Kmemleak

Kernel Memory Leak (kmemleak)

Contents

Introduction

Memory leak occurs when a piece of memory that was previously allocated by a programmer is not properly deallocated. **Kmemleak** is a feature of kernel, that helps in finding the memory leaks in **kernel space**. It only reports the list of memory leaks in the kernel space.

Valgrind is a similar tool to find the memory leak in **user space**.

Prerequisites

CONFIG_DEBUG_KMEMLEAK in kernel configuration need to be enabled as follows

1. Open configuration file.

```
$ make menuconfig
```

2. Enable CONFIG_DEBUG_KMEMLEAK option in Memory Debugging and the location is as below

```
Location:
   -> Kernel Hacking
     -> Memory Debugging
      -> Maximum kmemleak early log entries (change 400 to 8000)
```

3. Once configuration is enabled, set the early log values in **Maximum kmemleak early log entries** to 4000. The default value of 400 may not work correctly in all configurations.

4. Build the kernel using following commands

```
$ make
$ make modules
$ make modules_install
$ make install
```

5. Reboot, once the build is complete.

6. Upon rebooting, mount debugfs file system.

```
$ mount -t debugfs nodev /sys/kernel/debug/
```

Note :If **/sys/kernel/debug** is already mounted, you will get response message '**/sys/kernel/debug/ busy**'

7. Once debugfs is mounted, we can see a file **kmemleak** under debugfs mounted location.

```
$ cat /sys/kernel/debug/kmemleak
```

The above ***/sys/kernel/debug/kmemleak*** contains the information about any memory leak that has been detected.

Usage

1. Now let us see the usage of kmemleak to detect memory leak in the sample source (sample_module.c)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/vmalloc.h>
void myfunc(void)
{
        char *ptr;
        ptr = vmalloc(512);
}
static int my_init(void)
{
        printk(KERN_ALERT "Hello World");
        myfunc();
        return 0;
}
static void my_exit(void)
{
        printk(KERN_ALERT "Exit");

}
module_init(my_init);
module_exit(my_exit);

MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("Sample_module");
```

2. Compile the above code and insert the sample_module.ko module.

```
$ insmod sample.ko
```

3. The memory leak detection thread runs periodically, now perform the test.

```
$ echo scan > /sys/kernel/debug/kmemleak
```

4. Let us check for leak.

```
$ cat /sys/kernel/debug/kmemleak
```

5. If leak detected, you should see a prompt like:

```
unreferenced object 0xffffc90000667000 (size 512):
 comm "insmod", pid 5602, jiffies 4297141725 (age 18.452s)
 hex dump (first 32 bytes):
   0a 00 00 00 0e 00 00 00 6b 56 00 00 00 00 00 00  ........kV......
```

```
  e8 82 00 00 00 00 00 0a 00 00 00 0e 00 00 00  ...............
backtrace:
  [<ffffffff817e449a>] kmemleak_alloc+0x4a/0xa0
  [<ffffffff811ca55c>] __vmalloc_node_range+0x1ac/0x290
  [<ffffffff811ca904>] vmalloc+0x54/0x60
  [<ffffffffa054e041>] my_init+0x21/0x30 [sample]
  [<ffffffff81002190>] do_one_initcall+0x50/0x190
  [<ffffffff81182b89>] do_init_module+0x60/0x1f1
  [<ffffffff81107094>] load_module+0x21a4/0x2910
  [<ffffffff811079f6>] SYSC_finit_module+0x96/0xd0
  [<ffffffff81107a4e>] SyS_finit_module+0xe/0x10
  [<ffffffff817eeb3b>] entry_SYSCALL_64_fastpath+0x1e/0xad
  [<ffffffffffffffff>] 0xffffffffffffffff
```

The bold text above represents the leak detected in my_init function.

Note : The memory leak detector code may take sometime to detect the leak, so repeat step 3 and 4.

6. To clear the list of all current memory leaks

```
$ echo clear > /sys/kernel/debug/kmemleak
```

References