ltp 之 fork14 问题分析

1 问题概述

在测试 ltp 的 fork14 程序时，出现错误：

mmap() fails too manytimes, so we are almost impossible to get an vm_area_struct sized 16TB.

2 问题分析

排查 fork14.c 代码，发现有段代码会不停的调用 mmap 函数：

```
#define EXTENT    (16 * 1024 + 10)
#define GB    (1024 * 1024 * 1024L)
for (i = 0; i < EXTENT; i++)
  {
    addr = mmap(NULL, 1 *GB, PROT_READ | PROT_WRITE,
          MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
  }
```

本段代码打算调用 16 * 1024 + 10 = 16394 次 mmap，每次映射 1GB 空间，获取共计 16TB 的空间。排查代码发现，fork14 是为了验证内核代码的一个小 bug，这个小 bug 在 https://lkml.org/lkml/2012/4/24/328 有说明。即 vma length 在内核代码 dup_mmap 中计算，算完了会存到一个 int 型变量中，但是对于超大空间的映射比如 16TB，这个 int 量存的长度就不够用了。

于是检查我们的内核代码 4.19.90 中的 dup_mmap 函数，发现如下代码片段：

```
if (mpnt->vm_flags & VM_ACCOUNT) {
      unsigned long len = vma_pages(mpnt);
}
```

即 4.19.90 的代码已经修复了这个问题，算出来的长度是存到 unsigned long 类型变量中。那么造成 mmap 调用失败另有原因。

随即收集一些信息：

1. 首先用 strace -o log ./fork14 把系统调用 dump 到 log 文件中，分析发现 mmap 调用会出现 ENOMEM 错误，另外 clone 也会出现 ENOMEM 错误。

2. 用 perror 打印下出错调用的描述信息：Cannot allocate memory，对应错误代码 12，用命令

   cpp -dM /usr/include/errno.h | grep 'define E' | sort -n -k 3

   看下正好对应 ENOMEM。

那么造成 ENOMEM 的原因究竟是什么呢？于是 man mmap 发现有三个可能的原因：

1. No memory is available.
2. The process's maximum number of mappings would have been exceeded. This error can also occur for munmap(), when unmapping a region in the middle of an existing mapping, since this results in two smaller mappings on either side of the region being unmapped.
3. (since Linux 4.7) The process's RLIMIT_DATA limit, described in getrlimit(2), would have been exceeded.

3 实验验证

于是按照上述三个原因依次验证。对于第一个内存（虚拟空间）不足，看起来不太容易验证，查找一个进程的虚拟空间（再细节的说，还要知道哪段对应的是 mmap 所能用的）有多大，是 OS+Architecture 相关的，采用排除法，先验证 2 和 3 两个较为简单的。

2 是说一个进程调用 mmap 的次数也是有限制的，那当前这个次数是多少呢？命令：

cat /proc/sys/vm/max_map_count

可以看到是 65530<16 * 1024 + 10 = 16394 次，所以原因 2 排除。

原因三是说，自 4.7 的内核版本起，一个进程能 mmpa 的空间还受到一个叫做 RLIMIT_DATA 量的限制，那这个又是什么呢？于是 man getrlimit 得知：

- RLIMIT_DATA: This is the maximum size of the process's data segment (initialized data, uninitialized data, and heap). The limit is specified in bytes, and is rounded down to the system page size. This limit affects calls to brk(2), sbrk(2), and (since Linux 4.7) mmap(2), which fail with the error ENOMEM upon encountering the soft limit of this resource.

于是编写程序：

```
#include <sys/time.h>
#include <sys/resource.h>
#include <stdio.h>
int main()
{
        struct rlimit r_lims;
        getrlimit(RLIMIT_DATA, &r_lims);
        printf("rlim_cur of RLIMIT_DATA: %llu\n", r_lims.rlim_cur);
        printf("rlim_max of RLIMIT_DATA: %llu\n", r_lims.rlim_max);
        return 0;
}
```

运行得知限制值为 9223372036854775807，其远大于 16TB，所以这个也不是原因。

所以还是得回到原因 1，寻找 OS+Architecture 上的限制。

那么，mmap 能映射的虚拟空间，其起始和终止地址在哪里呢？这个问题需要恰当的关键字 +Google，然后学习一点知识再结合内核代码和申威架构才能回答了，主要的关键字就是 process address space。参考文献 1 讲述了一点基础理论。

但是可以先尝试修改 fork14.c，看看这个映射出来的地址最大和最小值是啥。在适当位置添加如下代码到 fork14.c 中：

```
unsigned long min = 0x40000000000, max = 1;

if ((unsigned long)addr < min)
  min = (unsigned long)addr;
if ((unsigned long)addr > max &&
  addr != MAP_FAILED)
  max = (unsigned long)addr;

printf("min:\t%lx\n", min);
printf("max:\t%lx\n", max);
printf("i: %d\n", i);
```

输出是：

min: 2000
max: 3ff8cfaa000
i: 4094

其中 i 是 mmap 能成功执行的次数，即能映射 4094GB，而 min 是映射的起始最小地址，max 是映射的终止地址。多次运行发现，i 总为 4094，min 总为 2000，而 max 在 0x40000000000 附近。

通过参考文献 1 知道，TASK_UNMAPPED_BASE（即 mm->mmap_base）应该（只是应该）为映射的起始地址，而终止地址应该在靠近 TASK_SIZE 附近。

检查内核代码，文件 arch/sw_64/include/asm/processor.h 定义了 TASK_UNMAPPED_BASE 和 TASK_SIZE，我们的情况是三级页表，TASK_SIZE 是 0x40000000000UL，TASK_UNMAPPED_BASE 是 TASK_SIZE / 2 = 0x20000000000UL，而不是 0x40000000。为了避免内核重启，而我们的目的仅仅是想获取某些内核里的信息，这可以写个模块加载到内核中得到验证。

模块 C 程序 lkm_example.c：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/mm.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Qiyuan Pu");
```

```
MODULE_DESCRIPTION("get info");
MODULE_VERSION("0.01");

static int __init get_info_init(void)
{
  printk(KERN_INFO "The process is \"%s\" (pid %i)\n",
       current->comm, current->pid);
  printk(KERN_INFO "mmap_base: %lx\n",
       current->mm->mmap_base);
  printk("current.task_size: %lx\n", current->mm->task_size);
  printk("last_wakee.mmap_base: %lx\n",
       current->last_wakee->mm->mmap_base);
  printk(KERN_INFO "Memory TASK_SIZE: 0x%lx\n", TASK_SIZE);

  return 0;
}

static void __exit get_info_exit(void)
{
  printk(KERN_INFO "Goodbye, World!\n");
}
module_init(get_info_init);
module_exit(get_info_exit);
```

这个程序会取出当前进程的 comm 名字，当前进程的 mmap 映射起始地址 mmap_base 以及 task_size，另外还取了 last_wakee 的 mmap_base 以验证 TASK_UNMAPPED_BASE 的不变性。内核定义的 TASK_SIZE 常量也验证了等于进程自己的 task_size。

采用如下 Makefile 来编译测试：

```
obj-m += lkm_example.o
all:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
test:
  sudo dmesg -C
  make
  sudo insmod lkm_example.ko
  sudo rmmod lkm_example.ko
  dmesg
```

使用命令 make test 编译、测试以及打印。当然要注意安装需要的构建包和内核头文件，也可能要进行必要的链接，比如如下的一些命令可能需要。自己构建包的话就是自己手动 dpkg 安装 header 文件了，注意构建 modlue 的内核版本要和运行的内核版本相对应：

```
$ sudo apt-get install build-essential linux-headers-`uname -r`
$ ln -s /usr/src/linux-headers-`uname -r` /lib/modules/`uname -r`/build
```

最后可以看到输出：

[74230.556000] The process is ”insmod” (pid 20434)
[74230.556000] mmap_base: 20000000000
[74230.556000] current.task_size: 40000000000
[74230.556000] last_wakee.mmap_base: 20000000000
[74230.556000] Memory TASK_SIZE: 0x40000000000
[74230.620000] Goodbye, World!

即 mmap 的起始映射地址应该为 0x20000000000，而终止地址应该是 0x40000000000，由于最高的 userland 程序并不能达到 0x40000000000，最高端处有一点 gap，所以 0x3ff8cfaa000 这样的值靠近 0x40000000000 相对合理。但是起始地址 2000 明显远小于 0x20000000000，通过搜索 mmap min addr 关键字和 man mmap 知道：

> on Linux, the kernel will pick a nearby page boundary (but always above or equal to the value specified by /proc/sys/vm/mmap_min_addr) and attempt to create the mapping there. If another mapping already exists there

所以/proc/sys/vm/mmap_min_addr 实际上还会控制 mmap 能起始的最小地址，当前实验平台的值是 4096，所以 0x2000 靠近而大于 4096 是合理的，从 0x2000 到

至此，我们从理论与实验角度共同验证了为什么在申威（三级页表）时最多只能 mmap 映射 4094GB 的空间。

4 解决方案

如果想要更大（大于 4094GB，实际比较少见，这个 mmap 映射大小实际上已经占用了 stack、heap 等的空间）的 mmap 映射空间，可以考虑四级页表的内核方案，这时 TASK_SIZE 为 0x10000000000000UL，意即 mmap 能映射的最大极限空间约为 4096TB>16TB。

5 参考文献

1. https://www.ics.uci.edu/ aburtsev/cs5460/lectures/lecture22-virtual-process-address-space/lecture22-virtual-process-address-space.pdf
2. https://www.kernel.org/doc/gorman/html/understand/understand007.html