SYSTEMD−ANALYZE

NAME

systemd-analyze − Analyze and debug system manager

SYNOPSIS

systemd−analyze [OPTIONS...] [time]
systemd−analyze [OPTIONS...] blame
systemd−analyze [OPTIONS...] critical−chain [***UNIT***...]
systemd−analyze [OPTIONS...] dump [***PATTERN***...]
systemd−analyze [OPTIONS...] plot [>file.svg]
systemd−analyze [OPTIONS...] dot [***PATTERN***...] [>file.dot]
systemd−analyze [OPTIONS...] unit−files
systemd−analyze [OPTIONS...] unit−paths
systemd−analyze [OPTIONS...] exit−status [***STATUS***...]
systemd−analyze [OPTIONS...] capability [***CAPABILITY***... | {−m | −−mask} ***MASK***]
systemd−analyze [OPTIONS...] condition ***CONDITION***...
systemd−analyze [OPTIONS...] syscall−filter [***SET***...]
systemd−analyze [OPTIONS...] filesystems [***SET***...]
systemd−analyze [OPTIONS...] calendar ***SPEC***...
systemd−analyze [OPTIONS...] timestamp ***TIMESTAMP***...
systemd−analyze [OPTIONS...] timespan ***SPAN***...
systemd−analyze [OPTIONS...] cat−config ***NAME***|***PATH***...
systemd−analyze [OPTIONS...] compare−versions ***VERSION1*** [***OP***] ***VERSION2***
systemd−analyze [OPTIONS...] verify ***FILE***...
systemd−analyze [OPTIONS...] security [***UNIT***...]
systemd−analyze [OPTIONS...] inspect−elf ***FILE***...
systemd−analyze [OPTIONS...] malloc [***D−BUS SERVICE***...]
systemd−analyze [OPTIONS...] fdstore ***UNIT***...
systemd−analyze [OPTIONS...] image−policy ***POLICY***...
systemd−analyze [OPTIONS...] has−tpm2
systemd−analyze [OPTIONS...] pcrs [***PCR***...]
systemd−analyze [OPTIONS...] srk [***>FILE***]
systemd−analyze [OPTIONS...] architectures [***NAME***...]
systemd−analyze [OPTIONS...] smbios11

DESCRIPTION

systemd−analyze may be used to determine system boot−up performance statistics and retrieve other state and tracing information from the system and service manager, and to verify the correctness of unit files. It is also used to access special functions useful for advanced system manager debugging.

If no command is passed, **systemd−analyze time** is implied.

systemd−analyze time

This command prints the time spent in the kernel before userspace has been reached, the time spent in the initrd before normal system userspace has been reached, and the time normal system userspace took to initialize. Note that these measurements simply measure the time passed up to the point where all system services have been spawned, but not necessarily until they fully finished initialization or the disk is idle.

Example 1. Show how long the boot took

```
$ systemd-analyze time # in a container
Startup finished in 296ms (userspace)
multi-user.target reached after 275ms in userspace

$ systemd-analyze time # on a real machine
Startup finished in 2.584s (kernel) + 19.176s (initrd) + 47.847s (userspace) = 1min 9.608s
multi-user.target reached after 47.820s in userspace
```

systemd−analyze blame

This command prints a list of all running units, ordered by the time they took to initialize. This information may be used to optimize boot−up times. Note that the output might be misleading as the initialization of one service might be slow simply because it waits for the initialization of another service to complete. Also note: **systemd−analyze blame** does not display results for services with ***Type=simple***, because systemd considers such services to be started immediately, hence no measurement of the initialization delays can be done. Also note that this command only shows the time units took for starting up, it does not show how long unit jobs spent in the execution queue. In particular it shows the time units spent in "activating" state, which is not defined for units such as device units that transition directly from "inactive" to "active". This command hence gives an impression of the performance of program code, but cannot accurately reflect latency introduced by waiting for hardware and similar events.

Example 2. Show which units took the most time during boot

```
$ systemd-analyze blame
         32.875s pmlogger.service
         20.905s systemd-networkd-wait-online.service
         13.299s dev-vda1.device
         ...
            23ms sysroot.mount
            11ms initrd-udevadm-cleanup-db.service
             3ms sys-kernel-config.mount
```

systemd−analyze critical−chain [UNIT...]

This command prints a tree of the time−critical chain of units (for each of the specified ***UNIT***s or for the default target otherwise). The time after the unit is active or started is printed after the "@" character. The time the unit takes to start is printed after the "+" character. Note that the output might be misleading as the initialization of services might depend on socket activation and because of the parallel execution of units. Also, similarly to the **blame** command, this only takes into account the time units spent in "activating" state, and hence does not cover units that never went through an "activating" state (such as device units that transition directly from "inactive" to "active"). Moreover, it does not show information on jobs (and in particular not jobs that timed out).

Example 3. systemd−analyze critical−chain

```
$ systemd-analyze critical-chain
multi-user.target @47.820s
└─pmie.service @35.968s +548ms
  └─pmcd.service @33.715s +2.247s
    └─network-online.target @33.712s
      └─systemd-networkd-wait-online.service @12.804s +20.905s
        └─systemd-networkd.service @11.109s +1.690s
          └─systemd-udevd.service @9.201s +1.904s
            └─systemd-tmpfiles-setup-dev.service @7.306s +1.776s
              └─kmod-static-nodes.service @6.976s +177ms
                └─systemd-journald.socket
                  └─system.slice
                    └─-.slice
```

systemd−analyze dump [pattern...]

Without any parameter, this command outputs a (usually very long) human−readable serialization of the complete service manager state. Optional glob pattern may be specified, causing the output to be limited to units whose names match one of the patterns. The output format is subject to change without notice and should not be parsed by applications. This command is rate limited for unprivileged users.

Example 4. Show the internal state of user manager

```
$ systemd-analyze --user dump
Timestamp userspace: Thu 2019-03-14 23:28:07 CET
Timestamp finish: Thu 2019-03-14 23:28:07 CET
Timestamp generators-start: Thu 2019-03-14 23:28:07 CET
Timestamp generators-finish: Thu 2019-03-14 23:28:07 CET
Timestamp units-load-start: Thu 2019-03-14 23:28:07 CET
Timestamp units-load-finish: Thu 2019-03-14 23:28:07 CET
-> Unit proc-timer_list.mount:
        Description: /proc/timer_list
        ...
-> Unit default.target:
        Description: Main user target
...
```

systemd−analyze malloc [D−Bus service...]

This command can be used to request the output of the internal memory state (as returned by **malloc_info**(3)) of a D−Bus service. If no service is specified, the query will be sent to org.freedesktop.systemd1 (the system or user service manager). The output format is not guaranteed to be stable and should not be parsed by applications.

The service must implement the org.freedesktop.MemoryAllocation1 interface. In the systemd suite, it is currently only implemented by the manager.

systemd−analyze plot

This command prints either an SVG graphic, detailing which system services have been started at what time, highlighting the time they spent on initialization, or the raw time data in JSON or table format.

Example 5. Plot a bootchart

```
$ systemd-analyze plot >bootup.svg
$ eog bootup.svg
```

Note that this plot is based on the most recent per−unit timing data of loaded units. This means that if a unit gets started, then stopped and then started again the information shown will cover the most recent start cycle, not the first one. Thus it is recommended to consult this information only shortly after boot, so that this distinction does not matter. Moreover, units that are not referenced by any other unit through a dependency might be unloaded by the service manager once they terminate (and did not fail). Such units will not show up in the plot.

systemd−analyze dot [pattern...]

This command generates textual dependency graph description in dot format for further processing with the GraphViz **dot**(1) tool. Use a command line like **systemd−analyze dot | dot −Tsvg >systemd.svg** to generate a graphical dependency tree. Unless **−−order** or **−−require** is passed, the generated graph will show both ordering and requirement dependencies. Optional pattern globbing style specifications (e.g. *.target) may be given at the end. A unit dependency is included in the graph if any of these patterns match either the origin or destination node.

Example 6. Plot all dependencies of any unit whose name starts with "avahi−daemon"

```
$ systemd-analyze dot 'avahi-daemon.*' | dot -Tsvg >avahi.svg
$ eog avahi.svg
```

Example 7. Plot the dependencies between all known target units

```
$ systemd-analyze dot --to-pattern='*.target' --from-pattern='*.target' \
      | dot -Tsvg >targets.svg
$ eog targets.svg
```

systemd−analyze unit−paths

This command outputs a list of all directories from which unit files, .d overrides, and .wants, .requires symlinks may be loaded. Combine with **−−user** to retrieve the list for the user manager instance, and **−−global** for the global configuration of user manager instances.

Example 8. Show all paths for generated units

```
$ systemd-analyze unit-paths | grep '^/run'
/run/systemd/system.control
/run/systemd/transient
/run/systemd/generator.early
/run/systemd/system
/run/systemd/system.attached
```

```
/run/systemd/generator
/run/systemd/generator.late
```

Note that this verb prints the list that is compiled into **systemd−analyze** itself, and does not communicate with the running manager. Use

```
systemctl [--user] [--global] show -p UnitPath --value
```

to retrieve the actual list that the manager uses, with any empty directories omitted.

systemd−analyze exit−status [STATUS...]

This command prints a list of exit statuses along with their "class", i.e. the source of the definition (one of "glibc", "systemd", "LSB", or "BSD"), see the Process Exit Codes section in **systemd.exec**(5). If no additional arguments are specified, all known statuses are shown. Otherwise, only the definitions for the specified codes are shown.

Example 9. Show some example exit status names

```
$ systemd-analyze exit-status 0 1 {63..65}
NAME    STATUS CLASS
SUCCESS 0      glibc
FAILURE 1      glibc
-       63     -
USAGE   64     BSD
DATAERR 65     BSD
```

systemd−analyze capability [CAPABILITY... | {−m | −−mask} MASK]

This command prints a list of Linux capabilities along with their numeric IDs. See **capabilities**(7) for details. If no argument is specified the full list of capabilities known to the service manager and the kernel is shown. Capabilities defined by the kernel but not known to the service manager are shown as "cap_???". Optionally, if arguments are specified they may refer to specific cabilities by name or numeric ID, in which case only the indicated capabilities are shown in the table.

Alternatively, if **−−mask** is passed, a single numeric argument must be specified, which is interpreted as a hexadecimal capability mask. In this case, only the capabilities present in the mask are shown in the table. This mode is intended to aid in decoding capability sets available via various debugging interfaces (e.g. "/proc/PID/status").

Example 10. Show some example capability names

```
$ systemd-analyze capability 0 1 {30..32}
NAME              NUMBER
cap_chown              0
cap_dac_override       1
cap_audit_control     30
cap_setfcap           31
cap_mac_override      32
```

Example 11. Decode a capability mask extracted from /proc

```
$ systemd-analyze capability -m 0000000000003c00
NAME                 NUMBER
cap_net_bind_service     10
cap_net_broadcast        11
cap_net_admin            12
cap_net_raw              13
```

systemd−analyze condition CONDITION...

This command will evaluate ***Condition*=...*** and ***Assert*=...*** assignments, and print their values, and the resulting value of the combined condition set. See **systemd.unit**(5) for a list of available conditions and asserts.

Example 12. Evaluate conditions that check kernel versions

```
$ systemd-analyze condition 'ConditionVersion = ! <4.0' \
        'ConditionVersion = >=5.1' \
        'ConditionACPower=|false' \
        'ConditionArchitecture=|!arm' \
        'AssertPathExists=/etc/os-release'
test.service: AssertPathExists=/etc/os-release succeeded.
Asserts succeeded.
test.service: ConditionArchitecture=|!arm succeeded.
test.service: ConditionACPower=|false failed.
test.service: ConditionVersion=>=5.1 succeeded.
test.service: ConditionVersion=!<4.0 succeeded.
Conditions succeeded.
```

systemd−analyze syscall−filter [SET...]

This command will list system calls contained in the specified system call set ***SET***, or all known sets if no sets are specified. Argument ***SET*** must include the "@" prefix.

systemd−analyze filesystems [SET...]

This command will list filesystems in the specified filesystem set ***SET***, or all known sets if no sets are specified. Argument ***SET*** must include the "@" prefix.

systemd−analyze calendar EXPRESSION...

This command will parse and normalize repetitive calendar time events, and will calculate when they elapse next. This takes the same input as the ***OnCalendar=*** setting in **systemd.timer**(5), following the syntax described in **systemd.time**(7). By default, only the next time the calendar expression will elapse is shown; use **−−iterations=** to show the specified number of next times the expression elapses. Each time the expression elapses forms a timestamp, see the **timestamp** verb below.

Example 13. Show leap days in the near future

```
$ systemd-analyze calendar --iterations=5 '*-2-29 0:0:0'
  Original form: *-2-29 0:0:0
Normalized form: *-02-29 00:00:00
    Next elapse: Sat 2020-02-29 00:00:00 UTC
       From now: 11 months 15 days left
       Iter. #2: Thu 2024-02-29 00:00:00 UTC
       From now: 4 years 11 months left
       Iter. #3: Tue 2028-02-29 00:00:00 UTC
       From now: 8 years 11 months left
       Iter. #4: Sun 2032-02-29 00:00:00 UTC
```

```
           From now: 12 years 11 months left
           Iter. #5: Fri 2036-02-29 00:00:00 UTC
           From now: 16 years 11 months left
```

systemd−analyze timestamp TIMESTAMP...

This command parses a timestamp (i.e. a single point in time) and outputs the normalized form and the difference between this timestamp and now. The timestamp should adhere to the syntax documented in **systemd.time**(7), section "PARSING TIMESTAMPS".

Example 14. Show parsing of timestamps

```
$ systemd-analyze timestamp yesterday now tomorrow
  Original form: yesterday
Normalized form: Mon 2019-05-20 00:00:00 CEST
       (in UTC): Sun 2019-05-19 22:00:00 UTC
   UNIX seconds: @15583032000
       From now: 1 day 9h ago

  Original form: now
Normalized form: Tue 2019-05-21 09:48:39 CEST
       (in UTC): Tue 2019-05-21 07:48:39 UTC
   UNIX seconds: @1558424919.659757
       From now: 43us ago

  Original form: tomorrow
Normalized form: Wed 2019-05-22 00:00:00 CEST
       (in UTC): Tue 2019-05-21 22:00:00 UTC
   UNIX seconds: @15584760000
       From now: 14h left
```

systemd−analyze timespan EXPRESSION...

This command parses a time span (i.e. a difference between two timestamps) and outputs the normalized form and the equivalent value in microseconds. The time span should adhere to the syntax documented in **systemd.time**(7), section "PARSING TIME SPANS". Values without units are parsed as seconds.

Example 15. Show parsing of timespans

```
$ systemd-analyze timespan 1s 300s '1year 0.000001s'
Original: 1s
      μs: 1000000
   Human: 1s

Original: 300s
      μs: 300000000
   Human: 5min

Original: 1year 0.000001s
      μs: 31557600000001
   Human: 1y 1us
```

systemd−analyze cat−config NAME|PATH...

This command is similar to **systemctl cat**, but operates on config files. It will copy the contents of a config file and any drop−ins to standard output, using the usual systemd set of directories and rules for precedence. Each argument must be either an absolute path including the prefix (such as /etc/systemd/logind.conf or /usr/lib/systemd/logind.conf), or a name relative to the prefix (such as systemd/logind.conf).

Example 16. Showing logind configuration

```
$ systemd-analyze cat-config systemd/logind.conf
# /etc/systemd/logind.conf
...
[Login]
NAutoVTs=8
...

# /usr/lib/systemd/logind.conf.d/20-test.conf
... some override from another package

# /etc/systemd/logind.conf.d/50-override.conf
... some administrator override
```

systemd−analyze compare−versions VERSION1 [OP] VERSION2

This command has two distinct modes of operation, depending on whether the operator ***OP*** is specified.

In the first mode — when ***OP*** is not specified —, it will compare the two version strings and print either "***VERSION1*** < ***VERSION2***", or "***VERSION1*** == ***VERSION2***", or "***VERSION1*** > ***VERSION2***" as appropriate.

The exit status is **0** if the versions are equal, **11** if the version of the right is smaller, and **12** if the version of the left is smaller. (This matches the convention used by **rpmdev−vercmp**.)

In the second mode — when ***OP*** is specified — it will compare the two version strings using the operation ***OP*** and return **0** (success) if they condition is satisfied, and **1** (failure) otherwise. **OP** may be **lt**, **le**, **eq**, **ne**, **ge**, **gt**. In this mode, no output is printed. (This matches the convention used by **dpkg**(1) **−−compare−versions**.)

Example 17. Compare versions of a package

```
$ systemd-analyze compare-versions systemd-250~rc1.fc36.aarch64 systemd-251.fc36.aarch64
systemd-250~rc1.fc36.aarch64 < systemd-251.fc36.aarch64
$ echo $?
12

$ systemd-analyze compare-versions 1 lt 2; echo $?
0
```

```
$ systemd-analyze compare-versions 1 ge 2; echo $?
1
```

systemd−analyze verify FILE...

This command will load unit files and print warnings if any errors are detected. Files specified on the command line will be loaded, but also any other units referenced by them. A unit's name on disk can be overridden by specifying an alias after a colon; see below for an example. The full unit search path is formed by combining the directories for all command line arguments, and the usual unit load paths. The variable ***$SYSTEMD_UNIT_PATH*** is supported, and may be used to replace or augment the compiled in set of unit load paths; see **systemd.unit**(5). All units files present in the directories containing the command line arguments will be used in preference to the other paths. If a template unit without an instance name is specified (e.g. foo@.service), "test_instance" will be used as the instance name, which can be controlled by **−−instance=** option.

The following errors are currently detected:

- unknown sections and directives,
- missing dependencies which are required to start the given unit,
- man pages listed in ***Documentation=*** which are not found in the system,
- commands listed in ***ExecStart=*** and similar which are not found in the system or not executable.

Example 18. Misspelt directives

```
$ cat ./user.slice
[Unit]
WhatIsThis=11
Documentation=man:nosuchfile(1)
Requires=different.service

[Service]
Description=x

$ systemd-analyze verify ./user.slice
[./user.slice:9] Unknown lvalue 'WhatIsThis' in section 'Unit'
[./user.slice:13] Unknown section 'Service'. Ignoring.
Error: org.freedesktop.systemd1.LoadFailed:
   Unit different.service failed to load:
   No such file or directory.
Failed to create user.slice/start: Invalid argument
user.slice: man nosuchfile(1) command failed with code 16
```

Example 19. Missing service units

```
$ tail ./a.socket ./b.socket
==> ./a.socket <==
[Socket]
ListenStream=100

==> ./b.socket <==
[Socket]
ListenStream=100
Accept=yes

$ systemd-analyze verify ./a.socket ./b.socket
Service a.service not loaded, a.socket cannot be started.
Service b@0.service not loaded, b.socket cannot be started.
$ cat /tmp/source
[Unit]
Description=Hostname printer

[Service]
Type=simple
ExecStart=/usr/bin/echo %H
MysteryKey=true

$ systemd-analyze verify /tmp/source
Failed to prepare filename /tmp/source: Invalid argument

$ systemd-analyze verify /tmp/source:alias.service
alias.service:7: Unknown key name 'MysteryKey' in section 'Service', ignoring.
```

systemd−analyze security [UNIT...]

This command analyzes the security and sandboxing settings of one or more specified service units. If at least one unit name is specified the security settings of the specified service units are inspected and a detailed analysis is shown. If no unit name is specified, all currently loaded, long−running service units are inspected and a terse table with results shown. The command checks for various security−related service settings, assigning each a numeric "exposure level" value, depending on how important a setting is. It then calculates an overall exposure level for the whole unit, which is an estimation in the range 0.0...10.0 indicating how exposed a service is security−wise. High exposure levels indicate very little applied sandboxing. Low exposure levels indicate tight sandboxing and strongest security restrictions. Note that this only analyzes the per−service security features systemd itself implements. This means that any additional security mechanisms applied by the service code itself are not accounted for. The exposure level determined this way should not be misunderstood: a high exposure level neither means that there is no effective sandboxing applied by the service code itself, nor that the service is actually vulnerable to remote or local attacks. High exposure levels do indicate however that most likely the service might benefit from additional settings applied to them.

Please note that many of the security and sandboxing settings individually can be circumvented — unless combined with others. For example, if a service retains the privilege to establish or undo mount points many of the sandboxing options can be undone by the service code itself. Due to that is essential that each service uses the most comprehensive and strict sandboxing and security settings possible. The tool will take into account some of these combinations and relationships between the settings, but not all. Also note that the security and sandboxing settings analyzed here only apply to the operations executed by the service code itself. If a service has access to an IPC system (such as D−Bus) it might request operations from other services that are not subject to the same restrictions. Any comprehensive security and sandboxing analysis is hence incomplete if the IPC access policy is not validated too.

Example 21. Analyze systemd−logind.service

```
$ systemd-analyze security --no-pager systemd-logind.service
  NAME                DESCRIPTION                              EXPOSURE
✗ PrivateNetwork=     Service has access to the host's network      0.5
✗ User=/DynamicUser=  Service runs as root user                     0.4
✗ DeviceAllow=        Service has no device ACL                     0.2
✓ IPAddressDeny=      Service blocks all IP address ranges
...
→ Overall exposure level for systemd-logind.service: 4.1 OK 🙂
```

systemd−analyze inspect−elf FILE...

This command will load the specified files, and if they are ELF objects (executables, libraries, core files, etc.) it will parse the embedded packaging metadata, if any, and print it in a table or json format. See the **Packaging Metadata** documentation for more information.

Example 22. Print information about a core file as JSON

```
$ systemd-analyze inspect-elf --json=pretty \
        core.fsverity.1000.f77dac5dc161402aa44e15b7dd9dcf97.58561.1637106137000000
{
        "elfType" : "coredump",
        "elfArchitecture" : "AMD x86-64",
        "/home/bluca/git/fsverity-utils/fsverity" : {
                "type" : "deb",
                "name" : "fsverity-utils",
                "version" : "1.3-1",
                "buildId" : "7c895ecd2a271f93e96268f479fdc3c64a2ec4ee"
        },
        "/home/bluca/git/fsverity-utils/libfsverity.so.0" : {
                "type" : "deb",
                "name" : "fsverity-utils",
                "version" : "1.3-1",
                "buildId" : "b5e428254abf14237b0ae70ed85fffbb98a78f88"
        }
}
```

systemd−analyze fdstore UNIT...

Lists the current contents of the specified service unit's file descriptor store. This shows names, inode types, device numbers, inode numbers, paths and open modes of the open file descriptors. The specified units must have ***FileDescriptorStoreMax=*** enabled, see **systemd.service**(5) for details.

Example 23. Table output

```
$ systemd-analyze fdstore systemd-journald.service
FDNAME TYPE DEVNO   INODE RDEVNO PATH             FLAGS
stored sock 0:8   4218620 -      socket:[4218620] ro
stored sock 0:8   4213198 -      socket:[4213198] ro
stored sock 0:8   4213190 -      socket:[4213190] ro
…
```

Note: the "DEVNO" column refers to the major/minor numbers of the device node backing the file system the file descriptor's inode is on. The "RDEVNO" column refers to the major/minor numbers of the device node itself if the file descriptor refers to one. Compare with corresponding ***.st_dev*** and ***.st_rdev*** fields in **struct stat** (see **stat**(2) for details). The listed inode numbers in the "INODE" column are on the file system indicated by "DEVNO".

systemd−analyze image−policy POLICY...

This command analyzes the specified image policy string, as per **systemd.image-policy**(7). The policy is normalized and simplified. For each currently defined partition identifier (as per the **Discoverable Partitions Specification**) the effect of the image policy string is shown in tabular form.

Example 24. Example Output

```
$ systemd-analyze image-policy swap=encrypted:usr=read-only-on+verity:root=encrypted
Analyzing policy: root=encrypted:usr=verity+read-only-on:swap=encrypted
       Long form: root=encrypted:usr=verity+read-only-on:swap=encrypted:=unused+absent

PARTITION       MODE        READ-ONLY GROWFS
root            encrypted   -         -
usr             verity      yes       -
home            ignore      -         -
srv             ignore      -         -
esp             ignore      -         -
xbootldr        ignore      -         -
swap            encrypted   -         -
root-verity     ignore      -         -
usr-verity      unprotected yes       -
root-verity-sig ignore      -         -
usr-verity-sig  ignore      -         -
tmp             ignore      -         -
var             ignore      -         -
default         ignore      -         -
```

systemd−analyze has−tpm2

Reports whether the system is equipped with a usable TPM2 device. If a TPM2 device has been discovered, is supported, and is being used by firmware, by the OS kernel drivers and by userspace (i.e. systemd) this prints "yes" and exits with exit status zero. If no such device is discovered/supported/used, prints "no". Otherwise, prints "partial". In either of these two cases exits with non−zero exit status. It also shows five lines indicating separately whether firmware, drivers, the system, the kernel and libraries discovered/support/use TPM2. Currently, required libraries are libtss2−esys.so.0, libtss2−rc.so.0, and libtss2−mu.so.0. The requirement may be changed in the future release.

Note, this checks for TPM 2.0 devices only, and does not consider TPM 1.2 at all.

Combine with **--quiet** to suppress the output.

Example 25. Example Output

```
yes
+firmware
+driver
+system
+subsystem
+libraries
  +libtss2-esys.so.0
  +libtss2-rc.so.0
  +libtss2-mu.so.0
```

Added in version 257.

systemd-analyze pcrs [PCR...]

This command shows the known TPM2 PCRs along with their identifying names and current values.

Example 26. Example Output

```
$ systemd-analyze pcrs
NR NAME                SHA256
 0 platform-code       bcd2eb527108bbb1f5528409bcbe310aa9b74f687854cc5857605993f3d9eb11
 1 platform-config     b60622856eb7ce52637b80f30a520e6e87c347daa679f3335f4f1a600681bb01
 2 external-code       1471262403e9a62f9c392941300b4807fbdb6f0bfdd50abfab752732087017dd
 3 external-config     3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
 4 boot-loader-code    939f7fa1458e1f7ce968874d908e524fc0debf890383d355e4ce347b7b78a95c
 5 boot-loader-config  864c61c5ea5ecbdb6951e6cb6d9c1f4b4eac79772f7fe13b8bece569d83d3768
 6 -                   3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
 7 secure-boot-policy  9c905bd9b9891bfb889b90a54c4b537b889cfa817c4389cc25754823a9443255
 8 -                   0000000000000000000000000000000000000000000000000000000000000000
 9 kernel-initrd       9caa29b128113ef42aa53d421f03437be57211e5ebafc0fa8b5d4514ee37ff0c
10 ima                 5ea9e3dab53eb6b483b6ec9e3b2c712bea66bca1b155637841216e0094387400
11 kernel-boot         0000000000000000000000000000000000000000000000000000000000000000
12 kernel-config       627ffa4b405e911902fe1f1a8b0164693b31acab04f805f15bccfe2209c7eace
13 sysexts             0000000000000000000000000000000000000000000000000000000000000000
14 shim-policy         0000000000000000000000000000000000000000000000000000000000000000
15 system-identity     0000000000000000000000000000000000000000000000000000000000000000
16 debug               0000000000000000000000000000000000000000000000000000000000000000
17 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
18 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
19 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
20 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
21 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
22 -                   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
23 application-support 0000000000000000000000000000000000000000000000000000000000000000
```

systemd-analyze srk [>FILE]

This command reads the Storage Root Key (SRK) from the TPM2 device, and writes it in marshalled TPM2B_PUBLIC format to stdout. The output is non-printable data, so it should be redirected to a file or into a pipe.

Example 27. Save the Storage Root Key to srk.tpm2b_public

```
systemd-analyze srk >srk.tpm2b_public
```

systemd-analyze architectures [NAME...]

Lists all known CPU architectures, and which ones are native. The listed architecture names are those ***ConditionArchitecture=*** supports, see **systemd.unit**(5) for details. If architecture names are specified only those specified are listed.

Example 28. Table output

```
$ systemd-analyze architectures
NAME        SUPPORT
alpha       foreign
arc         foreign
arc-be      foreign
arm         foreign
arm64       foreign
…
sparc       foreign
sparc64     foreign
tilegx      foreign
x86         secondary
x86-64      native
```

systemd-analyze smbios11

Shows a list of SMBIOS Type #11 strings passed to the system. Also see **smbios-type-11**(7).

Example 29. Example output

```
$ systemd-analyze smbios11
io.systemd.stub.kernel-cmdline-extra=console=ttyS0
io.systemd.credential.binary:ssh.ephemeral-authorized_keys-all=c3NoLWVkMjU1MTkgQUFBQUMzTnphQzFsWkRJMU5URTVBQUFBSURGd20xbFp4WlRGclJteG9ZQlozOTYzcE1uYlJCaDMwM1MxVXhLSUM2NmYgbGVubmFydEEE
io.systemd.credential:vmm.notify_socket=vsock-stream:2:254570042

3 SMBIOS Type #11 strings passed.
```

Added in version 257.

systemd-analyze chid

Shows a list of Computer Hardware IDs (CHIDs) of the local system. These IDs identify the system's computer hardware, based on SMBIOS data. See Using Computer Hardware IDs (CHIDs) for details about CHIDs.

Example 30. Example Output

```
$ systemd-analyze chid
TYPE INPUT  CHID
   3 MFPSmp 520537c0-3b59-504f-b062-9682ea236b21
   4 MFPS-- edf05dc8-a53d-5b2c-8023-630bca2a2463
   5 MFP--- ebc6a4d9-ec48-537a-916b-c69fa4fdd814
   6 M--Smp 5ebe4bba-f598-5e90-9ff2-9fd0d3211465
   7 M--S-- 1a3fb835-b42a-5f9c-a38c-eff5bfd5c41d
   8 M-P-mp 2a831dce-8163-5bad-8406-435b8c752dd8
   9 M-P--- 7c21c878-4a75-50f7-9816-21e811588da0
  10 MF--mp 9a003537-bcc5-500e-b10a-8d8892e4fc64
  11 MF---- bb9122bb-8a5c-50d2-a742-a85beb719909
  13 M---mp bfc36935-5032-5987-a0a3-6311f01de33a

LEGEND: M → sys_vendor (LENOVO) ┄ F → product_family (ThinkPad X1 Carbon Gen 9) ┄ P → product_name (20XW0055GE)
        S → product_sku (LENOVO_MT_20XW_BU_Think_FM_ThinkPad X1 Carbon Gen 9) ┄ m → board_vendor (LENOVO)
        p → board_name (20XW0055GE)
```

Added in version 258.

systemd−analyze transient-settings TYPE...

Lists properties that can be set for various unit types via command line interfaces, in particular systemctl(1) set-property and --property=/--automount-property= options in systemd-run(1), systemd-nspawn(1), and systemd-mount(1). Those assignments are possible for a subset of the properties that can be set in config files, see systemd.unit(5), systemd.exec(5), systemd.resource-control(5), and the other unit-type-specific pages. The TYPE argument must be a unit type ("service", "socket", ...). The properties that apply to the specific types are listed.

Note: D-Bus properties documented in org.freedesktop.systemd1(5) form a partially overlapping set with the lists generated by this command. Many D-Bus properties and transient settings share the same names, but for example, LogRateLimitIntervalSec= is described in systemd.exec(5) and would be listed by this command, but the corresponding D-Bus property described in systemd.exec(5) is LogRateLimitIntervalUSec.

This verb is intended primarily for programmatic generation of shell completions.

Added in version 258.

systemd-analyze unit-shell SERVICE [command...]

The given command runs on the namespace of the specified running service. If no command is given, spawn and attach a shell with the namespace to the service.

Example 31. Example Output

```
$ systemd-analyze unit-shell systemd-resolved.service ls
bin   dev  etc     home  lib    lib64   lost+found  mnt  proc  run   srv  tmp  var      vmlinuz.old
boot  efi  exitrd  init  lib32  libx32  media       opt  root  sbin  sys  usr  vmlinuz  work
```

Added in version 258.

systemd-analyze unit-gdb SERVICE

Spawn and attach a debugger to the given service. By default, gdb(1) will be used. This may be changed using the --debugger= option or the $SYSTEMD_DEBUGGER environment variable. Use the --debugger-arguments= option to pass extra command line arguments to the debugger and quote as appropriate when ARGS contain whitespace (See Example).

Example 32. Example Output

```
$ systemd-analyze --debugger-arguments="-batch -ex 'info all-registers'" unit-gdb systemd-oomd.service
```

Added in version 258.

OPTIONS

The following options are understood:

−−system

Operates on the system systemd instance. This is the implied default.
Added in version 209.

−−user

Operates on the user systemd instance.
Added in version 186.

−−global

Operates on the system−wide configuration for user systemd instance.
Added in version 238.

−−order, **−−require**

When used in conjunction with the **dot** command (see above), selects which dependencies are shown in the dependency graph. If **−−order** is passed, only dependencies of type ***After=*** or ***Before=*** are shown. If **−−require** is passed, only dependencies of type ***Requires=***, ***Requisite=***, ***BindsTo=***, ***Wants=***, and ***Conflicts=*** are shown. If neither is passed, this shows dependencies of all these types.
Added in version 198.

−−from−pattern=, **−−to−pattern=**

When used in conjunction with the **dot** command (see above), this selects which relationships are shown in the dependency graph. Both options require a **glob**(7) pattern as an argument, which will be matched against the left−hand and the right−hand, respectively, nodes of a relationship.
Each of these can be used more than once, in which case the unit name must match one of the values. When tests for both sides of the relation are present, a relation must pass both tests to be shown. When patterns are also specified as positional arguments, they must match at least one side of the relation. In other words, patterns specified with those two options will trim the list of edges matched by the positional arguments, if any are given, and fully determine the list of edges shown otherwise.
Added in version 201.

−−fuzz=*timespan*

When used in conjunction with the **critical−chain** command (see above), also show units, which finished ***timespan*** earlier, than the latest unit in the same level. The unit of ***timespan*** is seconds unless specified with a different unit, e.g. "50ms".

Added in version 203.

−−man=no
Do not invoke **man**(1) to verify the existence of man pages listed in ***Documentation=***.
Added in version 235.

−−generators
Invoke unit generators, see **systemd.generator**(7). Some generators require root privileges. Under a normal user, running with generators enabled will generally result in some warnings.
Added in version 235.

−−instance=NAME
Specifies fallback instance name for template units. This will be used when one or more template units without an instance name (e.g. foo@.service) specified for **systemd−analyze condition** with **−−unit=**, **systemd−analyze security**, and **systemd−analyze verify**. If unspecified, "test_instance" will be used.
Added in version 257.

−−recursive−errors=*MODE*
Control verification of units and their dependencies and whether **systemd−analyze verify** exits with a non−zero process exit status or not. With **yes**, return a non−zero process exit status when warnings arise during verification of either the specified unit or any of its associated dependencies. With **no**, return a non−zero process exit status when warnings arise during verification of only the specified unit. With **one**, return a non−zero process exit status when warnings arise during verification of either the specified unit or its immediate dependencies. If this option is not specified, zero is returned as the exit status regardless whether warnings arise during verification or not.
Added in version 250.

−−root=*PATH*
With **cat−config**, **verify**, **condition** and **security** when used with **−−offline=**, operate on files underneath the specified root path ***PATH***.
Added in version 239.

−−image=*PATH*
With **cat−config**, **verify**, **condition** and **security** when used with **−−offline=**, operate on files inside the specified image path ***PATH***.
Added in version 250.

−−image−policy=*policy*
Takes an image policy string as argument, as per **systemd.image-policy**(7). The policy is enforced when operating on the disk image specified via **−−image=**, see above. If not specified, defaults to the "*" policy, i.e. all recognized file systems in the image are used.

−−offline=*BOOL*
With **security**, perform an offline security review of the specified unit files, i.e. does not have to rely on PID 1 to acquire security information for the files like the **security** verb when used by itself does. This means that **−−offline=** can be used with **−−root=** and **−−image=** as well. If a unit's overall exposure level is above that set by **−−threshold=** (default value is 100), **−−offline=** will return an error.
Added in version 250.

−−profile=*PATH*
With **security −−offline=**, takes into consideration the specified portable profile when assessing unit settings. The profile can be passed by name, in which case the well−known system locations will be searched, or it can be the full path to a specific drop−in file.
Added in version 250.

−−threshold=*NUMBER*
With **security**, allow the user to set a custom value to compare the overall exposure level with, for the specified unit files. If a unit's overall exposure level, is greater than that set by the user, **security** will return an error. **−−threshold=** can be used with **−−offline=** as well and its default value is 100.
Added in version 250.

−−security−policy=*PATH*
With **security**, allow the user to define a custom set of requirements formatted as a JSON file against which to compare the specified unit file(s) and determine their overall exposure level to security threats.
Table 1. Accepted Assessment Test Identifiers See example "JSON Policy" below.
Added in version 250.

−−json=*MODE*
With the **security** command, generate a JSON formatted output of the security analysis table. The format is a JSON array with objects containing the following fields: ***set*** which indicates if the setting has been enabled or not, ***name*** which is what is used to refer to the setting, ***json_field*** which is the JSON compatible identifier of the setting, ***description*** which is an outline of the setting state, and ***exposure*** which is a number in the range 0.0...10.0, where a higher value corresponds to a higher security threat. The JSON version of the table is printed to standard output. The ***MODE*** passed to the option can be one of three: **off** which is the default, **pretty** and **short** which respectively output a prettified or shorted JSON version of the security table. With the **plot** command, generate a JSON formatted output of the raw time data. The format is a JSON array with objects containing the following fields: ***name*** which is the unit name, ***activated*** which is the time after startup the service was activated, ***activating*** which is how long after startup the service was initially started, ***time*** which is how long the service took to activate from when it was initially started, ***deactivated*** which is the time after startup that the service was deactivated, ***deactivating*** which is the time after startup that the service was initially told to deactivate.
Added in version 250.

−−iterations=*NUMBER*
When used with the **calendar** command, show the specified number of iterations the specified calendar expression will elapse next. Defaults to 1.
Added in version 242.

−−base−time=*TIMESTAMP*
When used with the **calendar** command, show next iterations relative to the specified point in time. If not specified, defaults to the current time.
Added in version 244.

−−unit=*UNIT*
When used with the **condition** command, evaluate all the ***Condition*=...*** and ***Assert*=...*** assignments in the specified unit file. The full unit search path is formed by combining the directories for the specified unit with the usual unit load paths. The variable ***$SYSTEMD_UNIT_PATH*** is supported, and may be used to replace or augment the compiled in set of unit load paths; see **systemd.unit**(5). All units files present in the directory containing the specified unit will be used in preference to the other paths. If a template unit without an instance name is specified (e.g. foo@.service), "test_instance" will be used as the instance name, which can be controlled by **−−instance=** option.
Added in version 250.

−−table
When used with the **plot** command, the raw time data is output in a table.
Added in version 253.

−−no−legend
When used with the **plot** command in combination with either **−−table** or **−−json=**, no legends or hints are included in the output.

Added in version 253.

−H, **−−host=**

Execute the operation remotely. Specify a hostname, or a username and hostname separated by "@", to connect to. The hostname may optionally be suffixed by a port ssh is listening on, separated by ":", and then a container name, separated by "/", which connects directly to a specific container on the specified host. This will use SSH to talk to the remote machine manager instance. Container names may be enumerated with **machinectl −H** ***HOST***. Put IPv6 addresses in brackets.

−M, **−−machine=**

Execute operation on a local container. Specify a container name to connect to, optionally prefixed by a user name to connect as and a separating "@" character. If the special string ".host" is used in place of the container name, a connection to the local system is made (which is useful to connect to a specific user's user bus: "−−user −−machine=lennart@.host"). If the "@" syntax is not used, the connection is made as root user. If the "@" syntax is used either the left hand side or the right hand side may be omitted (but not both) in which case the local user name and ".host" are implied.

−q, **−−quiet**

Suppress hints and other non−essential output.

Added in version 250.

−−tldr

With **cat−config**, only print the "interesting" parts of the configuration files, skipping comments and empty lines and section headers followed only by comments and empty lines.

Added in version 255.

−−scale−svg=*FACTOR*

When used with the **plot** command, the x−axis of the plot can be stretched by FACTOR (default: 1.0).

Added in version 257.

−−detailed

When used with the **plot** command, activation timestamps details can be seen in SVG plot.

Added in version 257.

−h, **−−help**

Print a short help text and exit.

−−version

Print a short version string and exit.

−−no−pager

Do not pipe output into a pager.

EXIT STATUS

For most verbs, **0** is returned on success, and a non−zero failure code otherwise.

For the verb **compare−versions**, in the two−argument form, **12**, **0**, or **11** are returned if the second version string is respectively larger than, equal to, or smaller than the first. In the three−argument form, **0** or **1** are returned when the condition is respectively true or false.

For the verb **has−tpm2**, **0** is returned if a TPM2 device is discovered, supported, and used by firmware, driver, and userspace (i.e. **systemd**). Otherwise, the OR combination of the value **1** (in case firmware support is missing), **2** (in case driver support is missing), and **4** (in case userspace support is missing). If no TPM2 support is available at all, value **7** is hence returned.

ENVIRONMENT

$SYSTEMD_LOG_LEVEL

The maximum log level of emitted messages (messages with a higher log level, i.e. less important ones, will be suppressed). Takes a comma−separated list of values. A value may be either one of (in order of decreasing importance) **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, **debug**, or an integer in the range 0...7. See **syslog**(3) for more information. Each value may optionally be prefixed with one of **console**, **syslog**, **kmsg** or **journal** followed by a colon to set the maximum log level for that specific log target (e.g. **SYSTEMD_LOG_LEVEL=debug,console:info** specifies to log at debug level except when logging to the console which should be at info level). Note that the global maximum log level takes priority over any per target maximum log levels.

$SYSTEMD_LOG_COLOR

A boolean. If true, messages written to the tty will be colored according to priority.

This setting is only useful when messages are written directly to the terminal, because **journalctl**(1) and other tools that display logs will color messages based on the log level on their own.

$SYSTEMD_LOG_TIME

A boolean. If true, console log messages will be prefixed with a timestamp.

This setting is only useful when messages are written directly to the terminal or a file, because **journalctl**(1) and other tools that display logs will attach timestamps based on the entry metadata on their own.

$SYSTEMD_LOG_LOCATION

A boolean. If true, messages will be prefixed with a filename and line number in the source code where the message originates.

Note that the log location is often attached as metadata to journal entries anyway. Including it directly in the message text can nevertheless be convenient when debugging programs.

$SYSTEMD_LOG_TID

A boolean. If true, messages will be prefixed with the current numerical thread ID (TID).

Note that the this information is attached as metadata to journal entries anyway. Including it directly in the message text can nevertheless be convenient when debugging programs.

$SYSTEMD_LOG_TARGET

The destination for log messages. One of **console** (log to the attached tty), **console−prefixed** (log to the attached tty but with prefixes encoding the log level and "facility", see **syslog**(3), **kmsg** (log to the kernel circular log buffer), **journal** (log to the journal), **journal−or−kmsg** (log to the journal if available, and to kmsg otherwise), **auto** (determine the appropriate log target automatically, the default), **null** (disable log output).

$SYSTEMD_LOG_RATELIMIT_KMSG

Whether to ratelimit kmsg or not. Takes a boolean. Defaults to "true". If disabled, systemd will not ratelimit messages written to kmsg.

$SYSTEMD_PAGER, ***$PAGER***

Pager to use when **−−no−pager** is not given. ***$SYSTEMD_PAGER*** is used if set; otherwise ***$PAGER*** is used. If neither ***$SYSTEMD_PAGER*** nor ***$PAGER*** are set, a set of well−known pager implementations is tried in turn, including **less**(1) and **more**(1), until one is found. If no pager implementation is discovered, no pager is invoked. Setting those environment variables to an empty string or the value "cat" is equivalent to passing **−−no−pager**.

Note: if ***$SYSTEMD_PAGERSECURE*** is not set, ***$SYSTEMD_PAGER*** and ***$PAGER*** can only be used to disable the pager (with "cat" or ""), and are otherwise ignored.

$SYSTEMD_LESS

Override the options passed to **less** (by default "FRSXMK").

Users might want to change two options in particular:

K

This option instructs the pager to exit immediately when Ctrl+C is pressed. To allow **less** to handle Ctrl+C itself to switch back to the pager command prompt, unset this option.

If the value of *$SYSTEMD_LESS* does not include "K", and the pager that is invoked is **less**, Ctrl+C will be ignored by the executable, and needs to be handled by the pager.

X

This option instructs the pager to not send termcap initialization and deinitialization strings to the terminal. It is set by default to allow command output to remain visible in the terminal even after the pager exits. Nevertheless, this prevents some pager functionality from working, in particular paged output cannot be scrolled with the mouse.

Note that setting the regular *$LESS* environment variable has no effect for **less** invocations by systemd tools.

See **less**(1) for more discussion.

$SYSTEMD_LESSCHARSET

Override the charset passed to **less** (by default "utf−8", if the invoking terminal is determined to be UTF−8 compatible).

Note that setting the regular *$LESSCHARSET* environment variable has no effect for **less** invocations by systemd tools.

$SYSTEMD_PAGERSECURE

Common pager commands like **less**(1), in addition to "paging", i.e. scrolling through the output, support opening of or writing to other files and running arbitrary shell commands. When commands are invoked with elevated privileges, for example under **sudo**(8) or **pkexec**(1), the pager becomes a security boundary. Care must be taken that only programs with strictly limited functionality are used as pagers, and unintended interactive features like opening or creation of new files or starting of subprocesses are not allowed. "Secure mode" for the pager may be enabled as described below, ***if the pager supports that*** (most pagers are not written in a way that takes this into consideration). It is recommended to either explicitly enable "secure mode" or to completely disable the pager using **−−no−pager** or *PAGER=cat* when allowing untrusted users to execute commands with elevated privileges.

This option takes a boolean argument. When set to true, the "secure mode" of the pager is enabled. In "secure mode", **LESSSECURE=1** will be set when invoking the pager, which instructs the pager to disable commands that open or create new files or start new subprocesses. Currently only **less**(1) is known to understand this variable and implement "secure mode".

When set to false, no limitation is placed on the pager. Setting *SYSTEMD_PAGERSECURE=0* or not removing it from the inherited environment may allow the user to invoke arbitrary commands.

When *$SYSTEMD_PAGERSECURE* is not set, systemd tools attempt to automatically figure out if "secure mode" should be enabled and whether the pager supports it. "Secure mode" is enabled if the effective UID is not the same as the owner of the login session, see **geteuid**(2) and **sd_pid_get_owner_uid**(3), or when running under **sudo**(8) or similar tools (*$SUDO_UID* is set). In those cases, *SYSTEMD_PAGERSECURE=1* will be set and pagers which are not known to implement "secure mode" will not be used at all. Note that this autodetection only covers the most common mechanisms to elevate privileges and is intended as convenience. It is recommended to explicitly set *$SYSTEMD_PAGERSECURE* or disable the pager.

Note that if the *$SYSTEMD_PAGER* or *$PAGER* variables are to be honoured, other than to disable the pager, *$SYSTEMD_PAGERSECURE* must be set too.

$SYSTEMD_COLORS

Takes a boolean argument. When true, **systemd** and related utilities will use colors in their output, otherwise the output will be monochrome. Additionally, the variable can take one of the following special values: "16", "256" to restrict the use of colors to the base 16 or 256 ANSI colors, respectively. This can be specified to override the automatic decision based on *$TERM* and what the console is connected to.

$SYSTEMD_URLIFY

The value must be a boolean. Controls whether clickable links should be generated in the output for terminal emulators supporting this. This can be specified to override the decision that **systemd** makes based on *$TERM* and other conditions.

EXAMPLES

Example 30. JSON Policy

The JSON file passed as a path parameter to **−−security−policy=** has a top−level JSON object, with keys being the assessment test identifiers mentioned above. The values in the file should be JSON objects with one or more of the following fields: **description_na** (string), **description_good** (string), **description_bad** (string), **weight** (unsigned integer), and **range** (unsigned integer). If any of these fields corresponding to a specific id of the unit file is missing from the JSON object, the default built−in field value corresponding to that same id is used for security analysis as default. The weight and range fields are used in determining the overall exposure level of the unit files: the value of each setting is assigned a badness score, which is multiplied by the policy weight and divided by the policy range to determine the overall exposure that the setting implies. The computed badness is summed across all settings in the unit file, normalized to the 1...100 range, and used to determine the overall exposure level of the unit. By allowing users to manipulate these fields, the 'security' verb gives them the option to decide for themself which ids are more important and hence should have a greater effect on the exposure level. A weight of "0" means the setting will not be checked.

```
{
  "PrivateDevices":
    {
    "description_good": "Service has no access to hardware devices",
    "description_bad": "Service potentially has access to hardware devices",
    "weight": 1000,
    "range": 1
    },
  "PrivateMounts":
    {
    "description_good": "Service cannot install system mounts",
    "description_bad": "Service may install system mounts",
    "weight": 1000,
    "range": 1
    },
  "PrivateNetwork":
    {
    "description_good": "Service has no access to the host's network",
    "description_bad": "Service has access to the host's network",
    "weight": 2500,
    "range": 1
    },
  "PrivateTmp":
    {
```

```
                "description_good": "Service has no access to other software's temporary files",
                "description_bad": "Service has access to other software's temporary files",
                "weight": 1000,
                "range": 1
                },
              "PrivateUsers":
                {
                "description_good": "Service does not have access to other users",
                "description_bad": "Service has access to other users",
                "weight": 1000,
                "range": 1
                }
            }
```

SEE ALSO

systemd(1), **systemctl**(1)