## NAME

**KASAN** - Kernel Address SANitizer

## SYNOPSIS

The *GENERIC-KASAN* kernel configuration can be used to compile a KASAN-enabled kernel using *GENERIC* as a base configuration. Alternately, to compile KASAN into the kernel, place the following line in your kernel configuration file:

```
options KASAN
```

```
#include <sys/asan.h>
```

*void*
**kasan_mark**(*const void *addr*, *size_t size*, *size_t redzsize*, *uint8_t code*);

## DESCRIPTION

**KASAN** is a subsystem which leverages compiler instrumentation to detect invalid memory accesses in the kernel. Currently it is implemented on the amd64 and arm64 platforms.

When **KASAN** is compiled into the kernel, the compiler is configured to emit function calls upon every memory access. The functions are implemented by **KASAN** and permit run-time detection of several types of bugs including use-after-frees, double frees and frees of invalid pointers, and out-of-bounds accesses. These protections apply to memory allocated by uma(9), malloc(9) and related functions, and **kmem_malloc**() and related functions, as well as global variables and kernel stacks. **KASAN** is conservative and will not detect all instances of these types of bugs. Memory accesses through the kernel map are sanitized, but accesses via the direct map are not. When **KASAN** is configured, the kernel aims to minimize its use of the direct map.

## IMPLEMENTATION NOTES

**KASAN** is implemented using compiler instrumentation and a kernel runtime. When a kernel is built with the KASAN option enabled, the compiler inserts function calls before most memory accesses in the generated code. The runtime implements the corresponding functions, which decide whether a given access is valid. If not, the runtime prints a warning or panics the kernel, depending on the value of the **debug.kasan.panic_on_violation** sysctl/tunable.

The **KASAN** runtime works by maintaining a shadow map for the kernel map. There exists a linear mapping between addresses in the kernel map and addresses in the shadow map. The shadow map is used to store information about the current state of allocations from the kernel map. For example, when a buffer is returned by malloc(9), the corresponding region of the shadow map is marked to indicate that the buffer is valid. When it is freed, the shadow map is updated to mark the buffer as invalid. Accesses

to the buffer are intercepted by the **KASAN** runtime and validated using the contents of the shadow map.

Upon booting, all kernel memory is marked as valid. Kernel allocators must mark cached but free buffers as invalid, and must mark them valid before freeing the kernel virtual address range. This slightly reduces the effectiveness of **KASAN** but simplifies its maintenance and integration into the kernel.

Updates to the shadow map are performed by calling **kasan_mark**(). Parameter *addr* is the address of the buffer whose shadow is to be updated, *size* is the usable size of the buffer, and *redzsize* is the full size of the buffer allocated from lower layers of the system. *redzsize* must be greater than or equal to *size*. In some cases kernel allocators will return a buffer larger than that requested by the consumer; the unused space at the end is referred to as a red zone and is always marked as invalid. *code* allows the caller to specify an identifier used when marking a buffer as invalid. The identifier is included in any reports generated by **KASAN** and helps identify the source of the invalid access. For instance, when an item is freed to a uma(9) zone, the item is marked with KASAN_UMA_FREED. See <*sys/asan.h*> for the available identifiers. If the entire buffer is to be marked valid, i.e., *size* and *redzsize* are equal, *code* should be 0.

## SEE ALSO

build(7), KMSAN(9), malloc(9), memguard(9), redzone(9), uma(9)

## HISTORY

**KASAN** was ported from NetBSD and first appeared in FreeBSD 14.0.

## BUGS

Accesses to kernel memory outside of the kernel map are ignored by the **KASAN** runtime. When **KASAN** is configured, the kernel memory allocators are configured to use the kernel map, but some uses of the direct map remain. For example, on amd64 and arm64, accesses to page table pages are not tracked.

Some kernel memory allocators explicitly permit accesses after an object has been freed. These cannot be sanitized by **KASAN**. For example, memory from all uma(9) zones initialized with the UMA_ZONE_NOFREE flag are not sanitized.