Slub Memory Allocator -12- (debugging slub)

2016-06-05 (http://jake.dothome.co.kr/slub-debug/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

Debugging Slab Objects (SLUB)

SLAB_POISON

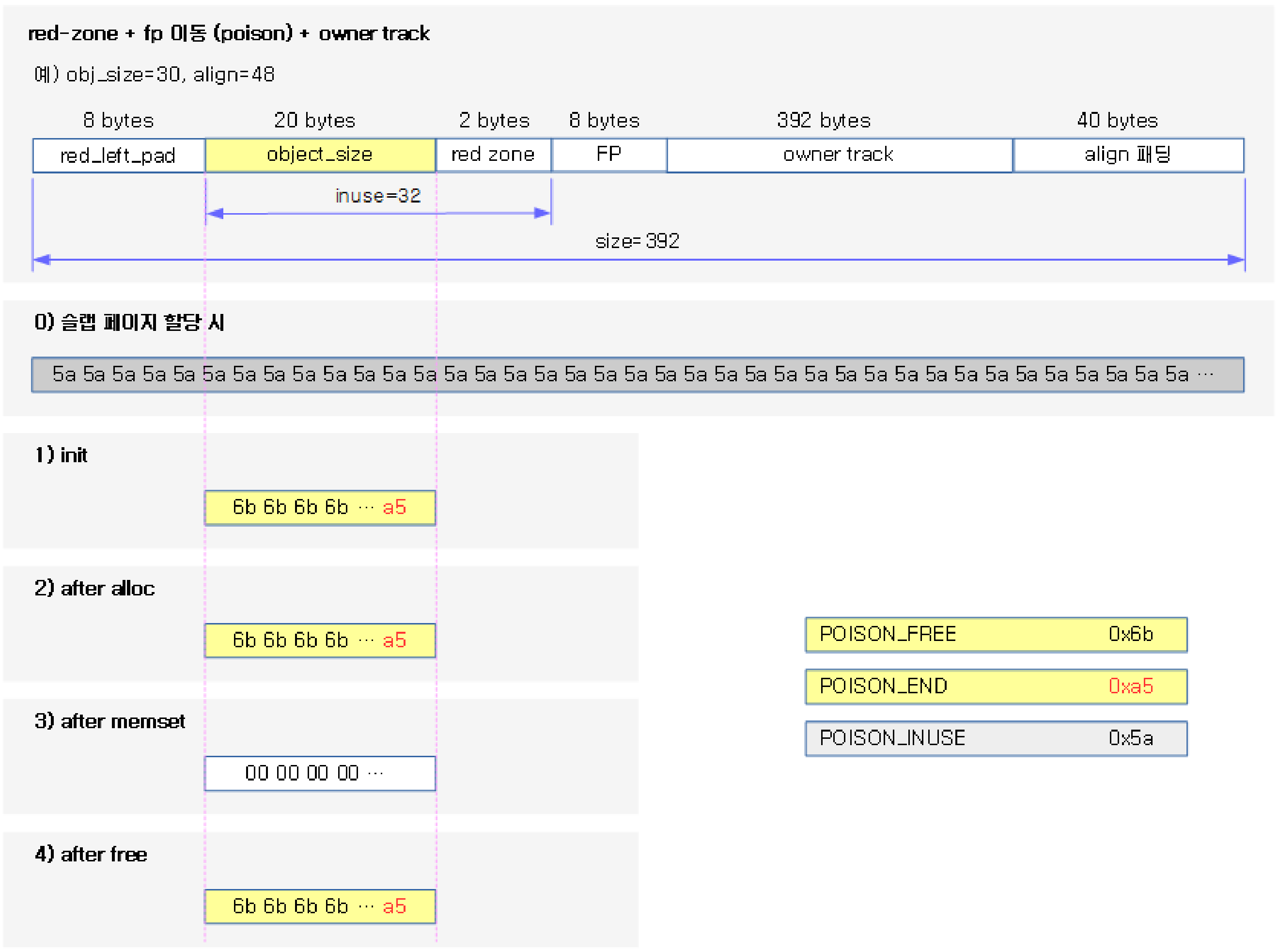
If you use SLAB_POISON to debug Slub, you can write special data to the object data area when allocating and releasing objects, and then use it to find the following errors:

- Object already free – Double Unlock
    - When it is released, poison free data (0x6b is continuous and 0xa5 at the end) is recorded in the object, and if it finds poison free data at the moment of releasing the released object again, it will know that it will try to release it repeatedly, and it will print an error.
- Poison overwritten – Use after release
    - Technically, if you use this free object in violation of it, you cannot check it. However, when assigning an object again, it sees that the poison free data of the object that was thought to be free has been destroyed, and it judges that someone has infringed and prints an error.

Kernel Parameter Settings

- "slub_debug=FP[,<slapname>]"

The following figure shows the change in the poison value of the slab object.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/slab-poison-1a.png)

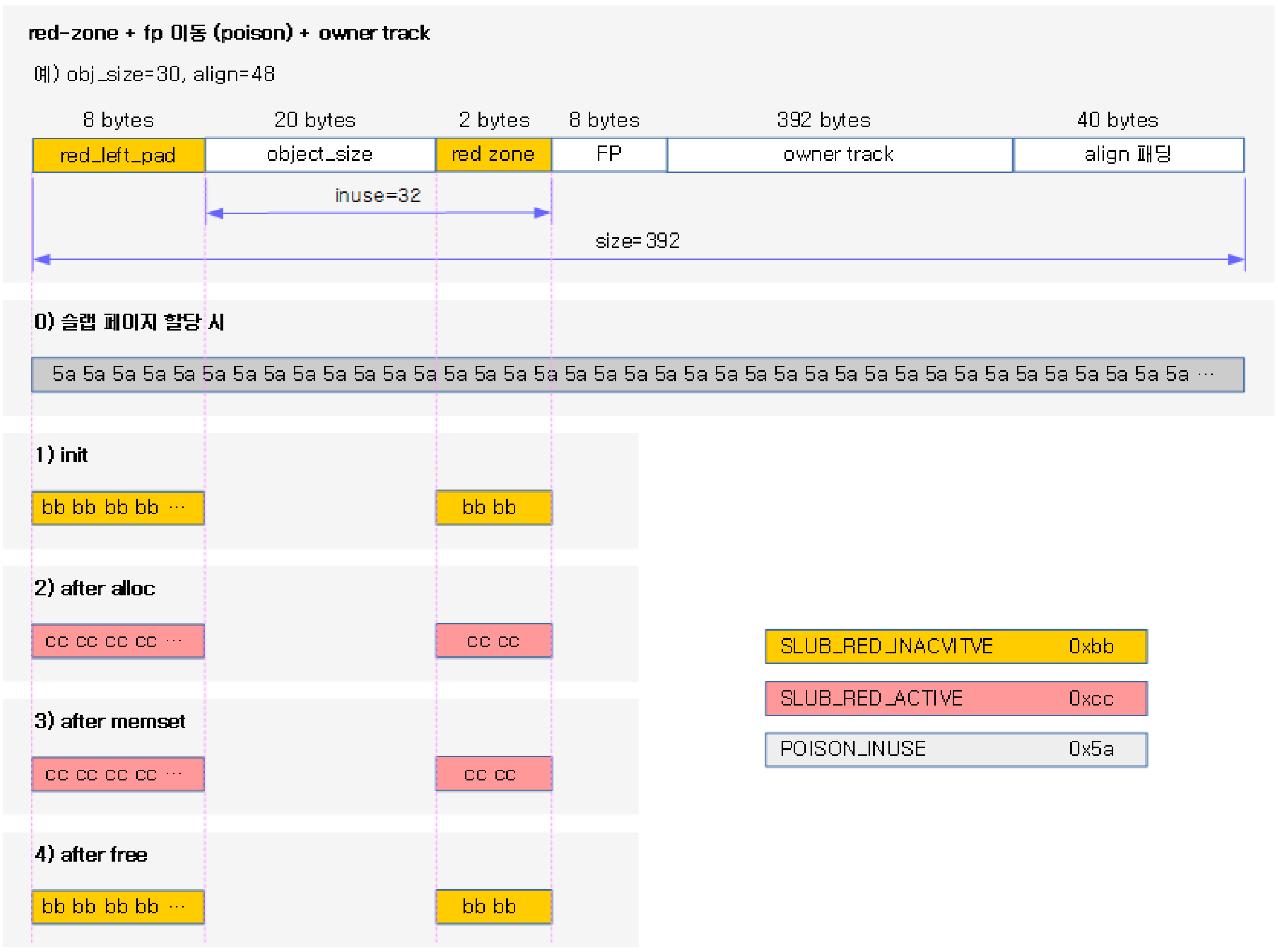
SLAB_RED_ZONE

If you use RED_ZONE for Slab debugging, you can write special data to the red zone when allocating and releasing objects, and then use it to find the following errors:

- Redzone overwritten
  - If the object data exceeds its own area and invades the red zone area, it will find out at the moment of assigning or canceling the object and output an error.

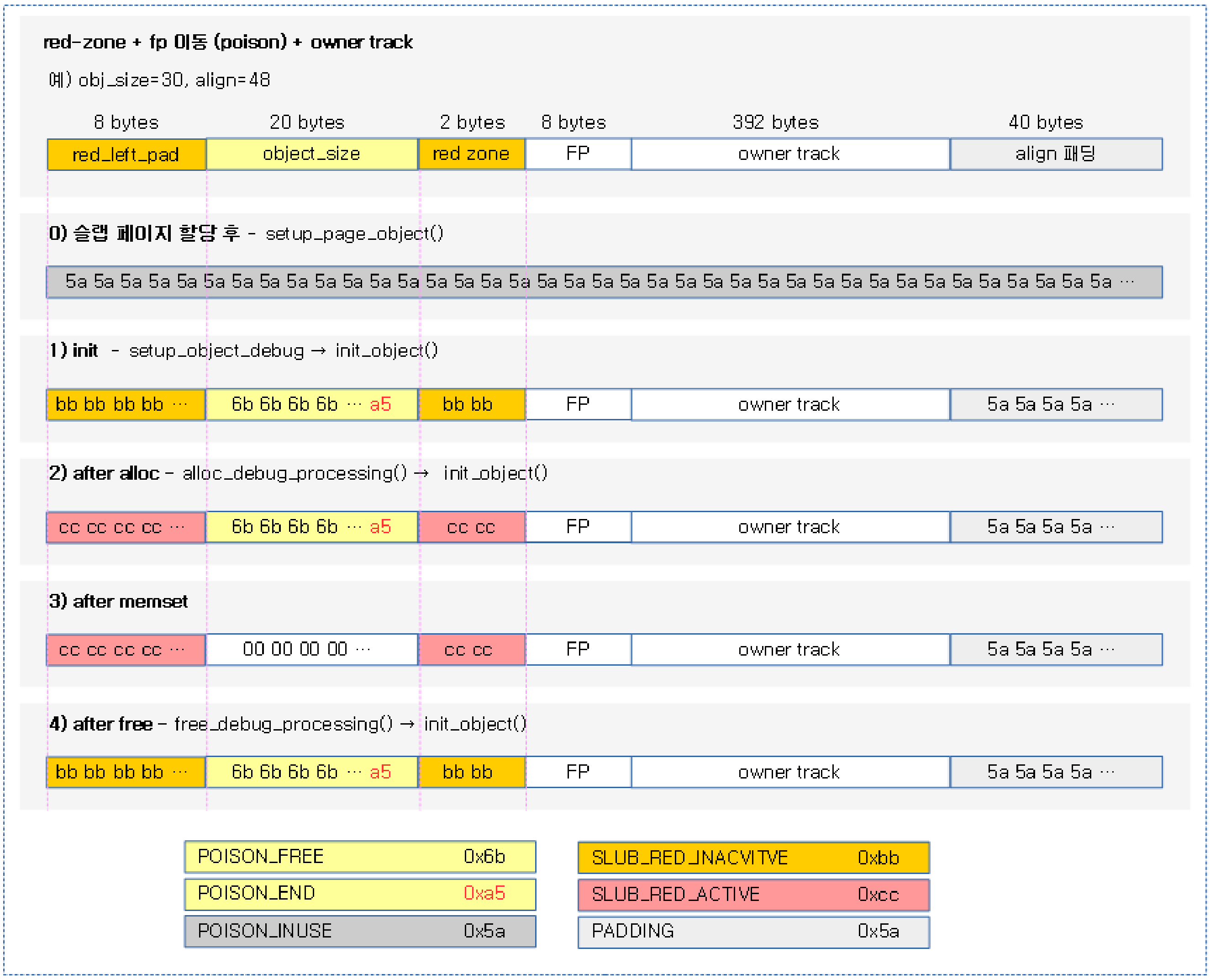
Kernel Parameter Settings

- "slub_debug=FZ[,<slapname>]"



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/slab-red-zone-1a.png)

The following figure shows the handling of values for poison and red-zone.


red-zone + fp 이동 (poison) + owner track
예) obj_size=30, align=48
8 bytes
20 bytes
2 bytes
8 bytes
392 bytes
40 bytes
red_left_pad
object_size
red zone
FP
owner track
align 패딩
0) 슬랩 페이지 할당 후 - setup_page_object()
1) init - setup_object_debug → init_object()
2) after alloc - alloc_debug_processing() → init_object()
3) after memset
4) after free - free_debug_processing() → init_object()
POISON_FREE 0x6b
POISON_END 0xa5
POISON_INUSE 0x5a
SLUB_RED_INACVITVE 0xbb
SLUB_RED_ACTIVE 0xcc
PADDING 0x5a

(http://jake.dothome.co.kr/wp-content/uploads/2016/06/slab-debug-1a.png)

Check the slab page and slab object configuration

Check if there are any problems with the configuration values, padding, and FP (Freelist) of the slab object, and report the following error.

- Object padding overwritten
  - The first new slab page is written as the initial value of poison inuse data (0x5a) before initializing the slab object for the entire area.
  - When allocating or disassigning slabs, it checks the 0x5a value recorded in the padding space and prints an error if it detects that it has changed.
- Alignment padding
  - When red-zone is not used, if s->object_size and s->inuse are different, an error is printed.
- Invalid object pointer 0x%p
  - If the pointer address is outside the slab page range, it prints an error.
- Freepointer corrupt
  - If you use a Free Pointer (FP) address that is outside the slab page range, it will print an error.
- Freechain corrupt
  - If the FP value is outside the slab page range, it will display an error.

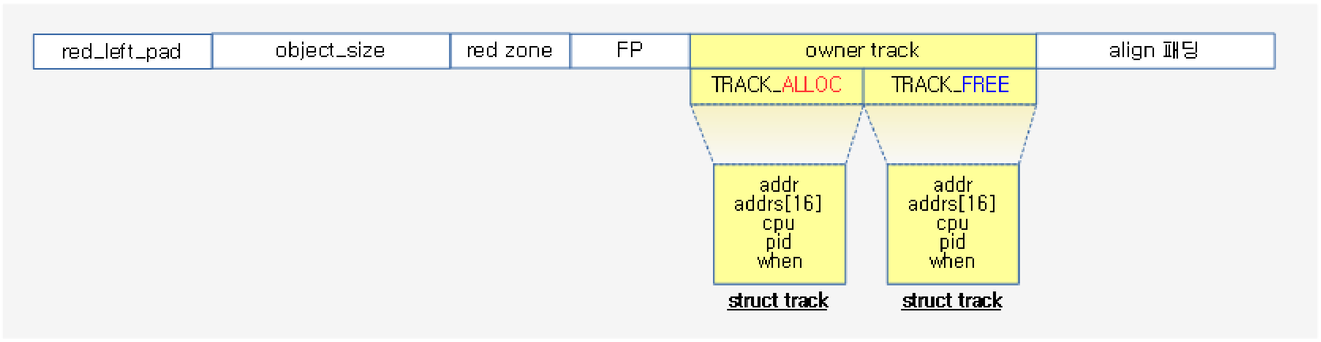
- Wrong number of objects. Found %d but should be %d
  - The maximum number of slab pages that can fit is incorrect and prints an error.
- Wrong object count. Counter is %d but counted were %d
  - If the number of objects in use on the slab page is incorrect, an error is printed.
  - If it's normal, p->objects – free object count = p->inuse.
- Not a valid slab page
  - This is the case when it is not a slap page.
- objects %u > max %u
  - The number of objects in use on the slab page is used to exceed the maximum number of slab objects that can be used, and an error is printed.
- inuse %u > max %u
  - Use the value of the number of objects being used in the slab page that exceeds the maximum number of slab objects.
- Padding overwritten. 0x%p-0x%p
  - The padding value of the unused space behind the slab page has been changed.

SLAB_STORE_USER

If you're using SLAB_STORE_USER for Slab (Slub) debugging, you can locate and unassign the object, the pid, the CPU, and the address from which it was called (function trace). You can track the time of day. If you use the CONFIG_STACKTRACE kernel options, you can track back up to 16 functions.

Kernel Parameter Settings

- "slub_debug=FU[,<slap name>]"



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/slab-store-user-1a.png)

Alloc User Tracking

The following shows the twice-allocated (alloc) in the foo slap cache that allowed user tracking.

```
$ cat /sys/kernel/slab/foo/alloc_calls
      1 0xffff000008b8a068 age=2609 pid=3481
      1 0xffff000008b8a078 age=2596 pid=3481
```

Free User Tracking

The following shows the history of once deallocated (free) from the foo slab cache that allowed user tracking.

```
$ cat /sys/kernel/slab/foo/free_calls
2 <not-available> age=4295439025 pid=0
```

TRACE

If you use SLAB_TRACE for slub debugging, you can track the allocation and release of an object by performing a dump of the object's address, inuse value, free pointer, and hexadecimal number of the object.

Kernel Parameter Settings

- "slub_debug=FT[,<slapname>]"

```
[  242.214756] TRACE jake free 0xffff800039509d20 inuse=2 fp=0xffff800039508190
[  242.217524] Object ffff800039509d20: 44 44 44 44 44 44 44 44 44 44 44 44 44 44 4
4 44  DDDDDDDDDDDDDDDD
[  242.221762] Object ffff800039509d30: 44 44 44 44 44 44 44 44 44 44 44 44 44 44
DDDDDDDDDDDDDD
```

Slap Assignment Flags

Here are some of the flags related to debug:

- SLAB_CONSISTENCY_CHECKS
  - Perform a check on the slab object for each alloc and free time. (expensive)
- SLAB_RED_ZONE
  - Debugging with Red Zone Zones
- SLAB_POISON
  - Debugging by writing poison data to the object area
- SLAB_STORE_USER
  - It stores information about the last owner for debugging purposes.
- SLAB_TRACE
  - Traces are supported for debugging.
- SLAB_DEBUG_OBJECTS
  - Prevents debugging of slab objects when they are free.
- SLAB_KASAN
  - It supports runtime debugging for slab debugging.

    - Memory consumption and performance are greatly reduced, but slab object infringement is detected immediately.
- SLAB_NOLEAKTRACE
    - kmemleak traces.
- SLAB_FAILSLAB
    - When assigning a slab object, an error is generated by fault injection.
- SLAB_PANIC
    - Call panic() on slap object error.

Debug initialization of object on slub page allocation

setup_object()

mm/slub.c

```
01 static void setup_object(struct kmem_cache *s, struct page *page,
02                                 void *object)
03 {
04         setup_object_debug(s, page, object);
05         object = kasan_init_slab_obj(s, object);
06         if (unlikely(s->ctor)) {
07                 kasan_unpoison_object_data(s, object);
08                 s->ctor(object);
09                 kasan_poison_object_data(s, object);
10         }
11 }
```

Initialize the Slub object for kernel options debugging and runtime memory debugging.

- LINE 4 OF CODE INSTALLS POISON, RED ZONE, AND TRACKING DATA FOR SLAP DEBUGGING.
- In line 5 of code, we initialize the shadow area to 0 for the runtime memory debugger.
- In line 6~10 of code, if the constructor is a prepared slab cache, the slab object constructor will be executed.

setup_object_debug()

mm/slub.c

```
01 /* Object debug checks for alloc/free paths */
02 static void setup_object_debug(struct kmem_cache *s, struct page *page,
03                                                                 void *ob
   ject)
04 {
05         if (!(s->flags & (SLAB_STORE_USER|SLAB_RED_ZONE|__OBJECT_POISO
   N)))
06                 return;
07 
08         init_object(s, object, SLUB_RED_INACTIVE);
09         init_tracking(s, object);
10 }
```

Install POISON, RED ZONE, and tracking data for slub debugging.

- If there is no slub debugging flag in line 5~6 of the code, exit the function.

- LINE 8 OF CODE INSTALLS THE POISON DATA AND THE RED ZONE DATA.
- In line 9 of the code, we initialize the data for alloc/free user tracking.

init_object()

mm/slub.c

```
01 static void init_object(struct kmem_cache *s, void *object, u8 val)
02 {
03         u8 *p = object;
04         if (s->flags & SLAB_RED_ZONE)
05                 memset(p - s->red_left_pad, val, s->red_left_pad);
06         if (s->flags & __OBJECT_POISON) {
07                 memset(p, POISON_FREE, s->object_size - 1);
08                 p[s->object_size - 1] = POISON_END;
09         }
10 
11         if (s->flags & SLAB_RED_ZONE)
12                 memset(p + s->object_size, val, s->inuse - s->object_size);
13 }
```

When initializing a slub object, specify @val values for poison_free (0x6b) and red-zone.

- In line 4~5 of the code, fill in the red-zone area on the left with the @val value.
- In line 6~9 of the code, fill the object with the value of poison_free(0x6b).
- In lines 11~12 of code, fill in the red-zone area to the right of the object with @val values.

init_tracking()

mm/slub.c

```
static void init_tracking(struct kmem_cache *s, void *object)
{
        if (!(s->flags & SLAB_STORE_USER))
                return;

        set_track(s, object, TRACK_FREE, 0UL);
        set_track(s, object, TRACK_ALLOC, 0UL);
}
```

For alloc/free user tracking, the data in two areas, TRACK_FREE and TRACK_ALLOC, is reset to zero.

set_track()

mm/slub.c

```
static void set_track(struct kmem_cache *s, void *object,
                        enum track_item alloc, unsigned long addr)
{
        struct track *p = get_track(s, object, alloc);

        if (addr) {
#ifdef CONFIG_STACKTRACE
                struct stack_trace trace;
                int i;

                trace.nr_entries = 0;
```

```
		trace.max_entries = TRACK_ADDRS_COUNT;
		trace.entries = p->addrs;
		trace.skip = 3;
		metadata_access_enable();
		save_stack_trace(&trace);
		metadata_access_disable();

		/* See rant in lockdep.c */
		if (trace.nr_entries != 0 &&
		    trace.entries[trace.nr_entries - 1] == ULONG_MAX)
			trace.nr_entries--;

		for (i = trace.nr_entries; i < TRACK_ADDRS_COUNT; i++)
			p->addrs[i] = 0;
#endif
		p->addr = addr;
		p->cpu = smp_processor_id();
		p->pid = current->pid;
		p->when = jiffies;
	} else
		memset(p, 0, sizeof(struct track));
}
```

It stores data about the call address, CPU, PID, and time for tracking.

- If you use the CONFIG_STACKTRACE kernel options, you can trace back up to 16 functions in slub.

Debug check when assigning slab object

It is called when a slab object is allocated and checked.

alloc_debug_processing()

mm/slub.c

```
static noinline int alloc_debug_processing(struct kmem_cache *s,
					struct page *page,
					void *object, unsigned long addr)
{
	if (s->flags & SLAB_CONSISTENCY_CHECKS) {
		if (!alloc_consistency_checks(s, page, object, addr))
			goto bad;
	}

	/* Success perform special debug activities for allocs */
	if (s->flags & SLAB_STORE_USER)
		set_track(s, object, TRACK_ALLOC, addr);
	trace(s, page, object, 1);
	init_object(s, object, SLUB_RED_ACTIVE);
	return 1;

bad:
	if (PageSlab(page)) {
		/*
		 * If this is a slab page then lets do the best we can
		 * to avoid issues in the future. Marking all objects
		 * as used avoids touching the remaining objects.
		 */
		slab_fix(s, "Marking all objects used");
		page->inuse = page->objects;
		page->freelist = NULL;
	}
28          return 0;
29  }
```

When assigning a slab object, perform a debug check. Returns 1 on success

- In line 5~8 of the code, perform a sanity check of the slab cache specified as "slub_debug=F,<slapcache name>".
- In line 11~12 of the code, record the owner function of the allocation request for the object of the slab cache specified as "slub_debug=FU,<slapcache name>".
- Perform a trace of the slab cache specified in line 13 as "slub_debug=FT,<slapcachename>".
  - "TRACE <slabname> alloc <object address> inuse=xx fp=<FP address>" is printed, and the hexadecimal values of "Object" are dumped.
- In line 14 of code, initialize the Slub object, specifying 0xcc SLUB_RED_ACTIVE values for poison_free (6x0b) and red-zone.
- Returns the normal value of 15 on line 1 of code.
- On line 17 of the code, the bad: label is entered when the sanity check fails.
- In line 18~28 of the code, to stop using all the free objects on the slab page, mark them as busy and return 0 failures.

Consistency check when assigning slab objects

alloc_consistency_checks()

mm/slub.c

```
01  static inline int alloc_consistency_checks(struct kmem_cache *s,
02                                          struct page *page,
03                                          void *object, unsigned long addr)
04  {
05          if (!check_slab(s, page))
06                  return 0;
07
08          if (!check_valid_pointer(s, page, object)) {
09                  object_err(s, page, object, "Freelist Pointer check fails");
10                  return 0;
11          }
12
13          if (!check_object(s, page, object, SLUB_RED_INACTIVE))
14                  return 0;
15
16          return 1;
17  }
```

@object Perform a consistency check before allocating. Returns 1 when normal

- In line 5~6 of the code, check the total number of objects in the slab page and the number of objects in use.
- In line 8~11 of the code, investigate whether the @object virtual address falls within the scope of the slab page.
- In line 13~14 of the code, check that the poison value (0x6b) and red-zone value (SLUB_RED_INACTIVE-0xbb) are well recorded before assigning the slab object.
- On line 16 of the code, the slab object is OK and returns 1.

Debug check on deallocating slab object

It is called when deallocating a slab object and checks it.

free_debug_processing()

mm/slub.c

```c
/* Supports checking bulk free of a constructed freelist */
static noinline int free_debug_processing(
        struct kmem_cache *s, struct page *page,
        void *head, void *tail, int bulk_cnt,
        unsigned long addr)
{
        struct kmem_cache_node *n = get_node(s, page_to_nid(page));
        void *object = head;
        int cnt = 0;
        unsigned long uninitialized_var(flags);
        int ret = 0;

        spin_lock_irqsave(&n->list_lock, flags);
        slab_lock(page);

        if (s->flags & SLAB_CONSISTENCY_CHECKS) {
                if (!check_slab(s, page))
                        goto out;
        }

next_object:
        cnt++;

        if (s->flags & SLAB_CONSISTENCY_CHECKS) {
                if (!free_consistency_checks(s, page, object, addr))
                        goto out;
        }

        if (s->flags & SLAB_STORE_USER)
                set_track(s, object, TRACK_FREE, addr);
        trace(s, page, object, 0);
        /* Freepointer not overwritten by init_object(), SLAB_POISON moved it */
        init_object(s, object, SLUB_RED_INACTIVE);

        /* Reached end of constructed freelist yet? */
        if (object != tail) {
                object = get_freepointer(s, object);
                goto next_object;
        }
        ret = 1;

out:
        if (cnt != bulk_cnt)
                slab_err(s, page, "Bulk freelist count(%d) invalid(%d)\n",
                         bulk_cnt, cnt);

        slab_unlock(page);
        spin_unlock_irqrestore(&n->list_lock, flags);
        if (!ret)
                slab_fix(s, "Object at 0x%p not freed", object);
        return ret;
}
```

Perform a debug check when deallocating a slab object. Returns 1 on success

- In line 16~19 of the code, perform a sanity check on the corresponding slab page of the slab cache specified as "slub_debug=F,<slapcache name>".
- In lines 21~22 of the code, it is a repeated next_object to check the entire free object: label.
- In lines 24~27 of code, perform an alloc sanity check on the traversing object in the slab cache specified as "slub_debug=F,<slapcache name>".
- In lines 29~30 of code, record the owner function to request the release of the object of the slab cache specified as "slub_debug=FU,<slapcache name>".
- Perform a trace of the slab cache specified in line 31 as "slub_debug=FT,<slapcachename>".
  - "TRACE <slabname> free <object address> inuse=xx fp=<FP address>" is printed, and the hexadecimal values of "Object" are dumped.
- In line 33 of code, initialize the Slub object, specifying 0xbb SLUB_RED_INACTIVE values for poison_free (6x0b) and red-zone.
- In code lines 36~39, go to next_object: label for traversal to @tail object.
- If completed on line 40 of the code, be prepared to return the normal value of 1.
- This is the out: label that will be used when exiting the function at line 42 of code.
- If the number of objects and @bulk_cnt from code lines 43~45 to @head ~ @tail is different, an error message will be printed.
- If an error is found in line 49~50 of the code, it will print an error message saying that the object is not free.

Check consistency when deallocating slab object

free_consistency_checks()

mm/slub.c

```
01 static inline int free_consistency_checks(struct kmem_cache *s,
02                 struct page *page, void *object, unsigned long addr)
03 {
04         if (!check_valid_pointer(s, page, object)) {
05                 slab_err(s, page, "Invalid object pointer 0x%p", object);
06                 return 0;
07         }
08 
09         if (on_freelist(s, page, object)) {
10                 object_err(s, page, object, "Object already free");
11                 return 0;
12         }
13 
14         if (!check_object(s, page, object, SLUB_RED_ACTIVE))
15                 return 0;
16 
17         if (unlikely(s != page->slab_cache)) {
18                 if (!PageSlab(page)) {
19                         slab_err(s, page, "Attempt to free object(0x%p) outside of slab",
20                                  object);
21                 } else if (!page->slab_cache) {
22                         pr_err("SLUB <none>: no slab for object 0x%p.\n",
23                                object);
24                         dump_stack();
25                 } else
26                         object_err(s, page, object,
27                                         "page slab pointer corrupt.");
28                 return 0;
29         }
30         return 1;
31 }
```

@object Perform a consistency check before deallocating. Returns 1 when normal

- In line 4~7 of the code, investigate whether the @object virtual address falls within the scope of the slab page.
- Search the freelist chain of the slab page requested in line 9~12 of the code to @object and check whether the FP(Free Pointer) value of each free object is valid.
- In line 14~15 of the code, check that the red-zone values (SLUB_RED_ACTIVE-0xcc) are well recorded before deallocating the slab object. Also, if you're not using red-zone and only poison, make sure that the poison value (0x6b) is properly recorded in the padding position after the object.
- In lines 17~29 of the code, if the srep page does not point to the slab cache @s, it prints an error and returns 0.
- On line 30 of the code, it returns 1 because it's normal.

Common Check Functions

Slap Check

check_slab()

mm/slub.c

```
static int check_slab(struct kmem_cache *s, struct page *page)
{
        int maxobj;

        VM_BUG_ON(!irqs_disabled());

        if (!PageSlab(page)) {
                slab_err(s, page, "Not a valid slab page");
                return 0;
        }

        maxobj = order_objects(compound_order(page), s->size);
        if (page->objects > maxobj) {
                slab_err(s, page, "objects %u > max %u",
                        page->objects, maxobj);
                return 0;
        }
        if (page->inuse > page->objects) {
                slab_err(s, page, "inuse %u > max %u",
                        page->inuse, page->objects);
                return 0;
        }
        /* Slab_pad_check fixes things up after itself */
        slab_pad_check(s, page);
        return 1;
}
```

Check the total number of objects on the slab page and the number of objects in use. It also checks and corrects unused padding areas on the slab page. Returns 1 on success

- If the requested page in line 7~10 is not a slab page, it returns 0 failure.
- If the maximum number of objects is specified incorrectly in lines 12~17, it will print an error and return 0.
- In line 18~22 of the code, if the number of objects in use exceeds the maximum number of objects, it will also print an error and return 0.
- In line 24 of the code, check the unused padding area of the slab page to print and correct the error where the poison value was incorrectly recorded.
- Success returns 25 on line 1 of code.

Pad Area Check

slab_pad_check()

mm/slub.c

```
01 /* Check the pad bytes at the end of a slab page */
02 static int slab_pad_check(struct kmem_cache *s, struct page *page)
03 {
04         u8 *start;
05         u8 *fault;
06         u8 *end;
07         u8 *pad;
08         int length;
09         int remainder;
10 
11         if (!(s->flags & SLAB_POISON))
12                 return 1;
13 
14         start = page_address(page);
15         length = PAGE_SIZE << compound_order(page);
16         end = start + length;
17         remainder = length % s->size;
18         if (!remainder)
19                 return 1;
20 
21         pad = end - remainder;
22         metadata_access_enable();
23         fault = memchr_inv(pad, POISON_INUSE, remainder);
24         metadata_access_disable();
25         if (!fault)
26                 return 1;
27         while (end > fault && end[-1] == POISON_INUSE)
28                 end--;
29 
30         slab_err(s, page, "Padding overwritten. 0x%p-0x%p", fault, end - 1);
31         print_section(KERN_ERR, "Padding ", pad, remainder);
32 
33         restore_bytes(s, "slab padding", POISON_INUSE, fault, end);
34         return 0;
35 }
```

Check the unused padding area on the slap page. Returns 1 on success

- If you don't use poison in lines 11~12 of the code, you don't need to check the poison value, so it returns success 1.

- If there is no area left in the slab page on code lines 14~19, it returns success 1.
- If the POISON_INUSE(21x26a) value is well recorded in the padding area left in code lines 0~5, success returns 1.
- Find the address of the failed location in line 27~34 and print a "Padding overwritten" error, rewrite the poison value of the wrong part, and return 0 failure.

Object Check

check_object()

mm/slub.c

```
01 static int check_object(struct kmem_cache *s, struct page *page,
02                                         void *object, u8 val)
03 {
04         u8 *p = object;
05         u8 *endobject = object + s->object_size;
06 
07         if (s->flags & SLAB_RED_ZONE) {
08                 if (!check_bytes_and_report(s, page, object, "Redzone",
09                         object - s->red_left_pad, val, s->red_left_pad))
10                         return 0;
11 
12                 if (!check_bytes_and_report(s, page, object, "Redzone",
13                         endobject, val, s->inuse - s->object_size))
14                         return 0;
15         } else {
16                 if ((s->flags & SLAB_POISON) && s->object_size < s->inuse) {
17                         check_bytes_and_report(s, page, p, "Alignment padding",
18                                 endobject, POISON_INUSE,
19                                 s->inuse - s->object_size);
20                 }
21         }
22 
23         if (s->flags & SLAB_POISON) {
24                 if (val != SLUB_RED_ACTIVE && (s->flags & __OBJECT_POISON) &&
25                         (!check_bytes_and_report(s, page, p, "Poison", p,
26                                         POISON_FREE, s->object_size - 1) ||
27                          !check_bytes_and_report(s, page, p, "Poison",
28                                 p + s->object_size - 1, POISON_END, 1)))
29                         return 0;
30                 /*
31                  * check_pad_bytes cleans up on its own.
32                  */
33                 check_pad_bytes(s, page, p);
34         }
35 
36         if (!s->offset && val == SLUB_RED_ACTIVE)
37                 /*
38                  * Object and freepointer overlap. Cannot check
39                  * freepointer while object is allocated.
40                  */
41                 return 1;
42 
43         /* Check free pointer validity */
44         if (!check_valid_pointer(s, page, get_freepointer(s, p))) {
45                 object_err(s, page, p, "Freepointer corrupt");
46                 /*
47                      * No choice but to zap it and thus lose the remainder
48                      * of the free objects in this slab. May cause
49                      * another error because the object count is now wrong.
50                      */
51                     set_freepointer(s, p, NULL);
52                     return 0;
53             }
54             return 1;
55     }
```

Before allocating/unallocating a slab object, make sure that the poison and red-zone values are properly recorded. Returns 1 on success

- In lines 7~14 of the code, check if the two red-zone areas are filled with @val values.
- In code lines 15~21, if you don't use red-zone, but use poison, make sure that the POISON_INUSE (0x5a) value is properly recorded in the alignment padding area behind the object.
- If you use poison in lines 23~34 of the code, make sure that the padding area is clearly written in POISON_INUSE (0x5a). Also, if you enter for object assignment, make sure that the object area is filled with POISON_FREE (0x6b) values. However, the last byte must contain POISON_END (0xa5).
- In code lines 36~41, there is no reason for the FP movement, so the offset pointing to the FP is 0, and if it enters for deallocation, it returns normal 1.
- In line 44~53 of the code, check if the FP(Free Pointer) value pointing to the next object uses the address range of the slab page, and if there is a problem, log a null value, and return 0 failure.
- Success returns 54 on line 1 of code.

check_pad_bytes()

mm/slub.c

```
/*
 * Object layout:
 *
 * object address
 *      Bytes of the object to be managed.
 *      If the freepointer may overlay the object then the free
 *      pointer is the first word of the object.
 *
 *      Poisoning uses 0x6b (POISON_FREE) and the last byte is
 *      0xa5 (POISON_END)
 *
 * object + s->object_size
 *      Padding to reach word boundary. This is also used for Redzoning.
 *      Padding is extended by another word if Redzoning is enabled and
 *      object_size == inuse.
 *
 *      We fill with 0xbb (RED_INACTIVE) for inactive objects and with
 *      0xcc (RED_ACTIVE) for objects in use.
 *
 * object + s->inuse
 *      Meta data starts here.
 *
 *      A. Free pointer (if we cannot overwrite object on free)
 *      B. Tracking data for SLAB_STORE_USER
 *      C. Padding to reach required alignment boundary or at mininum
 *              one word if debugging is on to be able to detect writes
 *              before the word boundary.
 *
 *      Padding is done using 0x5a (POISON_INUSE)
 *
 * object + s->size
 *      Nothing is used beyond s->size.
 *
 * If slabcaches are merged then the object_size and inuse boundaries are mostly
 * ignored. And therefore no slab options that rely on these boundaries
 * may be used with merged slabcaches.
 */
```

```
01 static int check_pad_bytes(struct kmem_cache *s, struct page *page, u8 *p)
02 {
03         unsigned long off = s->inuse;   /* The end of info */
04 
05         if (s->offset)
06                 /* Freepointer is placed after the object. */
07                 off += sizeof(void *);
08 
09         if (s->flags & SLAB_STORE_USER)
10                 /* We also have user information there */
11                 off += 2 * sizeof(struct track);
12 
13         off += kasan_metadata_size(s);
14 
15         if (size_from_object(s) == off)
16                 return 1;
17 
18         return check_bytes_and_report(s, page, p, "Object padding",
19                         p + off, POISON_INUSE, size_from_object(s) - off);
20 }
```

Check the poison values in the padding area of the alignment after the object. Returns 1 on success

- In line 3 of code, the off value is prepared to calculate the padding position, first specifying an object and an s->inuse value that includes the object-specific alignment value.
- If the position value of the FP (Free Pointer) pointing to the free object in line 5~7 of the code is given, it goes beyond that area.
- If you use user tracking in line 9~11 of the code, it goes beyond that area.
- In line 13 of the code, it goes beyond the area used by KASAN.
- If it is aligned exactly in lines 15~16 and does not require the last padding, it returns success 1.
- In lines 18~19 of the code, check that the last alignment padding area is filled with a POISON_INUSE(0x5a) value.

check_bytes_and_report()

mm/slub.c

```
static int check_bytes_and_report(struct kmem_cache *s, struct page *page,
                        u8 *object, char *what,
                        u8 *start, unsigned int value, unsigned int bytes)
{
        u8 *fault;
        u8 *end;

        metadata_access_enable();
        fault = memchr_inv(start, value, bytes);
        metadata_access_disable();
        if (!fault)
                return 1;

        end = start + bytes;
        while (end > fault && end[-1] == value)
                end--;

        slab_bug(s, "%s overwritten", what);
        pr_err("INFO: 0x%p-0x%p. First byte 0x%x instead of 0x%x\n",
                                        fault, end - 1, fault[0], value);
        print_trailer(s, page, object);

        restore_bytes(s, what, value, fault, end);
        return 0;
}
```

Check if the @value value is recorded @bytes the @start address. It returns 1 on success, prints an error message if there is a problem, dumps it, and returns 0. It is used to display the following error name:

- "Object padding overwritten"
- "Redzone overwritten"
- "Alignment padding overwritten"
- "Poison overwritten"

Freelist Check

on_freelist()

mm/slub.c

```
/*
 * Determine if a certain object on a page is on the freelist. Must hold the
 * slab lock to guarantee that the chains are in a consistent state.
 */
```

```
static int on_freelist(struct kmem_cache *s, struct page *page, void *search)
{
        int nr = 0;
        void *fp;
        void *object = NULL;
        int max_objects;

        fp = page->freelist;
        while (fp && nr <= page->objects) {
                if (fp == search)
                        return 1;
                if (!check_valid_pointer(s, page, fp)) {
                        if (object) {
                                object_err(s, page, object,
                                        "Freechain corrupt");
                                set_freepointer(s, object, NULL);
                        } else {
                                slab_err(s, page, "Freepointer corrupt");
                                page->freelist = NULL;
                                page->inuse = page->objects;
                                slab_fix(s, "Freelist cleared");
                                return 0;
                        }
                        break;
25                 }
26                 object = fp;
27                 fp = get_freepointer(s, object);
28                 nr++;
29         }
30 
31         max_objects = order_objects(compound_order(page), s->size);
32         if (max_objects > MAX_OBJS_PER_PAGE)
33                 max_objects = MAX_OBJS_PER_PAGE;
34 
35         if (page->objects != max_objects) {
36                 slab_err(s, page, "Wrong number of objects. Found %d but should be %d",
37                          page->objects, max_objects);
38                 page->objects = max_objects;
39                 slab_fix(s, "Number of objects adjusted.");
40         }
41         if (page->inuse != page->objects - nr) {
42                 slab_err(s, page, "Wrong object count. Counter is %d but counted were %d",
43                          page->inuse, page->objects - nr);
44                 page->inuse = page->objects - nr;
45                 slab_fix(s, "Object count adjusted.");
46         }
47         return search == NULL;
48 }
```

Search for @search free objects in the freelist chain of the requested slab page and check whether the FP (Free Pointer) value of each free object is valid. Returns 1 for success, or 0 for failure if no @search object is found or if there is a problem with the FP value. (If @search is null, search until the end)

- In line 8~29 of the code, we @search all the free objects connected to the freelist of the slab page and check the validity of the FP(Free Pointer) values until the object is found. If @search object is found, it returns a healthy 1. If there is a problem, disconnect the connection after the problematic free object and display a "Freechain corrupt" error.
- In lines 31~40 of the code, check if the number of objects that can be included in the order page is the same as s->objects, and if not, "Wrong number of objects. Found %d but should be %d", and corrects the value of s->objects.
- In lines 41~46 of the code, even if the number of objects in use is different, you will get "Wrong object count. Counter is %d but counted were %d", and corrects the s->inuse value.
- Returns Success 47 only if the input argument @search entered null on line 1, and Failure 0 if the @search object is specified but not found.

check_valid_pointer()

mm/slub.c

```
01 /* Verify that a pointer has an address that is valid within a slab page */
02 static inline int check_valid_pointer(struct kmem_cache *s,
03                                 struct page *page, void *object)
04 {
05         void *base;
06 
07         if (!object)
08                 return 1;
09 
10         base = page_address(page);
11         object = kasan_reset_tag(object);
```

```
        object = restore_red_left(s, object);
        if (object < base || object >= base + page->objects * s->size ||
                (object - base) % s->size) {
                return 0;
        }

        return 1;
}
```

@object Investigate whether the virtual address falls within the scope of the slab page. Returns 1 when normal

Debugging Cases

Object already free

It finds out when you free the slab object twice.

```
[ 2202.358934] =====================================================================
=========
[ 2202.362125] BUG jake (Tainted: G    B   W  O     ): Object already free
[ 2202.365019] ---------------------------------------------------------------------
---------
[ 2202.365019]
[ 2202.369155] INFO: Allocated in foo_init+0x54/0x1000 [poison] age=131 cpu=0 pid=6
663
[ 2202.372442]  __slab_alloc+0x24/0x34
[ 2202.374002]  kmem_cache_alloc+0x1b8/0x264
[ 2202.375754]  foo_init+0x54/0x1000 [poison]
[ 2202.377567]  do_one_initcall+0x48/0x160
[ 2202.379247]  do_init_module+0x54/0x1c4
[ 2202.380937]  load_module+0x1ec8/0x20ac
[ 2202.382578]  __se_sys_finit_module+0xc8/0xdc
[ 2202.384429]  __arm64_sys_finit_module+0x18/0x20
[ 2202.386411]  el0_svc_common+0xc0/0x100
[ 2202.388060]  el0_svc_handler+0x2c/0x70
[ 2202.389726]  el0_svc+0x8/0xc
[ 2202.390985] INFO: Freed in foo_init+0xfc/0x1000 [poison] age=12 cpu=0 pid=6663
[ 2202.394121]  kmem_cache_free+0x220/0x230
[ 2202.395815]  foo_init+0xfc/0x1000 [poison]
[ 2202.397603]  do_one_initcall+0x48/0x160
[ 2202.399291]  do_init_module+0x54/0x1c4
[ 2202.400931]  load_module+0x1ec8/0x20ac
[ 2202.402580]  __se_sys_finit_module+0xc8/0xdc
[ 2202.404422]  __arm64_sys_finit_module+0x18/0x20
[ 2202.406395]  el0_svc_common+0xc0/0x100
[ 2202.408001]  el0_svc_handler+0x2c/0x70
[ 2202.409637]  el0_svc+0x8/0xc
[ 2202.410917] INFO: Slab 0xffff7e0000e5cf00 objects=20 used=0 fp=0xffff80003973c00
8 flags=0xfffc00000010201
[ 2202.415035] INFO: Object 0xffff80003973c008 @offset=8 fp=0xffff80003973dd20
[ 2202.415035]
[ 2202.418671] Redzone ffff80003973c000: bb bb bb bb bb bb bb bb
........
[ 2202.422397] Object ffff80003973c008: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6
b 6b  kkkkkkkkkkkkkkkk
[ 2202.426349] Object ffff80003973c018: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5
kkkkkkkkkkkkk.
[ 2202.430291] Redzone ffff80003973c026: bb bb
..
[ 2202.433803] Padding ffff80003973c160: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a  ZZZZZZZZZZZZZZZZ
[ 2202.437785] Padding ffff80003973c170: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a  ZZZZZZZZZZZZZZZZ
[ 2202.441817] Padding ffff80003973c180: 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZ
[ 2202.445556] CPU: 0 PID: 6663 Comm: insmod Tainted: G    B   W  O      5.0.0+ #24
[ 2202.448676] Hardware name: linux,dummy-virt (DT)
[ 2202.450626] Call trace:
[ 2202.451732]  dump_backtrace+0x0/0x154
[ 2202.453299]  show_stack+0x14/0x1c
[ 2202.454739]  dump_stack+0x88/0xb0
[ 2202.456150]  print_trailer+0x108/0x194
[ 2202.457787]  object_err+0x3c/0x4c
[ 2202.459198]  free_debug_processing+0x278/0x484
[ 2202.461125]  __slab_free+0x74/0x4b0
[ 2202.462602]  kmem_cache_free+0x220/0x230
[ 2202.464400]  foo_init+0x108/0x1000 [poison]
[ 2202.466332]  do_one_initcall+0x48/0x160
[ 2202.468010]  do_init_module+0x54/0x1c4
[ 2202.469673]  load_module+0x1ec8/0x20ac
[ 2202.471324]  __se_sys_finit_module+0xc8/0xdc
[ 2202.473174]  __arm64_sys_finit_module+0x18/0x20
[ 2202.475137]  el0_svc_common+0xc0/0x100
[ 2202.476783]  el0_svc_handler+0x2c/0x70
[ 2202.478387]  el0_svc+0x8/0xc
[ 2202.484819] FIX jake: Object at 0xffff80003973c008 not freed
```

Poison overwritten

One byte was written to a 32-byte object that was freed.

- After freeing from foo_init+0x58/0x1000, the data is written to find out that there was a problem when the slab object is alloced, and output.

```
[ 1774.452218] =============================================================================
[ 1774.455625] BUG jake (Tainted: G    B   W  O     ): Poison overwritten
[ 1774.458420] -----------------------------------------------------------------------------
[ 1774.458420]
[ 1774.462528] INFO: 0xffff8000364ae008-0xffff8000364ae008. First byte 0x11 instead of 0x6b
[ 1774.465971] INFO: Allocated in foo_init+0x54/0x1000 [poison] age=132 cpu=0 pid=5940
[ 1774.469231]  __slab_alloc+0x24/0x34
[ 1774.470709]  kmem_cache_alloc+0x1b8/0x264
[ 1774.472424]  foo_init+0x54/0x1000 [poison]
[ 1774.474216]  do_one_initcall+0x48/0x160
[ 1774.475892]  do_init_module+0x54/0x1c4
[ 1774.477545]  load_module+0x1ec8/0x20ac
[ 1774.479150]  __se_sys_finit_module+0xc8/0xdc
[ 1774.480982]  __arm64_sys_finit_module+0x18/0x20
[ 1774.482968]  el0_svc_common+0xc0/0x100
[ 1774.484590]  el0_svc_handler+0x2c/0x70
[ 1774.486226]  el0_svc+0x8/0xc
[ 1774.487470] INFO: Freed in foo_init+0xfc/0x1000 [poison] age=11 cpu=0 pid=5940
[ 1774.490571]  kmem_cache_free+0x220/0x230
[ 1774.492256]  foo_init+0xfc/0x1000 [poison]
[ 1774.494028]  do_one_initcall+0x48/0x160
[ 1774.495697]  do_init_module+0x54/0x1c4
[ 1774.497343]  load_module+0x1ec8/0x20ac
[ 1774.498957]  __se_sys_finit_module+0xc8/0xdc
[ 1774.500790]  __arm64_sys_finit_module+0x18/0x20
[ 1774.502726]  el0_svc_common+0xc0/0x100
[ 1774.504325]  el0_svc_handler+0x2c/0x70
[ 1774.505964]  el0_svc+0x8/0xc
[ 1774.507206] INFO: Slab 0xffff7e0000d92b80 objects=20 used=20 fp=0x0000000000000000 flags=0xfffc00000010200
[ 1774.511296] INFO: Object 0xffff8000364ae008 @offset=8 fp=0xffff8000364afd20
[ 1774.511296]
[ 1774.514915] Redzone ffff8000364ae000: bb bb bb bb bb bb bb bb
........
[ 1774.518603] Object ffff8000364ae008: 11 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  .kkkkkkkkkkkkkkk
[ 1774.522580] Object ffff8000364ae018: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5
kkkkkkkkkkkkk.
[ 1774.526446] Redzone ffff8000364ae026: bb bb
..
[ 1774.529898] Padding ffff8000364ae160: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[ 1774.533904] Padding ffff8000364ae170: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[ 1774.537913] Padding ffff8000364ae180: 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZ
[ 1774.541595] CPU: 0 PID: 5940 Comm: insmod Tainted: G    B   W  O      5.0.0+ #24
[ 1774.544697] Hardware name: linux,dummy-virt (DT)
[ 1774.546662] Call trace:
[ 1774.547769]  dump_backtrace+0x0/0x154
[ 1774.549315]  show_stack+0x14/0x1c
[ 1774.550895]  dump_stack+0x88/0xb0
[ 1774.552466]  print_trailer+0x108/0x194
[ 1774.554235]  check_bytes_and_report+0xe0/0x124
[ 1774.556293]  check_object+0x210/0x258
[ 1774.558037]  alloc_debug_processing+0x154/0x1c4
[ 1774.560215]  ___slab_alloc+0x89c/0x8bc
[ 1774.562112]  __slab_alloc+0x24/0x34
[ 1774.563769]  kmem_cache_alloc+0x1b8/0x264
[ 1774.565689]  foo_init+0x110/0x1000 [poison]
[ 1774.567645]  do_one_initcall+0x48/0x160
[ 1774.569487]  do_init_module+0x54/0x1c4
[ 1774.571259]  load_module+0x1ec8/0x20ac
[ 1774.573014]  __se_sys_finit_module+0xc8/0xdc
[ 1774.575059]  __arm64_sys_finit_module+0x18/0x20
[ 1774.577203]  el0_svc_common+0xc0/0x100
[ 1774.578981]  el0_svc_handler+0x2c/0x70
[ 1774.580742]  el0_svc+0x8/0xc
[ 1774.582118] FIX jake: Restoring 0xffff8000364ae008-0xffff8000364ae008=0x6b
```

Redzone overwritten

The end of the red-zone on the left side of the 32-byte object size was invaded.

- After being allocated in foo_init+0x58/0x1000, it finds out when the slab object is free that the red-zone has been invaded.

```
[  901.853653] =============================================================================
[  901.857016] BUG jake (Tainted: G    B   W  O     ): Redzone overwritten
[  901.860153] -----------------------------------------------------------------------------
[  901.860153]
[  901.864665] INFO: 0xffff80003971c007-0xffff80003971c007. First byte 0x11 instead of 0xcc
[  901.868451] INFO: Allocated in foo_init+0x54/0x1000 [poison] age=139 cpu=0 pid=4085
[  901.872126]  __slab_alloc+0x24/0x34
[  901.873805]  kmem_cache_alloc+0x1b8/0x264
[  901.875729]  foo_init+0x54/0x1000 [poison]
[  901.877650]  do_one_initcall+0x48/0x160
[  901.879484]  do_init_module+0x54/0x1c4
[  901.881251]  load_module+0x1ec8/0x20ac
[  901.883053]  __se_sys_finit_module+0xc8/0xdc
[  901.885051]  __arm64_sys_finit_module+0x18/0x20
[  901.887255]  el0_svc_common+0xc0/0x100
[  901.888996]  el0_svc_handler+0x2c/0x70
[  901.890799]  el0_svc+0x8/0xc
[  901.892227] INFO: Slab 0xffff7e0000e5c700 objects=20 used=1 fp=0xffff80003971dd20 flags=0xfffc00000010201
[  901.896674] INFO: Object 0xffff80003971c008 @offset=8 fp=0xffff80003971c190
[  901.896674]
[  901.900681] Redzone ffff80003971c000: cc cc cc cc cc cc cc 11
........
[  901.904770] Object ffff80003971c008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
[  901.909141] Object ffff80003971c018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
..............
[  901.913406] Redzone ffff80003971c026: cc cc
..
[  901.917241] Padding ffff80003971c160: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[  901.921614] Padding ffff80003971c170: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[  901.926018] Padding ffff80003971c180: 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZ
[  901.930064] CPU: 0 PID: 4085 Comm: insmod Tainted: G    B   W  O      5.0.0+ #24
[  901.933449] Hardware name: linux,dummy-virt (DT)
[  901.935553] Call trace:
[  901.936723]  dump_backtrace+0x0/0x154
[  901.938436]  show_stack+0x14/0x1c
[  901.939978]  dump_stack+0x88/0xb0
[  901.941523]  print_trailer+0x108/0x194
[  901.943253]  check_bytes_and_report+0xe0/0x124
[  901.945320]  check_object+0x184/0x258
[  901.946981]  free_debug_processing+0x1e0/0x484
[  901.949706]  __slab_free+0x74/0x4b0
[  901.951371]  kmem_cache_free+0x220/0x230
[  901.953287]  foo_init+0x104/0x1000 [poison]
[  901.955290]  do_one_initcall+0x48/0x160
[  901.957125]  do_init_module+0x54/0x1c4
[  901.958932]  load_module+0x1ec8/0x20ac
[  901.960718]  __se_sys_finit_module+0xc8/0xdc
[  901.962721]  __arm64_sys_finit_module+0x18/0x20
[  901.964827]  el0_svc_common+0xc0/0x100
[  901.966606]  el0_svc_handler+0x2c/0x70
[  901.968403]  el0_svc+0x8/0xc
[  901.969785] FIX jake: Restoring 0xffff80003971c007-0xffff80003971c007=0xcc
[  901.969785]
[  901.978507] FIX jake: Object at 0xffff80003971c008 not freed
```

It's 32 bytes of object size, but by overwriting 2 more characters, the right red-zone is invaded.

- After being allocated in foo_init+0x54/0x1000, it finds out when the slab object is free that the red-zone has been invaded.

```
[ 1416.793614] =============================================================================
[ 1416.796913] BUG jake (Tainted: G    B   W  O     ): Redzone overwritten
[ 1416.799971] -----------------------------------------------------------------------------
[ 1416.799971]
[ 1416.804445] INFO: 0xffff80003b1c6026-0xffff80003b1c6027. First byte 0x11 instead of 0xcc
[ 1416.808208] INFO: Allocated in foo_init+0x54/0x1000 [poison] age=126 cpu=0 pid=4905
[ 1416.811812]  __slab_alloc+0x24/0x34
[ 1416.813468]  kmem_cache_alloc+0x1b8/0x264
[ 1416.815368]  foo_init+0x54/0x1000 [poison]
[ 1416.817276]  do_one_initcall+0x48/0x160
[ 1416.819100]  do_init_module+0x54/0x1c4
[ 1416.820886]  load_module+0x1ec8/0x20ac
[ 1416.822650]  __se_sys_finit_module+0xc8/0xdc
[ 1416.824629]  __arm64_sys_finit_module+0x18/0x20
[ 1416.826814]  el0_svc_common+0xc0/0x100
[ 1416.828593]  el0_svc_handler+0x2c/0x70
[ 1416.830363]  el0_svc+0x8/0xc
[ 1416.831725] INFO: Slab 0xffff7e0000ec7180 objects=20 used=1 fp=0xffff80003b1c7d20 flags=0xfffc00000010201
[ 1416.836152] INFO: Object 0xffff80003b1c6008 @offset=8 fp=0xffff80003b1c6190
[ 1416.836152]
[ 1416.840133] Redzone ffff80003b1c6000: cc cc cc cc cc cc cc cc
........
[ 1416.844160] Object ffff80003b1c6008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
[ 1416.848511] Object ffff80003b1c6018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
..............
[ 1416.852767] Redzone ffff80003b1c6026: 11 11
..
[ 1416.856576] Padding ffff80003b1c6160: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[ 1416.860998] Padding ffff80003b1c6170: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  ZZZZZZZZZZZZZZZZ
[ 1416.865338] Padding ffff80003b1c6180: 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZ
[ 1416.869421] CPU: 0 PID: 4905 Comm: insmod Tainted: G    B   W  O      5.0.0+ #24
[ 1416.872812] Hardware name: linux,dummy-virt (DT)
[ 1416.875023] Call trace:
[ 1416.876164]  dump_backtrace+0x0/0x154
[ 1416.877885]  show_stack+0x14/0x1c
[ 1416.879479]  dump_stack+0x88/0xb0
[ 1416.881010]  print_trailer+0x108/0x194
[ 1416.882789]  check_bytes_and_report+0xe0/0x124
[ 1416.884901]  check_object+0x1b4/0x258
[ 1416.886653]  free_debug_processing+0x1e0/0x484
[ 1416.888706]  __slab_free+0x74/0x4b0
[ 1416.890352]  kmem_cache_free+0x220/0x230
[ 1416.892196]  foo_init+0x104/0x1000 [poison]
[ 1416.894140]  do_one_initcall+0x48/0x160
[ 1416.895959]  do_init_module+0x54/0x1c4
[ 1416.897739]  load_module+0x1ec8/0x20ac
[ 1416.899555]  __se_sys_finit_module+0xc8/0xdc
[ 1416.901553]  __arm64_sys_finit_module+0x18/0x20
[ 1416.903754]  el0_svc_common+0xc0/0x100
[ 1416.905550]  el0_svc_handler+0x2c/0x70
[ 1416.907376]  el0_svc+0x8/0xc
[ 1416.908711] FIX jake: Restoring 0xffff80003b1c6026-0xffff80003b1c6027=0xcc
[ 1416.908711]
[ 1416.917035] FIX jake: Object at 0xffff80003b1c6008 not freed
```

Object padding overwritten

One byte was recorded in the padding position and the padding was violated.

- When freeing a slab object in foo_init+0x10c/0x1000, it finds out that the padding has been invaded and prints it out.

```
[  641.785286] =====================================================================
=========
[  641.788638] BUG jake (Tainted: G    B   W  O     ): Object padding overwritten
[  641.791951] ---------------------------------------------------------------------
---------
[  641.791951]
[  641.796458] INFO: 0xffff80003b2e4187-0xffff80003b2e4187. First byte 0x11 instead
of 0x5a
[  641.800288] INFO: Allocated in foo_init+0x54/0x1000 [poison] age=124 cpu=0 pid=3
812
[  641.803902]  __slab_alloc+0x24/0x34
[  641.805508]  kmem_cache_alloc+0x1b8/0x264
[  641.807441]  foo_init+0x54/0x1000 [poison]
[  641.809350]  do_one_initcall+0x48/0x160
[  641.811194]  do_init_module+0x54/0x1c4
[  641.812974]  load_module+0x1ec8/0x20ac
[  641.814741]  __se_sys_finit_module+0xc8/0xdc
[  641.816755]  __arm64_sys_finit_module+0x18/0x20
[  641.818906]  el0_svc_common+0xc0/0x100
[  641.820700]  el0_svc_handler+0x2c/0x70
[  641.822490]  el0_svc+0x8/0xc
[  641.823938] INFO: Slab 0xffff7e0000ecb900 objects=20 used=1 fp=0xffff80003b2e5d2
0 flags=0xfffc00000010201
[  641.828332] INFO: Object 0xffff80003b2e4008 @offset=8 fp=0xffff80003b2e4190
[  641.828332]
[  641.832301] Redzone ffff80003b2e4000: cc cc cc cc cc cc cc cc
........
[  641.836372] Object ffff80003b2e4008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00  ................
[  641.840714] Object ffff80003b2e4018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
..............
[  641.844984] Redzone ffff80003b2e4026: cc cc
..
[  641.848801] Padding ffff80003b2e4160: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a  ZZZZZZZZZZZZZZZZ
[  641.853194] Padding ffff80003b2e4170: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a  ZZZZZZZZZZZZZZZZ
[  641.857547] Padding ffff80003b2e4180: 5a 5a 5a 5a 5a 5a 5a 11
ZZZZZZZ.
[  641.861674] CPU: 0 PID: 3812 Comm: insmod Tainted: G    B   W  O      5.0.0+ #24
[  641.865067] Hardware name: linux,dummy-virt (DT)
[  641.867282] Call trace:
[  641.868449]  dump_backtrace+0x0/0x154
[  641.870200]  show_stack+0x14/0x1c
[  641.871793]  dump_stack+0x88/0xb0
[  641.873373]  print_trailer+0x108/0x194
[  641.875134]  check_bytes_and_report+0xe0/0x124
[  641.877221]  check_object+0xb4/0x258
[  641.878925]  free_debug_processing+0x1e0/0x484
[  641.880980]  __slab_free+0x74/0x4b0
[  641.882649]  kmem_cache_free+0x220/0x230
[  641.884519]  foo_init+0x104/0x1000 [poison]
[  641.886502]  do_one_initcall+0x48/0x160
[  641.888325]  do_init_module+0x54/0x1c4
[  641.890102]  load_module+0x1ec8/0x20ac
[  641.891877]  __se_sys_finit_module+0xc8/0xdc
[  641.893882]  __arm64_sys_finit_module+0x18/0x20
[  641.896013]  el0_svc_common+0xc0/0x100
[  641.897772]  el0_svc_handler+0x2c/0x70
[  641.899587]  el0_svc+0x8/0xc
[  641.900994] FIX jake: Restoring 0xffff80003b2e4187-0xffff80003b2e4187=0x5a
```

Sample Source – poison.c

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/string.h>
#include <linux/slab.h>
#include <linux/version.h>      /* LINUX_VERSION_CODE & KERNEL_VERSION() */
#include <linux/printk.h>

static int __init foo_init(void)
{
        int ret = 0;
        void *obj;
        int size = 768;

        printk("%s\n", __func__);

        s = kmem_cache_create("jake", OBJ_SIZE, OBJ_ALIGN, 0, NULL);

        obj = kmem_cache_alloc(s, 0);

        printk(KERN_ERR "\n");
        printk(KERN_ERR "---------------------------------------------------\n");
        print_hex_dump(KERN_ERR, "", DUMP_PREFIX_ADDRESS, 16, 1, obj, size, 1);

        memset(obj, 0, OBJ_SIZE);

        // memset(obj - 1, 0x11, 1);    /* Redzone overwritten */

        // memset(obj + 383, 0x11, 1);  /* Object padding overwritten */

        printk(KERN_ERR "\n");
        printk(KERN_ERR "---------------------------------------------------\n");
        print_hex_dump(KERN_ERR, "", DUMP_PREFIX_ADDRESS, 16, 1, obj, size, 1);

        kmem_cache_free(s, obj);

        // kmem_cache_free(s, obj);     /* Object already free */

        // memset(obj, 0x11, 1);        /* Poison overwritten */
        // obj = kmem_cache_alloc(s, 0);
        // kmem_cache_free(s, obj);

        printk(KERN_ERR "\n");
        printk(KERN_ERR "---------------------------------------------------\n");
        print_hex_dump(KERN_ERR, "", DUMP_PREFIX_ADDRESS, 16, 1, obj, size, 1);

        return ret;     /* 0=success */
}

```

```
static void __exit foo_exit(void)
{

        printk("%s\n", __func__);

        kmem_cache_destroy(s);
}

module_init(foo_init);
module_exit(foo_exit);
MODULE_LICENSE("GPL");
```

Here is the normal output:

```
$ insmod poison.ko
[81403.699138] foo_init
[  448.848125] ---------------------------------------------
[  448.968221] ffff80003b1b6008: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  k
kkkkkkkkkkkkkkk
[  448.978732] ffff80003b1b6018: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5 cc cc  k
kkkkkkkkkkkk...
[  448.985722] ffff80003b1b6028: 90 61 1b 3b 00 80 ff ff 54 a0 b8 08 00 00 ff ff  .
a.;....T.......
[  448.988850] ffff80003b1b6038: 5c b0 22 10 00 00 ff ff 24 b2 22 10 00 00 ff ff
\.".....$.".....
[  448.993154] ffff80003b1b6048: 54 a0 b8 08 00 00 ff ff 18 42 08 10 00 00 ff ff
T........B......
[  448.996288] ffff80003b1b6058: 18 19 16 10 00 00 ff ff e0 39 16 10 00 00 ff ff
.........9......
[  449.000385] ffff80003b1b6068: 80 3e 16 10 00 00 ff ff cc 3e 16 10 00 00 ff ff  .
>.......>......
[  449.004501] ffff80003b1b6078: 04 4e 09 10 00 00 ff ff 70 4e 09 10 00 00 ff ff  .
N......pN......
[  449.008632] ffff80003b1b6088: 48 40 08 10 00 00 ff ff 00 00 00 00 00 00 00 00  H
@..............
[  449.012671] ffff80003b1b6098: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.016834] ffff80003b1b60a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.020962] ffff80003b1b60b8: 00 00 00 00 d0 0d 00 00 25 91 00 00 01 00 00 00
........%.......
[  449.025210] ffff80003b1b60c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.028337] ffff80003b1b60d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.032464] ffff80003b1b60e8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.036560] ffff80003b1b60f8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.040701] ffff80003b1b6108: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.044866] ffff80003b1b6118: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.049156] ffff80003b1b6128: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.052225] ffff80003b1b6138: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.056406] ffff80003b1b6148: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.060468] ffff80003b1b6158: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a
........ZZZZZZZZ
[  449.064601] ffff80003b1b6168: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.068740] ffff80003b1b6178: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.072893] ffff80003b1b6188: bb bb bb bb bb bb bb bb 6b 6b 6b 6b 6b 6b 6b 6b
........kkkkkkkk
[  449.077104] ffff80003b1b6198: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  k
kkkkkkkkkkkkkkk
[  449.080262] ffff80003b1b61a8: 6b 6b 6b 6b 6b a5 bb bb 00 00 00 00 00 00 00 00  k
```

```
kkkk...........
[  449.084328] ffff80003b1b61b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.088401] ffff80003b1b61c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.092573] ffff80003b1b61d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.096696] ffff80003b1b61e8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.100803] ffff80003b1b61f8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.104885] ffff80003b1b6208: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.109140] ffff80003b1b6218: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.112250] ffff80003b1b6228: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.116340] ffff80003b1b6238: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.120410] ffff80003b1b6248: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.124536] ffff80003b1b6258: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.128709] ffff80003b1b6268: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.132914] ffff80003b1b6278: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.137103] ffff80003b1b6288: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.140199] ffff80003b1b6298: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.144304] ffff80003b1b62a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.148720] ffff80003b1b62b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.152893] ffff80003b1b62c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.157051] ffff80003b1b62d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.160192] ffff80003b1b62e8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.164503] ffff80003b1b62f8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.168664]
[  449.169693] --------------------------------------------------
[  449.171833] ffff80003b1b6008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.176219] ffff80003b1b6018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 cc cc
................
[  449.180475] ffff80003b1b6028: 90 61 1b 3b 00 80 ff ff 54 a0 b8 08 00 00 ff ff  .
a.;....T.......
[  449.184559] ffff80003b1b6038: 5c b0 22 10 00 00 ff ff 24 b2 22 10 00 00 ff ff
\.".....$.".....
[  449.188792] ffff80003b1b6048: 54 a0 b8 08 00 00 ff ff 18 42 08 10 00 00 ff ff
T........B......
[  449.192897] ffff80003b1b6058: 18 19 16 10 00 00 ff ff e0 39 16 10 00 00 ff ff
```

```
.........9......
[  449.197151] ffff80003b1b6068: 80 3e 16 10 00 00 ff ff cc 3e 16 10 00 00 ff ff  .
>.......>......
[  449.200274] ffff80003b1b6078: 04 4e 09 10 00 00 ff ff 70 4e 09 10 00 00 ff ff  .
N......pN......
[  449.204465] ffff80003b1b6088: 48 40 08 10 00 00 ff ff 00 00 00 00 00 00 00 00  H
@..............
[  449.208564] ffff80003b1b6098: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.212645] ffff80003b1b60a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.216746] ffff80003b1b60b8: 00 00 00 00 d0 0d 00 00 25 91 00 00 01 00 00 00
........%.......
[  449.220898] ffff80003b1b60c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.225150] ffff80003b1b60d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.228231] ffff80003b1b60e8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.232343] ffff80003b1b60f8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.236527] ffff80003b1b6108: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.240811] ffff80003b1b6118: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.244950] ffff80003b1b6128: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.249140] ffff80003b1b6138: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.252289] ffff80003b1b6148: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.256440] ffff80003b1b6158: 00 00 00 00 00 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a
........ZZZZZZZZ
[  449.260563] ffff80003b1b6168: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.264670] ffff80003b1b6178: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.268835] ffff80003b1b6188: bb bb bb bb bb bb bb bb 6b 6b 6b 6b 6b 6b 6b 6b
........kkkkkkkk
[  449.272971] ffff80003b1b6198: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  k
kkkkkkkkkkkkkkk
[  449.277139] ffff80003b1b61a8: 6b 6b 6b 6b 6b a5 bb bb 00 00 00 00 00 00 00 00  k
kkkk...........
[  449.280274] ffff80003b1b61b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.284607] ffff80003b1b61c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.288818] ffff80003b1b61d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.293014] ffff80003b1b61e8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.296146] ffff80003b1b61f8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.300283] ffff80003b1b6208: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.304305] ffff80003b1b6218: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
................
[  449.308369] ffff80003b1b6228: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.312433] ffff80003b1b6238: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.316562] ffff80003b1b6248: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.320615] ffff80003b1b6258: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.324791] ffff80003b1b6268: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.329059] ffff80003b1b6278: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.332165] ffff80003b1b6288: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.336283] ffff80003b1b6298: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.340471] ffff80003b1b62a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.345025] ffff80003b1b62b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.348173] ffff80003b1b62c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.352216] ffff80003b1b62d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.356293] ffff80003b1b62e8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.360331] ffff80003b1b62f8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.408843]
[  449.409632] -------------------------------------------------
[  449.411684] ffff80003b1b6008: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  k
kkkkkkkkkkkkkkk
[  449.415920] ffff80003b1b6018: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5 bb bb  k
kkkkkkkkkkkk...
[  449.419903] ffff80003b1b6028: 20 7d 1b 3b 00 80 ff ff 54 a0 b8 08 00 00 ff ff
}.;....T.......
[  449.424012] ffff80003b1b6038: 5c b0 22 10 00 00 ff ff 24 b2 22 10 00 00 ff ff
\.".....$.".....
[  449.427968] ffff80003b1b6048: 54 a0 b8 08 00 00 ff ff 18 42 08 10 00 00 ff ff
T........B......
[  449.431977] ffff80003b1b6058: 18 19 16 10 00 00 ff ff e0 39 16 10 00 00 ff ff
.........9......
[  449.435914] ffff80003b1b6068: 80 3e 16 10 00 00 ff ff cc 3e 16 10 00 00 ff ff  .
>.......>......
[  449.439910] ffff80003b1b6078: 04 4e 09 10 00 00 ff ff 70 4e 09 10 00 00 ff ff  .
N......pN......
[  449.443825] ffff80003b1b6088: 48 40 08 10 00 00 ff ff 00 00 00 00 00 00 00 00  H
@..............
[  449.447773] ffff80003b1b6098: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.451670] ffff80003b1b60a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.455609] ffff80003b1b60b8: 00 00 00 00 d0 0d 00 00 25 91 00 00 01 00 00 00
........%.......
[  449.459796] ffff80003b1b60c8: fc a0 b8 08 00 00 ff ff c4 cb 22 10 00 00 ff ff
```

```
..........".....
[  449.464003] ffff80003b1b60d8: fc a0 b8 08 00 00 ff ff 18 42 08 10 00 00 ff ff
.........B......
[  449.468412] ffff80003b1b60e8: 18 19 16 10 00 00 ff ff e0 39 16 10 00 00 ff ff
.........9......
[  449.472887] ffff80003b1b60f8: 80 3e 16 10 00 00 ff ff cc 3e 16 10 00 00 ff ff  .
>.......>......
[  449.477496] ffff80003b1b6108: 04 4e 09 10 00 00 ff ff 70 4e 09 10 00 00 ff ff  .
N......pN......
[  449.480951] ffff80003b1b6118: 48 40 08 10 00 00 ff ff 00 00 00 00 00 00 00 00  H
@..............
[  449.485555] ffff80003b1b6128: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.489479] ffff80003b1b6138: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.493448] ffff80003b1b6148: 00 00 00 00 00 00 00 00 00 00 00 00 d0 0d 00 00
................
[  449.496857] ffff80003b1b6158: a3 91 00 00 01 00 00 00 5a 5a 5a 5a 5a 5a 5a 5a
........ZZZZZZZZ
[  449.501630] ffff80003b1b6168: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.505645] ffff80003b1b6178: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.509587] ffff80003b1b6188: bb bb bb bb bb bb bb bb 6b 6b 6b 6b 6b 6b 6b 6b
........kkkkkkkk
[  449.513589] ffff80003b1b6198: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b  k
kkkkkkkkkkkkkkk
[  449.517409] ffff80003b1b61a8: 6b 6b 6b 6b 6b a5 bb bb 00 00 00 00 00 00 00 00  k
kkkk...........
[  449.520900] ffff80003b1b61b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.525481] ffff80003b1b61c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.528980] ffff80003b1b61d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.533127] ffff80003b1b61e8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.536551] ffff80003b1b61f8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.541202] ffff80003b1b6208: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.544601] ffff80003b1b6218: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.549210] ffff80003b1b6228: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.552662] ffff80003b1b6238: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.557293] ffff80003b1b6248: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.560751] ffff80003b1b6258: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.565518] ffff80003b1b6268: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.568934] ffff80003b1b6278: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.573518] ffff80003b1b6288: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
................
[  449.577492] ffff80003b1b6298: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.581436] ffff80003b1b62a8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.585435] ffff80003b1b62b8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.588860] ffff80003b1b62c8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.593552] ffff80003b1b62d8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
[  449.597851] ffff80003b1b62e8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
[  449.601905] ffff80003b1b62f8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a  Z
ZZZZZZZZZZZZZZZ
```

kptr_restrict

As of kernel v4.15, the pointer address used by the kernel is hidden. To be able to see this, you need to use "%px" instead of "%p" for the format of the printk() function, or use "%pK" and have admin privileges perform "echo 1 > /proc/sys/kernel/kptr_restrict" to remove the kernel pointer output limit.

Makefile

```
obj-m := foo.o
KER := $(shell uname -r)
KER_DIR := /lib/modules/$(KER)/build

all:
        make -C ${KER_DIR} M=$(PWD) modules

clean:
        make -C ${KER_DIR} M=$(PWD) clean
```

foo.c

```
#include <linux/kernel.h>
#include <linux/module.h>

int param = 0;
module_param(param, int, 0);

int foo_init(void)
{
        void *p = foo_init;

        printk(KERN_INFO "kptr_restrict=%d, x=0x%llx, p=%p, pK=%pK, px=%
px\n",
                        param, (long long unsigned int) p, p, p, p);
        return 0;
}

void foo_exit(void)
{
}
```

```

module_init(foo_init);
module_exit(foo_exit);

MODULE_DESCRIPTION("foo");
MODULE_LICENSE("GPL");
```

run.sh

```
echo 0 > /proc/sys/kernel/kptr_restrict
insmod foo.ko param=0
rmmod foo

echo 1 > /proc/sys/kernel/kptr_restrict
insmod foo.ko param=1
rmmod foo

echo 2 > /proc/sys/kernel/kptr_restrict
insmod foo.ko param=2
rmmod foo
```

Execution results

If you print a function pointer from the printk() function with admin privileges, it will show the following result depending on the kptr_restrict settings.

- %llx option
  - Regardless of the kptr_restrict value, it shows the correct value.
- %p option
  - Regardless of the kptr_restrict value, it always shows a false value.
- %pK Options
  - If kptr_restrict=1, it will be shown correctly.
  - If kptr_restrict=2, it is printed as 0000000000000000.
- %px
  - Regardless of the kptr_restrict value, it shows the correct value.

```
$ ./run.sh
kptr_restrict=0, x=0xffff800008ae9000, p=00000000467430b8, pK=00000000467430b8, px=
ffff800008ae9000
kptr_restrict=1, x=0xffff800008ae9000, p=00000000467430b8, pK=ffff800008ae9000, px=
ffff800008ae9000
kptr_restrict=2, x=0xffff800008ae9000, p=00000000467430b8, pK=0000000000000000, px=
ffff800008ae9000
```

consultation

- Slab Memory Allocator -1- (Structure) (http://jake.dothome.co.kr/slub/) | Sentence C – Current post
- Slab Memory Allocator -2- (Initialize Cache) (http://jake.dothome.co.kr/kmem_cache_init) | Qc
- Slub Memory Allocator -3- (Create Cache) (http://jake.dothome.co.kr/slub-cache-create) | Qc

- Slub Memory Allocator -4- (Calculate Order) (http://jake.dothome.co.kr/slub-order) | Qc
- Slub Memory Allocator -5- | (http://jake.dothome.co.kr/slub-slub-alloc) Qc
- Slub Memory Allocator -6- (Assign Object) (http://jake.dothome.co.kr/slub-object-alloc) | Qc
- Slub Memory Allocator -7- (Object Unlocked) (http://jake.dothome.co.kr/slub-object-free) | Qc
- Slub Memory Allocator -8- (Drain/Flash Cache) (http://jake.dothome.co.kr/slub-drain-flush-cache) | Qc
- Slub Memory Allocator -9- (Cache Shrink) (http://jake.dothome.co.kr/slub-cache-shrink) | Qc
- Slub Memory Allocator -10- | (http://jake.dothome.co.kr/slub-slub-free) Qc
- Slub Memory Allocator -11- (Clear Cache (http://jake.dothome.co.kr/slub-cache-destroy)) | Qc
- Slub Memory Allocator -12- (Debugging Slub) (http://jake.dothome.co.kr/slub-debug) | Sentence C – Current post
- Slub Memory Allocator -13- (slabinfo) (http://jake.dothome.co.kr/slub-slabinfo) | 문c

- Kernel dynamic memory allocation tracking and reduction (2012) | Ezequiel García – 다운로드 pdf (https://elinux.org/images/8/81/Elc2013_Garcia.pdf)
- slub_debug: Detect Kernel heap memory corruption (http://techvolve.blogspot.kr/2014/04/slubdebug-detect-kernel-heap-memory.html) | TechVolve

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

NUMA -2- (Fallback Node) (http://jake.dothome.co.kr/numa-fallback-node/)

Slub Memory Allocator -8- (Drain/Flush Cache) (http://jake.dothome.co.kr/slub-drain-flush-cache/)