토리에 마운트됩니다. **cgroup** 컨트롤러의 설정을 사용하려면 하위 **cgroup** 에 대해 원하는 컨트롤러도 활성화해야 합니다. 루트 **cgroup** 에는 기본적으로 하위 **cgroup** 에 대한 메모리 및 **pids** 컨트롤러가 활성화되었습니다. 따라서 **Red Hat**은 **/sys/fs/cgroup/** **root** **cgroup** 내에 두 개 이상의 하위 **cgroup**을 생성할 것을 권장합니다. 이렇게 하면 하위 **cgroup**에서 메모리 및 **pids** 컨트롤러를 선택적으로 제거하고 **cgroup** 파일의 조직적인 명확성을 유지할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. **/sys/fs/cgroup/Example/** 디렉토리를 생성합니다.

```
# mkdir /sys/fs/cgroup/Example/
```

/sys/fs/cgroup/Example/ 디렉토리는 하위 그룹을 정의합니다. **/sys/fs/cgroup/Example/** 디렉토리를 생성하면 일부 **cgroups-v2** 인터페이스 파일이 디렉터리에 자동으로 생성됩니다. **/sys/fs/cgroup/Example/** 디렉터리에는 메모리 및 **pids** 컨트롤러에 대한 컨트롤러별 파일도 포함되어 있습니다.

2. 선택적으로 새로 생성된 하위 제어 그룹을 검사합니다.

```
# 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.procs
…
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
-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
```

예제 출력에는 **cgroup.procs** 또는 **cgroup.controllers** 와 같은 일반 **cgroup** 제어 인터페이스 파일이 표시됩니다. 이러한 파일은 활성화된 컨트롤러에 관계없이 모든 제어 그룹에 공통적입니다.

memory.high 및 **pids.max** 와 같은 파일은 루트 제어 그룹(**/sys/fs/cgroup/**)에 있는 메모리 및 **pids** 컨트롤러와 관련되어 있으며 **systemd** 에서 기본적으로 활성화됩니다.

기본적으로 새로 생성된 하위 그룹은 상위 **cgroup** 의 모든 설정을 상속합니다. 이 경우 루트 **cgroup** 에서 제한이 없습니다.

3. 원하는 컨트롤러를 **/sys/fs/cgroup/cgroup.controllers** 파일에서 사용할 수 있는지 확인합니다.

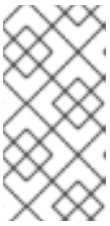
```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

4. 원하는 컨트롤러를 활성화합니다. 이 예제에서는 **cpu** 및 **cpuset** 컨트롤러입니다.

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

이 명령을 사용하면 **/sys/fs/cgroup/** **root** 제어 그룹의 즉시 하위 그룹에 **cpu** 및 **cpuset** 컨트롤러를 활성화합니다. 새로 생성된 예제 제어 그룹을 포함합니다. *하위 그룹은* 프로세스를 지정하고 기준에 따라 각 프로세스에 제어 검사를 적용할 수 있는 위치입니다.

사용자는 모든 수준에서 **cgroup.subtree_control** 파일의 내용을 읽고 즉시 하위 그룹에서 활성화에 사용할 수 있는 컨트롤러를 파악할 수 있습니다.



참고

기본적으로 **root** 제어 그룹의 **/sys/fs/cgroup/cgroup.subtree_control** 파일에는 메모리 및 **Pids** 컨트롤러 가 포함되어 있습니다.

5. **Example** 제어 그룹의 하위 **cgroups** 에 대해 원하는 컨트롤러를 활성화합니다.

```
# echo "+cpu +cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

이 명령을 사용하면 즉시 하위 제어 그룹이 **CPU** 시간 분배를 규제하는 데 관련된 컨트롤러 *만* 가질 수 있습니다. 메모리 또는 **pids** 컨트롤러와 다릅니다.

6. **/sys/fs/cgroup/Example/tasks/** 디렉터리를 생성합니다.

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

/sys/fs/cgroup/Example/tasks/ 디렉터리는 **cpu** 및 **cpu set** 컨트롤러에만 관련된 파일이 있는 하위 그룹을 정의합니다. 이제 프로세스를 이 제어 그룹에 할당하고 프로세스에 **cpu** 및 **cpuset** 컨트롤러 옵션을 활용할 수 있습니다.

7. 필요한 경우 하위 제어 그룹을 검사합니다.

```
# 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
```

중요

cpu 컨트롤러는 관련 하위 제어 그룹에 단일 **CPU**에서 시간을 위해 경쟁하는 최소 **2**개의 프로세스가 있는 경우에만 활성화됩니다.

검증 단계

- 선택 사항: 원하는 컨트롤러만 활성 상태로 새 **cgroup** 을 생성했는지 확인합니다.

```
# cat /sys/fs/cgroup/Example/tasks/cgroup.controllers
cpuset cpu
```

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러의 정의
- **cgroup-v1** 마운트
- **cgroups(7), sysfs(5)** 매뉴얼 페이지

24.2. CPU 가중치를 조정하여 애플리케이션의 CPU 시간 분배 제어

CPU 시간을 특정 **cgroup** 트리의 애플리케이션에 분산하기 위해 **CPU** 컨트롤러 의 관련 파일에 값을 할당해야 합니다.

사전 요구 사항

- 루트 권한이 있습니다.

- CPU 시간 배포를 제어하려는 애플리케이션이 있습니다.

- 다음 예와 같이 /sys/fs/cgroup/ *root 제어 그룹 내에 하위 제어 그룹의* 두 가지 수준 계층 구조를 생성했습니다.

```
...
├── Example
│   ├── g1
│   ├── g2
│   └── g3
...
```

- 상위 제어 그룹 및 하위 제어 그룹에서 cpu 컨트롤러를 활성화하여 cgroup 생성 및 cgroup-v2 파일 시스템의 컨트롤러 활성화에 설명된 것과 유사하게 활성화됩니다.

절차

1. 제어 그룹 내에서 리소스 제한을 달성하도록 원하는 CPU 가중치를 구성합니다.

```
# 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. 애플리케이션의 PID를 g1, g 2 및 g 3 하위 그룹에 추가합니다.

```
# 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
```

예제 명령을 사용하면 원하는 애플리케이션이 예제/g*/ 하위 cgroups의 멤버가 되고 해당 cgroups의 구성에 따라 CPU 시간이 배포됩니다.

프로세스가 실행 중인 하위 cgroup(g1,g2, g3)의 가중치는 상위 cgroup(Example) 수준에서 요약됩니다. 그런 다음 CPU 리소스는 해당 가중치에 비례하여 배포됩니다.

결과적으로 모든 프로세스가 동시에 실행되면 커널은 해당 cgroup의 cpu.weight 파일에 따라 각 CPU 시간을 비례합니다.

하위 cgroup	**cpu.weight** file	CPU 시간 할당
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

cpu.weight 컨트롤러 파일의 값은 백분율이 아닙니다.

한 프로세스가 중단되어 **cgroup g2** 에 실행 중인 프로세스가 없으면 계산에서는 **cgroup g2** 가 생략되고 **cgroup g 1** 및 **g 3** 의 가중치만 줄어듭니다:

하위 cgroup	**cpu.weight** file	CPU 시간 할당
g1	150	~75% (150/200)
g3	50	~25% (50/200)

중요

하위 **cgroup**에 여러 개의 실행 중인 프로세스가 있는 경우 해당 **cgroup**에 할당된 **CPU** 시간이 해당 **cgroup**의 멤버 프로세스에 동일하게 배포됩니다.

검증

1. 애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

명령 출력에는 **Example/g*/** 하위 **cgroup**에서 실행되는 지정된 애플리케이션의 프로세스가 표시됩니다.

2. 제한된 애플리케이션의 현재 **CPU** 사용량을 검사합니다.

```
# 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
...
```

참고

명확한 설명을 위해 모든 예제 프로세스를 단일 CPU에서 실행하도록 했습니다. CPU 가중치는 여러 CPU에서 사용되는 경우에도 동일한 원칙을 적용합니다.

PID 33373, PID 33374 및 PID 33 377 의 CPU 리소스가 해당 하위 cgroup에 할당된 가중치 150, 100, 50에 따라 할당되었습니다. 가중치는 각 애플리케이션의 CPU 시간을 약 50%, 33%, 16%에 해당합니다.

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러의 정의
- cgroup 생성 및 cgroup-v2 파일 시스템에서 컨트롤러 활성화
- 리소스 배포 모델
- cgroups(7), sysfs(5) 매뉴얼 페이지

24.3. CGROUP-V1 마운트

부팅 프로세스 중에 **RHEL 9**는 기본적으로 **cgroup-v2** 가상 파일 시스템을 마운트합니다. 애플리케이션에 대한 리소스를 제한하는 **cgroup-v1** 기능을 활용하려면 시스템을 수동으로 구성합니다.

참고

cgroup-v1 및 **cgroup-v2** 모두 커널에서 완전히 활성화됩니다. 커널 관점에서는 기본 제어 그룹 버전이 없으며 시작 시 **systemd** 에서 마운트하도록 결정합니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. **systemd** 시스템 및 서비스 관리자가 시스템을 부팅하는 동안 기본적으로 **cgroup-v1** 을 마운트하도록 시스템을 구성합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

그러면 현재 부팅 항목에 필요한 커널 명령줄 매개 변수가 추가됩니다.

모든 커널 부팅 항목에 동일한 매개변수를 추가하려면 다음을 수행합니다.

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

2. 변경 사항을 적용하려면 시스템을 재부팅합니다.

검증

1. 선택적으로 **cgroup-v1** 파일 시스템이 마운트되었는지 확인합니다.

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,seclabel,size=4096k,nr_inodes=1024,mode=755,inode64)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/syste
```

```
md-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,memory)
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,blkio)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,devices)
cgroup on /sys/fs/cgroup/misc type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,misc)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/rdma type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,rdma)
```

다양한 **cgroup-v1** 컨트롤러에 해당하는 **cgroup-v1** 파일 시스템이 **/sys/fs/cgroup/** 디렉터리에 성공적으로 마운트되었습니다.

2. 선택적으로 **/sys/fs/cgroup/** 디렉터리의 콘텐츠를 검사합니다.

```
# ll /sys/fs/cgroup/
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 blkio
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpu → cpu,cpuacct
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpuacct → cpu,cpuacct
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 cpu,cpuacct
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 cpuset
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 devices
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 freezer
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 hugetlb
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 memory
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 misc
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_cls → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 net_cls,net_prio
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_prio → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 perf_event
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 pids
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 rdma
dr-xr-xr-x. 11 root root  0 Mar 16 09:34 systemd
```

기본적으로 *루트 제어 그룹* 이라고 하는 **/sys/fs/cgroup/** 디렉터리에는 **cpuset** 와 같은 컨트

롤러별 디렉터리가 포함되어 있습니다. 또한 **systemd** 와 관련된 일부 디렉터리가 있습니다.

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러의 정의
- **cgroups(7), sysfs(5)** 매뉴얼 페이지
- RHEL 9에서 기본적으로 cgroup-v2 활성화

24.4. CGROUPS-V1을 사용하여 애플리케이션에 CPU 제한 설정

제어 그룹 버전 1 (cgroups-v1)을 사용하여 애플리케이션에 대한 CPU 제한을 구성하려면 **/sys/fs/** 가상 파일 시스템을 사용합니다.

사전 요구 사항

- 루트 권한이 있습니다.
- 제한하려는 CPU 사용량이 있는 애플리케이션이 있습니다.
- **systemd** 시스템 및 서비스 관리자가 시스템 부팅 중에 **cgroups-v1** 을 기본적으로 마운트하도록 시스템을 구성했습니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

그러면 현재 부팅 항목에 필요한 커널 명령줄 매개 변수가 추가됩니다.

절차

1.

CPU 소비에서 제한하려는 애플리케이션의 PID(프로세스 ID)를 식별합니다.

```
# 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
...
```

top 프로그램의 이 예제 출력에서는 PID 6955 가 있는 애플리케이션 sha1sum 이 많은 CPU 리소스를 사용한다는 것을 보여줍니다.

2.

cpu 리소스 컨트롤러 디렉터리에 하위 디렉터리를 생성합니다.

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

이 디렉터리는 특정 프로세스를 배치하고 프로세스에 특정 CPU 제한을 적용할 수 있는 제어 그룹을 나타냅니다. 동시에 여러 cgroups-v1 인터페이스 파일과 cpu 컨트롤러별 파일이 디렉터리에 생성됩니다.

3.

***선택 사항*: 새로 생성된 제어 그룹을 검사합니다.**

```
# 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
```

이 예제 출력은 예제 제어 그룹의 프로세스에 대해 설정할 수 있는 특정 구성 및/또는 제한을 나타내는 **cpuacct.usage,cpu.cfs._period_us** 와 같은 파일을 보여줍니다. 각 파일 이름 앞에는 자신이 속한 제어 그룹 컨트롤러의 이름이 접두어 있습니다.

기본적으로 새로 생성된 제어 그룹은 제한 없이 시스템의 전체 **CPU** 리소스에 대한 액세스를 상속합니다.

4. 제어 그룹에 대한 **CPU** 제한을 구성합니다.

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

- **cpu.cfs_period_us** 파일은 **CPU** 리소스에 대한 제어 그룹의 액세스를 다시 할당해야 하는 빈도에 대해 **microseconds(here**에서 **"us")**의 기간을 나타냅니다. 상한은 **1 000** 마이크로초이며 더 낮은 제한은 **1 000** 마이크로초입니다.

- **cpu.cfs_quota_us** 파일은 모든 프로세스가 하나의 기간**(cpu.cfs_period_us**에 정의된 대로) 제어 그룹에 전체적으로 실행될 수 있는 총 시간을 나타냅니다. 단일 기간 동안 제어 그룹의 프로세스가 할당량에 의해 지정된 모든 시간을 사용하는 경우 나머지 기간 동안 제한되며 다음 기간까지 실행되지 않습니다. 낮은 제한은 **1000** 마이크로초입니다.

위의 예제 명령은 **CPU** 시간 제한을 설정하므로 **Example** 제어 그룹의 모든 프로세스가 **0.2**초**(cpu.cfs_quota_us)** 중 **1**초**(cpu.cfs_period_us**로 정의됨)만 실행할 수 있도록 합니다.

5. *선택 사항*: 제한을 확인합니다.

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. 애플리케이션 **PID**를 **Example** 제어 그룹에 추가합니다.

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

이 명령은 특정 애플리케이션이 **Example** 제어 그룹의 멤버가 되도록 하여 **Example** 제어 그룹에 구성된 **CPU** 제한을 초과하지 않습니다. **PID**는 시스템의 기존 프로세스를 나타냅니다. 여기서 **PID 6955** 는 **cpu** 컨트롤러의 사용 사례를 설명하는 데 사용되는 **sha1sum /dev/zero & amp;** 를 처리하도록 할당되었습니다.

검증

1. 애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# 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
```

이 예제 출력은 원하는 애플리케이션의 프로세스가 **Example** 제어 그룹에서 실행되므로 애플리케이션의 프로세스에 **CPU** 제한을 적용합니다.

2. 제한 애플리케이션의 현재 **CPU** 사용량을 확인합니다.

```
# 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
...
```

PID 6955 의 **CPU** 사용량이 **99 %**에서 **20%**로 감소했습니다.

참고

cpu.cfs_period_us 및 **cpu.cfs_quota_us** 에 해당하는 **cgroups-v2** 파일은 **cpu.max** 파일입니다. **cpu.max** 파일은 **cpu** 컨트롤러를 통해 사용할 수 있습니다.

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러의 정의
- **cgroups(7), sysfs(5)** 매뉴얼 페이지

25장. BPF COMPILER COLLECTION을 사용하여 시스템 성능 분석

시스템 관리자는 BCC(BPF Compiler Collection) 라이브러리를 사용하여 Linux 운영 체제의 성능을 분석하기 위한 툴을 생성하고 다른 인터페이스를 통해 가져오기가 어려울 수 있는 정보를 수집할 수 있습니다.

25.1. BCC-TOOLS 패키지 설치

bcc-tools 패키지를 설치합니다. 이 패키지는 BPF Compiler Collection(BCC) 라이브러리를 종속성으로 설치합니다.

절차

1. bcc-tools 를 설치합니다.

```
# dnf install bcc-tools
```

BCC 툴은 /usr/share/bcc/tools/ 디렉토리에 설치됩니다.

2. 선택적으로 툴을 검사합니다.

```
# ll /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
...
```

위 목록의 doc 디렉터리에는 각 툴에 대한 문서가 포함되어 있습니다.

25.2. 성능 분석을 위해 선택된 BCC-TOOLS 사용

BPF Compiler Collection (BCC) 라이브러리에서 미리 생성된 특정 프로그램을 사용하여 이벤트별로 시스템 성능을 효율적이고 안전하게 분석합니다. BCC 라이브러리에서 사전 생성된 프로그램 세트는 추

가 프로그램 생성의 예로 사용될 수 있습니다.

사전 요구 사항

- 설치된 **bcc-tools** 패키지
- 루트 권한

execsnoop를 사용하여 시스템 프로세스 검사

1. 한 터미널에서 **execsnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/execsnoop
```

2. 다른 터미널 실행에서 예를 들면 다음과 같습니다.

```
$ ls /usr/share/bcc/tools/doc/
```

위의 명령은 **ls** 명령의 수명이 짧은 프로세스를 생성합니다.

3. **execsnoop** 을 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
PCOMM PID    PPID   RET ARGS
ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop 프로그램은 시스템 리소스를 사용하는 새 프로세스마다 출력 행을 출력합니다. 또한 **ls** 와 같이 매우 빨리 실행되는 프로그램 프로세스를 감지하며 대부분의 모니터링 툴은 등록하지 않습니다.

execsnoop 출력에 다음 필드가 표시됩니다.

PCOMM

상위 프로세스 이름입니다. **(ls)**

PID

프로세스 ID입니다. (8382)

PPID

상위 프로세스 ID입니다. (8287)

RET

프로그램 코드를 새 프로세스로 로드하는 **exec()** 시스템 호출(0)의 반환 값입니다.

ARGS

시작된 프로그램의 인수와 관련된 위치입니다.

execsnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/execsnoop_example.txt** 파일을 참조하십시오.

exec() 에 대한 자세한 내용은 **exec(3)** 매뉴얼 페이지를 참조하십시오.

opensnoop를 사용하여 명령이 여는 파일 추적

1. 한 터미널에서 **opensnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/opensnoop -n uname
```

위의 명령은 **uname** 명령의 프로세스에 의해서만 열려 있는 파일의 출력을 출력합니다.

2. 다른 터미널에서 다음을 입력합니다.

```
$ uname
```

위의 명령은 다음 단계에서 캡처된 특정 파일을 엽니다.

3. **opensnoop** 을 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
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
...
```

opensnoop 프로그램은 전체 시스템에서 **open()** 시스템 호출을 감시하고 **uname** 이 진행하려고 시도하는 각 파일에 대한 출력 행을 출력합니다.

opensnoop 출력에 다음 필드가 표시됩니다.

PID

프로세스 **ID**입니다. **(8596)**

COMM

프로세스 이름입니다. **(uname)**

FD

파일 설명자 **- open()** 이 열려 있는 파일을 참조하기 위해 반환하는 값입니다. **(3)**

ERR

모든 오류.

PATH

open() 가 열려는 파일의 위치입니다.

명령에서 존재하지 않는 파일을 읽으려고 하면 **FD** 열은 **-1** 을 반환하고 **ERR** 열 은 관련 오류에 해당하는 값을 출력합니다. 따라서 **opensnoop** 은 제대로 작동하지 않는 애플리케이션을 식별하는 데 도움이 될 수 있습니다.

opensnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/opensnoop_example.txt** 파일을 참조하십시오.

open() 에 대한 자세한 내용은 **open(2)** 매뉴얼 페이지를 참조하십시오.

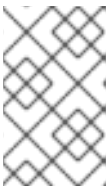
BIOStop를 사용하여 디스크에서 **I/O** 작업 검사

1. 한 터미널에서 **lighttop** 프로그램을 실행합니다.

-

```
# /usr/share/bcc/tools/biotop 30
```

명령을 사용하면 디스크에서 **I/O** 작업을 수행하는 상위 프로세스를 모니터링할 수 있습니다. 인수를 사용하면 명령이 **30**초 요약을 생성합니다.



참고

인수를 제공하지 않으면 기본적으로 출력 화면은 **1**초마다 새로 고칩니다.

2. 다른 터미널에서 다음을 입력합니다. 예를 들면 다음과 같습니다.

```
# dd if=/dev/vda of=/dev/zero
```

위의 명령은 로컬 하드 디스크 장치에서 콘텐츠를 읽고 출력을 **/dev/zero** 파일에 씁니다. 이 단계는 **BIOStop** 을 설명하기 위해 특정 **I/O** 트래픽을 생성합니다.

3. **biotop** 를 실행하는 터미널은 다음과 유사한 출력을 보여줍니다.

```
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
...
```

biotop 출력은 다음 필드를 표시합니다.

PID

프로세스 ID입니다. (9568)

COMM

프로세스 이름입니다. (dd)

DISK

읽기 작업을 수행하는 디스크입니다. (vda)

I/O

수행된 읽기 작업 수입니다. (16294)

kbytes

읽기 작업에서 도달한 K바이트 수입니다. (14,440,636)

AVGms

평균 읽기 작업의 I/O 시간입니다. (3.69)

자세한 내용은 /usr/share/bcc/tools/doc/ biotop _example.txt 파일을 참조하십시오.

dd 에 대한 자세한 내용은 dd(1) 매뉴얼 페이지를 참조하십시오.

xfsslower를 사용하여 예기치 않은 느린 파일 시스템 작업 노출

1. 한 터미널에서 **xfsslower** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/xfsslower 1
```

위의 명령은 **XFS** 파일 시스템이 읽기, 쓰기, 열기 또는 동기화(fsync) 작업을 수행하는 데 사용하는 시간을 측정합니다. **1** 인수는 프로그램이 **1ms**보다 느린 작업만 표시하는지 확인합니다.

참고

인수를 제공하지 않으면 기본적으로 **xfsslower** 는 **10ms**보다 느린 작업을 표시합니다.

2. 예를 들어 다른 터미널에서 다음을 입력합니다.

```
$ vim text
```

위의 명령은 **vim** 편집기에 텍스트 파일을 생성하여 **XFS** 파일 시스템과의 특정 상호 작용을 시작합니다.

3. **xfsslower** 를 실행하는 터미널에는 이전 단계에서 파일을 저장할 때와 비슷한 내용이 표시됩니다.

```
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'
...
```

위의 각 행은 파일 시스템의 작업을 특정 임계값보다 오래 걸리는 작업을 나타냅니다. **xfsslower** 는 가능한 파일 시스템 문제를 노출하면 예기치 않은 느린 작업을 수행할 수 있습니다.

xfsslower 출력에 다음 필드가 표시됩니다.

COMM

프로세스 이름입니다. **(b'bash')**

T

작업 유형입니다. **(R)**

- **Read**
- **write**
- **sync**

OFF_KB

KB의 파일 오프셋입니다. (0)

파일 이름

읽기, 쓰기 또는 동기화되는 파일입니다.

xfsslower 에 대한 자세한 내용, 예제 및 옵션을 보려면 /usr/share/bcc/tools/doc/xfsslower_example.txt 파일을 참조하십시오.

fsync 에 대한 자세한 내용은 fsync(2) 매뉴얼 페이지를 참조하십시오.