NAME

random - the entropy device

SYNOPSIS

options RANDOM_LOADABLE
options RANDOM_ENABLE_ETHER
options RANDOM_ENABLE_UMA

DESCRIPTION

The **random** device returns an endless supply of random bytes when read.

The generator will start in an *unseeded* state, and will block reads until it is seeded for the first time.

To provide prompt access to the random device at boot time, FreeBSD automatically saves some entropy data in */boot/entropy* for the loader(8) to provide to the kernel. Additional entropy is regularly saved in */var/db/entropy*. This saved entropy is sufficient to unblock the random device on devices with writeable media.

Embedded applications without writable media must determine their own scheme for re-seeding the random device on boot, or accept that the device will remain unseeded and block reads indefinitely. See *SECURITY CONSIDERATIONS* for more detail.

In addition to read(2), the direct output of the abstract kernel entropy device can be read with getrandom(2), getentropy(3), or the sysctl(8) pseudo-variable *kern.arandom*.

To see the current settings of the software **random** device, use the command line:

```
sysctl kern.random
```

which results in something like:

```
kern.random.block_seeded_status: 0
kern.random.fortuna.minpoolsize: 64
kern.random.harvest.mask_symbolic: ENABLEDSOURCE,[DISABLEDSOURCE],...,CACHED
kern.random.harvest.mask_bin: 00000010000000111011111
kern.random.harvest.mask: 66015
kern.random.use_chacha20_cipher: 0
kern.random.random_sources: 'Intel Secure Key RNG'
kern.random.initial_seeding.bypass_before_seeding: 1
kern.random.initial_seeding.read_random_bypassed_before_seeding: 0
```

kern.random.initial_seeding.arc4random_bypassed_before_seeding: 0
kern.random.initial_seeding.disable_bypass_warnings: 0

Other than *kern.random.block_seeded_status*, *kern.random.fortuna.minpoolsize*, and *kern.random.harvest.mask*, all settings are read-only via sysctl(8).

The *kern.random.fortuna.minpoolsize* sysctl is used to set the seed threshold. A smaller number gives a faster seed, but a less secure one. In practice, values between 64 and 256 are acceptable.

The *kern.random.harvest.mask* bitmask is used to select the possible entropy sources. A 0 (zero) value means the corresponding source is not considered as an entropy source. Set the bit to 1 (one) if you wish to use that source. The *kern.random.harvest.mask_bin* and *kern.random.harvest.mask_symbolic* sysctls can be used to confirm settings in a human readable form. Disabled items in the latter are listed in square brackets. See random_harvest(9) for more on the harvesting of entropy.

FILES

/dev/random

/dev/urandom

DIAGNOSTICS

The following tunables are related to initial seeding of the **random** device:

kern.random.initial_seeding.bypass_before_seeding
Defaults to 1 (on). When set, the system will bypass the **random** device prior to initial seeding. On is *unsafe*, but provides availability on many systems that lack early sources of entropy, or cannot load */boot/entropy* sufficiently early in boot for **random** consumers. When unset (0), the system will block read_random(9) and arc4random(9) requests if and until the **random** device is initially seeded.

kern.random.initial_seeding.disable_bypass_warnings
Defaults to 0 (off). When set non-zero, disables warnings in dmesg when the **random** device is bypassed.

The following read-only sysctl(8) variables allow programmatic diagnostic of whether **random** device bypass occurred during boot. If they are set (non-zero), the specific functional unit bypassed the strong **random** device output and either produced no output (read_random(9)) or seeded itself with minimal, non-cryptographic entropy (arc4random(9)).

- *kern.random.initial_seeding.read_random_bypassed_before_seeding*

- *kern.random.initial_seeding.arc4random_bypassed_before_seeding*

SEE ALSO

getrandom(2), arc4random(3), getentropy(3), random(3), sysctl(8), random(9)

Ferguson, Schneier, and Kohno, *Cryptography Engineering*, *Wiley*, ISBN 978-0-470-47424-2.

HISTORY

A **random** device appeared in FreeBSD 2.2. The implementation was changed to the *Yarrow algorithm in* FreeBSD 5.0. In FreeBSD 11.0, the Fortuna algorithm was introduced as the default. In FreeBSD 12.0, Yarrow was removed entirely.

AUTHORS

The current **random** code was authored by Mark R V Murray, with significant contributions from many people.

The *Fortuna* algorithm was designed by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno.

CAVEATS

When **options RANDOM_LOADABLE** is enabled, the */dev/random* device is not created until an "algorithm module" is loaded. The only module built by default is *random_fortuna*. Loadable random modules are less efficient than their compiled-in equivalents. This is because some functions must be locked against load and unload events, and also must be indirect calls to allow for removal.

When **options RANDOM_ENABLE_UMA** is enabled, the */dev/random* device will obtain entropy from the zone allocator. This is a very high rate source with significant performance impact. Therefore, it is disabled by default.

When **options RANDOM_ENABLE_ETHER** is enabled, the **random** device will obtain entropy from *mbuf* structures passing through the network stack. This source is both extremely expensive and a poor source of entropy, so it is disabled by default.

SECURITY CONSIDERATIONS

The initial seeding of random number generators is a bootstrapping problem that needs very careful attention. When writable media is available, the *Fortuna* paper describes a robust system for rapidly reseeding the device.

In some embedded cases, it may be difficult to find enough randomness to seed a random number generator until a system is fully operational. In these cases, is the responsibility of the system architect to ensure that blocking is acceptable, or that the random device is seeded. (This advice does not apply to

typical consumer systems.)

To emulate embedded systems, developers may set the *kern.random.block_seeded_status* tunable to 1 to verify boot does not require early availability of the **random** device.