

Oracle Linux 9

Debugging the Kernel With Drgn and Corelens



G18176-07
October 2025



Oracle Linux 9 Debugging the Kernel With Drgn and Corelens,

G18176-07

Copyright © 2024, 2025, Oracle and/or its affiliates.

Documentation License

The content in this document is licensed under the [Creative Commons Attribution–Share Alike 4.0](https://creativecommons.org/licenses/by-sa/4.0/) (CC-BY-SA) license. In accordance with CC-BY-SA, if you distribute this content or an adaptation of it, you must provide attribution to Oracle and retain the original copyright notices.

Contents

Preface

1 About the drgn and corelens Kernel Debugging Utilities

2 (Optional) Installing DebugInfo Packages

3 Installing drgn

4 Installing drgn-tools

5 Getting Started With drgn

6 drgn Command Reference

7 Using the drgn Library With Python

8 Getting Started With corelens

9 corelens Command Reference

Selecting Modules for corelens Command Output	1
Generating Reports With corelens	3

Preface

[Oracle Linux 9: Debugging the Kernel With Drgn and Corelens](#) describes how to install and use the `drgn` and `corelens` kernel debugging utilities to analyze crash logs and troubleshoot system problems.

1

About the `drgn` and `corelens` Kernel Debugging Utilities

`Drgn` is a tool and a programming library that can be used to extract debug information from both the live kernel of the running machine and memory crash dumps from halted systems (`vmcore`).

To configure an Oracle Linux 9 system to generate `vmcore` crash dumps, follow the kernel dump instructions in [Oracle Linux 9: Managing Kernels and System Boot](#).

`Drgn` can be used as part of a root cause analysis to provide extra metrics that aren't already exposed through existing dashboards and interfaces.

For more information, see <https://drgn.readthedocs.io/>.

`Corelens` is provided through a separate `drgn-tools` package and provides the same functionality as `Drgn`, but requires no prior knowledge of the kernel implementation, data structures, or Python programming.

For more information, see the `corelens(1)` manual pages.

2

(Optional) Installing DebugInfo Packages

You can optionally install `*-debuginfo` packages to add extra DWARF debugging information in generated core dumps. They're intended for development purposes only, so we recommend that you only install them in development environments.

Before installing `*-debuginfo` packages, enable the Oracle Linux 9 `debuginfo` repository by creating the `/etc/yum.repos.d/debuginfo.repo` file with root privileges and the following contents:

```
[debuginfo]
name=Oracle Linux 9 Debuginfo Packages
baseurl=https://oss.oracle.com/ol9/debuginfo/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
gpgcheck=1
enabled=1
```

```
sudo dnf update -y
```

For more information, see [Oracle Linux: Managing Software on Oracle Linux](#).

1. If you're running Oracle Linux with the Unbreakable Enterprise Kernel (UEK), install the `kernel-uek-debuginfo` package by using the `dnf` command:

```
sudo dnf install -y kernel-uek-debuginfo-$(uname -r)
```

If you're running Oracle Linux with the Red Hat Compatible Kernel (RHCK), install the `kernel-debuginfo` package instead:

```
sudo dnf install -y kernel-debuginfo-$(uname -r)
```

Run the install command each time the kernel is updated through the package manager. The DebugInfo package is only functional when it matches the running kernel, and it's not replaced automatically when a newer kernel version is installed on the system.

2. Use the package manager to search for DebugInfo packages to install:

```
dnf search *-debuginfo
```

The selected `*-debuginfo` packages are installed.

3

Installing `drgn`

Install the `drgn` package on Oracle Linux 9.

Before installing the `drgn` package, enable the `ol9_addons` repository:

```
sudo dnf config-manager --enable ol9_addons
```

```
sudo dnf update -y
```

For more information, see [Oracle Linux: Managing Software on Oracle Linux](#).

1. (Optional) Install relevant DebugInfo packages on the system.

For more information, see [\(Optional\) Installing DebugInfo Packages](#).

2. Install the `drgn` package:

```
sudo dnf install -y drgn
```

The `drgn` package is installed.

4

Installing `drgn-tools`

Install the `drgn-tools` package on Oracle Linux 9.

Before installing the `drgn-tools` package, enable the `ol9_addons` repository:

```
sudo dnf config-manager --enable ol9_addons
```

```
sudo dnf update -y
```

For more information, see [Oracle Linux: Managing Software on Oracle Linux](#).

1. (Optional) Install relevant DebugInfo packages on the system.

For more information, see [\(Optional\) Installing DebugInfo Packages](#).

2. Install the `drgn-tools` package:

```
sudo dnf install -y drgn-tools
```

The `drgn-tools` package is installed.

5

Getting Started With `drgn`

Debugging a live running kernel by using the `drgn` command.

Install the `drgn` package. For more information, see [Installing `drgn`](#).

The `drgn` command can be used to troubleshoot system problems by analyzing the contents of system images and crash dumps.

1. To debug the running kernel, use the `drgn` command to analyze the contents of the `/proc/kcore` live system image:

```
drgn
```

2. Type `exit()` or press the `Ctrl + D` keys to exit the Python shell.
3. For more information about how to use the `drgn` command, use the `-h` option:

```
drgn -h
```

A Python shell was started, provided access to live kernel debugging information, and was then stopped.

If a matching `DebugInfo` package isn't installed for the running kernel, you might see an error message. `Drgn` can still be used on a production system to debug a kernel dump that was generated by a separate development system that has `DebugInfo` packages installed.

Note

Kernel dumps generated on systems without `DebugInfo` packages installed can be debugged by using the `corelens` command, as that uses Common Type Format (CTF) debug information when DWARF debug information isn't present. For more information, see [Getting Started With `corelens`](#).

6

drgn Command Reference

This table provides information about the `drgn` command.

Action	Command	Description
Start a Python shell to analyze the contents of the <code>/proc/kcore</code> live system image.	<code>sudo drgn</code>	Provides information for debugging the running live kernel.
Start a Python shell to analyze the contents of a different dump file for a running kernel or <code>vmcore</code> crash dump.	<code>sudo drgn -c path/to/dumpfile</code>	Provides information for debugging a running kernel or generated crash dump.
Start a Python shell to analyze the contents of a dump file and specify the <code>vmlinux</code> and module symbols.	<code>sudo drgn -c path/to/dumpfile -s path/to/vmlinux</code>	Provides information for debugging a running kernel, or generated crash dump, filtered by relevant kernel modules.
Review further options provided with the <code>drgn</code> command.	<code>sudo drgn -h</code>	Provides a listing of command line options for the <code>drgn</code> command.

Note

Type `exit()` or press the `Ctrl + D` keys to exit the Python shell.

For example, to debug `/proc/kcore` for a live kernel and specify kernel drivers, run the following command:

```
sudo drgn -c /proc/kcore -s /usr/lib/debug/lib/modules/$(uname -r)/vmlinux \
-s /lib/modules/$(uname -r)/kernel/net/netfilter/xt_comment.ko.xz
```

To perform the same operation on a `vmcore` crash dump file:

```
sudo drgn -c /var/crash/127.0.0.1-2023-06-02-09:33:07/vmcore \
-s /usr/lib/debug/lib/modules/5.15.0-101.103.2.1.el9uek.x86_64/vmlinux \
-s /lib/modules/5.15.0-101.103.2.1.el9uek.x86_64/kernel/net/netfilter/
xt_comment.ko.xz
```

7

Using the `drgn` Library With Python

Debug live kernels and `vmcore` crash dumps in a Python shell, and Python scripts, by importing the `drgn` library.

Before you can start using `drgn` with Python scripts, ensure that Python is correctly installed on the system. For more information, see [Oracle Linux 9: Installing and Managing Python](#).

If the script runs on Python 3.9, also install the `drgn` package. For more information, see [Installing `drgn`](#).

Note

To import the `drgn` library in scripts that run on newer versions of Python 3, enable the `ol9_addons` repository, then specify the version in the package name.

For example, you could install the `python3.12-drgn` package to import the `drgn` library in a script that runs on Python 3.12:

```
sudo dnf config-manager --enable ol9_addons
```

```
sudo dnf install python3.12-drgn
```

If no matching packages are available in the `ol9_addons` yum repository, then that Python version might no longer be supported. For more information, see [Oracle Linux: Product Life Cycle Information](#).

You can optionally run the `drgn` command with Python 3.12 as the interpreter by running the following command:

```
python3.12 -m drgn
```

Unlike the `crash` utility, `Drgn` wasn't originally designed to be a standalone kernel debugging tool. `Drgn` is a Python programming library that exposes debugging information for scripting and review purposes.

1. The `prog` array variable contains the information about the kernel that you're debugging. For example, to return the data collected for `slab_caches`, run the following statements in the `drgn` shell:

```
prog["slab_caches"]

(struct list_head){
    .next = (struct list_head *)0xffff8b831d972260,
```

```
        .prev = (struct list_head *)0xffff8b8007c02060,  
    }
```

2. Standard python structures can also be used to iterate through debug information:

```
slab_caches = prog["slab_caches"]
```

```
slab_caches.next
```

```
*(struct list_head *)0xffff8b831d972260 = {  
    .next = (struct list_head *)0xffff8b831d972460,  
    .prev = (struct list_head *)slab_caches+0x0 = 0xffffffff836e3da0,  
}
```

3. For more information about the `drgn` API and script syntax, see <https://drgn.readthedocs.io/>. Or, run the following command in the Python shell:

```
help(drgn)
```

The Python script loaded an array of kernel debugging information and crash data.

8

Getting Started With `corelens`

Debug a running kernel or `vmcore` crash dump file by using the `corelens` command.

Install the `drgn-tools` package. For more information, see [Installing `drgn-tools`](#).

The `corelens` command requires kernel debugging information to function. That can be provided by installing a kernel DebugInfo package. For more information, see [\(Optional\) Installing DebugInfo Packages](#).

Note

If no DebugInfo packages are installed, for example because the system being debugged is deployed in a production environment, the `corelens` command instead uses the more lightweight Common Type Format (CTF) debugging information if that's available.

CTF is available if the system is running Oracle Linux 9 with the Unbreakable Enterprise Kernel (UEK).

The `corelens` command can be used to troubleshoot system problems by analyzing the contents of system images and crash dumps.

1. To debug `/proc/kcore` for a live kernel, run the following command:

```
sudo corelens /proc/kcore
```

2. For more information about how to use the `corelens` command, use the `-h` option:

```
corelens -h
```

A brief summary of the system state is provided as output from the `corelens` command.

Command Reference

This table provides information about the `corelens` command.

Action	Command	Description
Review a summary of the system state for a running kernel or <code>vmcore</code> crash dump.	<code>sudo corelens path/to/dumpfile</code>	Provides information for debugging the running live kernel or generated crash dump.
Run a <code>corelens</code> module to analyze the contents of a running kernel or <code>vmcore</code> crash dump, and then review the results.	<code>sudo corelens path/to/dumpfile -M module</code>	Provides information for debugging a running kernel or generated crash dump filtered by module.
Review a list of modules that can be specified.	<code>corelens -L</code>	Provides a listing of module filters for use with the <code>-M</code> option.
Create a report based on the output from the <code>corelens</code> command.	<code>sudo corelens path/to/dumpfile -a -o report</code>	Generates a diagnostic report containing all the debugging information captured by the <code>corelens</code> command.
Review further options provided with the <code>corelens</code> command.	<code>corelens -h</code>	Provides a listing of command line options for the <code>drgn</code> command.

For example, to debug `/proc/kcore` for a live kernel, run the following command:

```
sudo corelens /proc/kcore
```

To perform the same operation on a `vmcore` crash dump file:

```
sudo corelens /var/crash/127.0.0.1-2025-07-29-09:33:07/vmcore
```

Selecting Modules for `corelens` Command Output

Use the `-M` option to filter the output from `corelens` commands.

The `corelens` command can also filter output based on the parts of the system that require diagnosis by using the `-M` option. For example, to reproduce the full output for a live kernel, activate the `sys` module:

```
sudo corelens /proc/kcore -M sys
```

Similarly, to display a list of I/O requests that are still in progress, activate the `inflight-io` module:

```
sudo corelens /proc/kcore -M inflight-io
```

More than one module can be specified by reusing the `-M` option for each module. For example, to reproduce the full output for a live kernel and all the mounted directories that are now present, activate the `sys` and `mounts` modules:

```
sudo corelens /proc/kcore -M sys -M mounts
```

Example output follows:

```
warning: Running corelens against a live system.
        Data may be inconsistent, or corelens may crash.

===== MODULE sys =====
MODE       : Live kernel
DATE       : Fri Jul 12 18:21:34 2024
NODENAME   : oracle-example-ol9
RELEASE    : 5.15.0-206.153.7.el9uek.x86_64
VERSION    : #2 SMP Thu May 9 15:59:05 PDT 2024
MACHINE    : x86_64
UPTIME     : 1 day, 19:47:22
LOAD AVERAGE: 0.08 , 0.02 , 0.01
JIFFIES    : 4452309417
MEMORY     : 7.49 GiB
TASKS      : 251 R:1 D:0 S:167 I:83
PLATFORM   : QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.6.6 08/22/2023
X86_HYPER_KVM
CPU_VENDOR: AuthenticAMD
MODEL NAME: AMD EPYC 7763 64-Core Processor
CPU_FAMILY: 25
CPUS       : 2
CPUS_NUMA0: 0-1
MICROCODE  : 0x1000065
CSTATES    : 9

===== MODULE mounts =====
DEVNAME    TYPE          DIRNAME
-----
none       rootfs        /
proc       proc          /proc
sysfs      sysfs         /sys
devtmpfs   devtmpfs     /dev
securityfs securityfs    /sys/kernel/security
tmpfs      tmpfs         /dev/shm
devpts     devpts        /dev/pts
tmpfs      tmpfs         /run
cgroup2    cgroup2       /sys/fs/cgroup
pstore     pstore        /sys/fs/pstore
efivarfs   efivarfs     /sys/firmware/efi/efivars
bpf        bpf           /sys/fs/bpf
configfs   configfs      /sys/kernel/config
```

```

/dev/mapper/ocivolume-root xfs /
rpc_pipefs                rpc_pipefs /var/lib/nfs/rpc_pipefs
selinuxfs                 selinuxfs /sys/fs/selinux
systemd-1                 autofs /proc/sys/fs/binfmt_misc
debugfs                   debugfs /sys/kernel/debug
hugetlbfs                 hugetlbfs /dev/hugepages
mqueue                    mqueue /dev/mqueue
tracefs                   tracefs /sys/kernel/tracing
none                      ramfs /run/credentials/systemd-
sysctl.service           none /run/credentials/systemd-tmpfiles-
setup-dev.service        fusectl /sys/fs/fuse/connections
fusectl                   fusectl /sys/fs/fuse/connections
/dev/mapper/ocivolume-oled xfs /var/oled
/dev/sdb2                  xfs /boot
/dev/sdb1                  vfat /boot/efi
none                      ramfs /run/credentials/systemd-tmpfiles-
setup.service            tmpfs /run/user/983
tmpfs                     tmpfs /run/user/1000
tmpfs                     tmpfs /run/user/1000

```

To see a full list of all the modules that can be specified, run the `corelens` command with the `-L` option:

```
corelens -L
```

For more information about what each `corelens` module does, use the `-h` option after specifying each of them with the `-M` option:

```
corelens -M module -h
```

For example, to learn more about the `dentrycache` module that outputs the kernel directory entry cache, use the following command:

```
corelens -M dentrycache -h
```

The following output might be displayed:

```
usage: dentrycache [-h] [--limit LIMIT] [--negative] [--detailed]
```

List dentries from the dentry hash table

optional arguments:

```

-h, --help show this help message and exit
--limit LIMIT, -l LIMIT
                list at most <number> dentries, 50 by default
--negative, -n list negative dentries only, disabled by default
--detailed, -d include inode, super, file type, refcount

```

Generating Reports With `corelens`

Use the provided `corelens` command options to generate reports for later review.

To generate a report from the `corelens` command, use the `-o` option and specify the output directory for that report. For example, to generate a report for the live kernel and output that report into a folder called `report` in the current working directory, use the following command:

```
sudo corelens /proc/kcore -a -o report
```

If you don't explicitly specify modules by using the `-M` option, use the `-a` option to generate a report using standard modules, or the `-A` option to generate the report using detailed modules.

Note

If you generate a report using every module, the final report might contain warnings that some modules couldn't be run. This is expected behavior, because some `corelens` modules require a core dump or can only function when specific kernel modules are loaded.

Diagnostic information is stored in a plain-text file for each module that was active when the `corelens` command was run. For example, to review the mounted directories that were output from the `mounts` module, view the contents of the `report/mounts` file:

```
cat report/mounts
```

Example output follows:

DEVNAME	TYPE	DIRNAME
-----	-----	-----
none	rootfs	/
proc	proc	/proc
sysfs	sysfs	/sys
devtmpfs	devtmpfs	/dev
securityfs	securityfs	/sys/kernel/security
tmpfs	tmpfs	/dev/shm
devpts	devpts	/dev/pts
tmpfs	tmpfs	/run
cgroup2	cgroup2	/sys/fs/cgroup
pstore	pstore	/sys/fs/pstore
efivarfs	efivarfs	/sys/firmware/efi/efivars
bpf	bpf	/sys/fs/bpf
configfs	configfs	/sys/kernel/config
/dev/mapper/ocivolume-root	xfs	/
rpc_pipefs	rpc_pipefs	/var/lib/nfs/rpc_pipefs
selinuxfs	selinuxfs	/sys/fs/selinux
systemd-1	autofs	/proc/sys/fs/binfmt_misc
hugetlbfs	hugetlbfs	/dev/hugepages
mqueue	mqueue	/dev/mqueue
debugfs	debugfs	/sys/kernel/debug
tracefs	tracefs	/sys/kernel/tracing
fusectl	fusectl	/sys/fs/fuse/connections
none	ramfs	/run/credentials/systemd-
sysctl.service		
none	ramfs	/run/credentials/systemd-tmpfiles-
setup-dev.service		

```
/dev/mapper/ocivolume-oled xfs /var/oled
/dev/sda2 xfs /boot
/dev/sda1 vfat /boot/efi
none ramfs /run/credentials/systemd-tmpfiles-
setup.service
tmpfs tmpfs /run/user/0
tmpfs tmpfs /run/user/982
tmpfs tmpfs /run/user/1000
```