
Project 3

陈思贝 (718030290013)

0 实验配置

- Linux Kernel: 4.15.0
- OS: Ubuntu 18.04.5 LTS

1 实验过程

`mtest` 模块加载后即会生成一个 `/proc/mtest` 文件。利用实验一中掌握的对 `/proc` 文件的操作，实现模块对于不同输入的操作与反馈。

1.1 LISTVMA

该模块会列出当前进程的所有虚拟内存的地址。其中 `vm_area_struct` 的定义在 `linux/mm_types.h` 文件中。`vm_start` 和 `vm_end` 分别指向了 `vma` 的起始地址。这是一个双向链表，`vm_next` 指向了下一个 `vma`，只需用循环就可遍历所有虚拟内存的地址并输出。

```
280 struct vm_area_struct {
281     /* The first cache line has the info for VMA tree walking. */
282
283     unsigned long vm_start;    /* Our start address within vm_mm. */
284     unsigned long vm_end;     /* The first byte after our end address
285                               within vm_mm. */
286
287     /* linked list of VM areas per task, sorted by address */
288     struct vm_area_struct *vm_next, *vm_prev;
289     ...
303     unsigned long vm_flags;    /* Flags, see mm.h. */
304     ...
509 } __randomize_layout;
```

`vm_flags` 可以判断 `vma` 的读写执行权限，在 `linux/mm.h` 中有定义宏，只需将 `vm_flags` 和相应的宏进行二元操作，就可以计算出相对应的权限。实际运行结果见图1。

```
172 #define VM_READ    0x00000001 /* currently active flags */
173 #define VM_WRITE   0x00000002
174 #define VM_EXEC    0x00000004
```

1.2 FINDPAGE

该模块输入一个虚拟内存地址，会输出该虚拟地址在操作系统中所对应的物理内存的地址。本次实验所用的 Linux 系统仍然采用了四级分页的机制，其中分页的定义位于 `arch/arm64/include/asm/pgtable-types.h` 和 `arch/arm64/include/asm/pgtable.h` 中。分别为 `pte`、`pmd`、`pud` 和 `pgd` 四层分页。

```
25 typedef u64 pteval_t;
26 typedef u64 pmdval_t;
27 typedef u64 pudval_t;
28 typedef u64 pgdval_t;
```

其中还定义了大量的分页位移宏，通过这些宏，可以通过多次映射，计算出虚拟内存地址所对应的物理内存地址。然后通过返回的页物理地址和虚拟地址的偏移量相结合，计算出最终的物理内存地址。

测试结果见图2。

```

425 #define pte_offset_phys(dir,addr)  (pmd_page_paddr(READ_ONCE(*(dir))) + pte_index(addr) * sizeof(pte_t))
426 #define pte_offset_kernel(dir,addr) ((pte_t *)__va(pte_offset_phys((dir), (addr))))
427
428 #define pte_offset_map(dir,addr)    pte_offset_kernel((dir), (addr))
429 #define pte_offset_map_nested(dir,addr) pte_offset_kernel((dir), (addr))
...
476 #define pmd_offset_phys(dir, addr)  (pud_page_paddr(*(dir)) + pmd_index(addr) * sizeof(pmd_t))
477 #define pmd_offset(dir, addr)        ((pmd_t *)__va(pmd_offset_phys((dir), (addr))))
...
528 #define pud_offset_phys(dir, addr)  (pgd_page_paddr(*(dir)) + pud_index(addr) * sizeof(pud_t))
529 #define pud_offset(dir, addr)        ((pud_t *)__va(pud_offset_phys((dir), (addr))))
530 #define pgd_offset_raw(pgd, addr)    ((pgd) + pgd_index(addr))
...
558 #define pgd_offset(mm, addr)        (pgd_offset_raw((mm)->pgd, (addr)))
559
560 /* to find an entry in a kernel page-table-directory */
561 #define pgd_offset_k(addr)  pgd_offset(&init_mm, addr)

```

1.3 WRITEVAL

该模块接受两个值，分别为 1 个虚拟内存地址和 1 个数值，并将数值写入虚拟内存的地址。通过 `find_vma` 先找到 `vma` 的地址，并检验合格性，再检测是否为可写入的页。

再通过上一个模块的 `find_page` 计算出虚拟地址对应的物理地址，直接将数值写入该物理地址即可。测试结果见图3。

2 实验效果截图

```

tonychen@ecs-433a: ~/CS353/project3/718030290013_陈思贝_project3_src — ssh tonychen@huaweicloud.tonychen.page — 1...
tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$ echo listvma > /proc/mtest && dmesg | tail -20
[ 1570.128123] VMA 0xffff8b035000 - 0xffff8b036000
[ 1570.128124] r
-
x

[ 1570.128125] VMA 0xffff8b036000 - 0xffff8b037000
[ 1570.128125] r
-
-

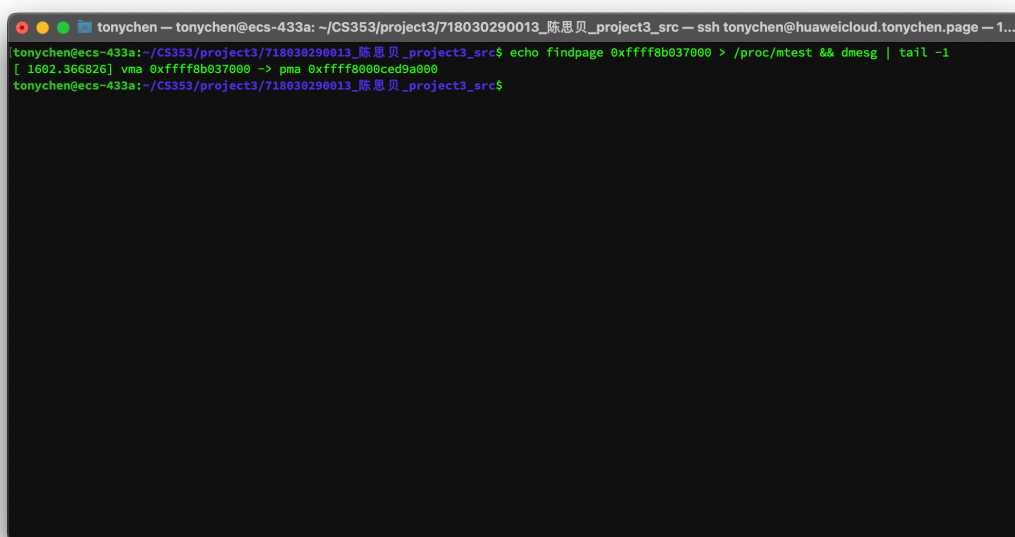
[ 1570.128126] VMA 0xffff8b037000 - 0xffff8b039000
[ 1570.128127] r
w
-

[ 1570.128128] VMA 0xfffff0000000 - 0xfffff0001000
[ 1570.128128] r
w
-

tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$

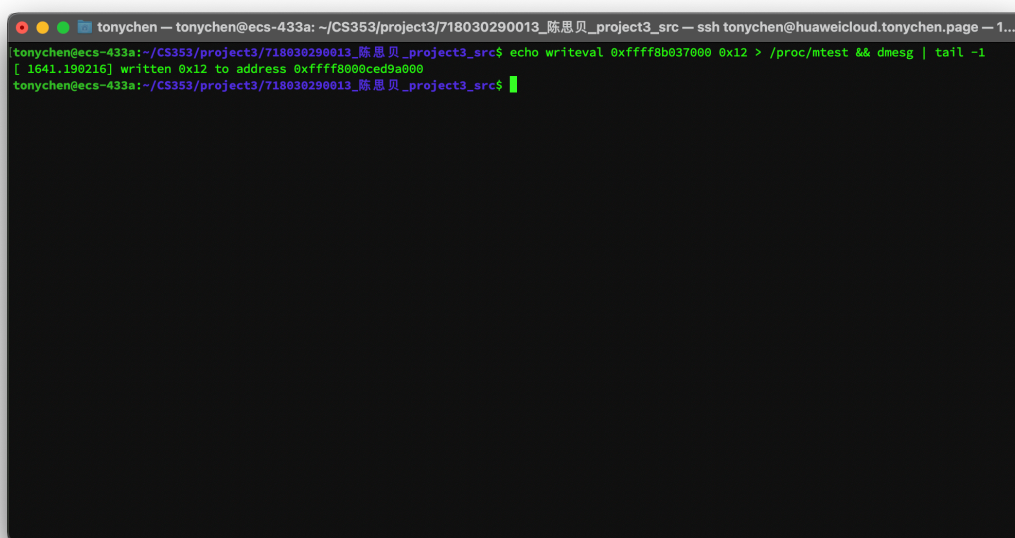
```

图 1: listvma 部分运行结果



```
tonychen@ecs-433a: ~/CS353/project3/718030290013_陈思贝_project3_src — ssh tonychen@huaweicloud.tonychen.page — 1...
tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$ echo findpage 0xffff8b037000 > /proc/mtest && dmesg | tail -1
[ 1602.366826] vma 0xffff8b037000 -> pma 0xffff8000ced9a000
tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$
```

图 2: findpage 运行结果



```
tonychen@ecs-433a: ~/CS353/project3/718030290013_陈思贝_project3_src — ssh tonychen@huaweicloud.tonychen.page — 1...
tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$ echo writeval 0xffff8b037000 0x12 > /proc/mtest && dmesg | tail -1
[ 1641.190216] written 0x12 to address 0xffff8000ced9a000
tonychen@ecs-433a:~/CS353/project3/718030290013_陈思贝_project3_src$
```

图 3: writeval 运行结果

3 实验心得

本次实验基于 Linux 内核的内存管理机制。通过对四层内存分页的逐级追踪解出虚拟内存与物理内存地址的对应关系。同时可以根据 vma 的不同变量的值，判断出模块对该块内存的权限状态。通过对内存物理地址的直接操作，可以做到对变量的直接输出与写入。通过本次实验，对 Linux 的内存管

理方式有了更深刻的了解。在更新的内核中，Linux 采用了更先进的五层内存分页，增加了支持的最大内存容量，但其计算原理与四层内存分页也是大同小异的。