
Project 2

陈思贝 (718030290013)

0 实验配置

- Linux Kernel: 5.11.16
- OS: Ubuntu 18.04.5 LTS

1 实验过程

1. 在 `<include/linux/sched.h>` 中合适的位置声明 `ctx`。

`task_struct` 前部变量顺序不能随意更改，因此选择在 `randomized_struct_fields_start` 后声明 `ctx`。注意不要写在 `#ifdef` 和 `#endif` 之间。

```

649     struct task_struct {
        ...
664     randomized_struct_fields_start
665     int          ctx;      // Declare ctx here
        ...
1387 };

```

2. 在 `<kernel/fork.c>` 中创建进程时初始化 `ctx=0`。

参考第 2526 至 2550 行代码，当 `syscall fork` 的时候，调用了 `kernel_clone(&args)`。因此定位至第 2429 行的 `kernel_clone()`，进一步调用了第 1852 行的 `copy_process()`。分析代码可得知在第 2326 行会返回创建成功的子进程，因此只需在此行之前初始化 `ctx` 即可。

```

1852     static __latent_entropy struct task_struct *copy_process(
1853         struct pid *pid,
1854         int trace,
1855         int node,
1856         struct kernel_clone_args *args)
1857     {
        ...
2324     copy_oom_score_adj(clone_flags, p);
2325
2326     p->ctx = 0;  // Initializes ctx to 0
2327     return p;
        ...
2389 }

```

3. 在 `<kernel/sched/core.c>` 中调度时 `ctx++`。

该文件中 `schedule()` 函数负责进程的调度。其中 `tsk` 指向当前进程，因此直接在该函数中添加 `ctx++` 即可。

```

5145 asmlinkage __visible void __sched schedule(void)
5146 {

```

```

5147     struct task_struct *tsk = current;
5148
5149     tsk->ctx++;    // Increments ctx
5150     sched_submit_work(tsk);
5151     ...
5157 }

```

4. 在 <fs/proc/base.c> 中创建 proc entry。

将 ctx 的值输出至 /proc/<pid>/ctx。base.c 文件中的 tgid_base_stuff 记录了目录下创建文件的属性。只需在该数组中添加一条记录即可。因为只需要只读权限，所以 ONE 宏就满足需求了。同时还需添加一条函数，用于定义读取 ctx 值的操作。

```

3159 static int proc_pid_ctx(struct seq_file *m, struct pid_namespace *ns,
3160                        struct pid *pid, struct task_struct *task) // Prints ctx value
3161 {
3162     int err = lock_trace(task);
3163     if (!err) {
3164         seq_printf(m, "%d\n", task->ctx);
3165         unlock_trace(task);
3166     }
3167     return err;
3168 }
3169
3170 static const struct pid_entry tgid_base_stuff[] = {
3171     ...
3280     ONE("ctx", S_IRUSR, proc_pid_ctx), // Creates ctx file
3281 };

```

5. 编译内核。

参照实验零的步骤，对内核再次进行编译与重新引导，耗时约 2 小时。使用 `disown -h` 指令以防突然断开连接前功尽弃。

6. 检查实验效果。

编译安装成功后重启设备，并用如下代码生成进程。

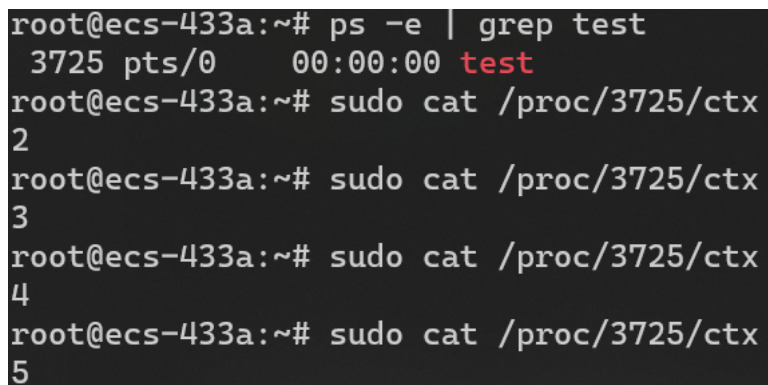
```

1  # include <stdio.h>
2
3  int main() {
4      while (1) getchar();
5      return 0;
6  }

```

再另开一个终端窗口并用 `cat /proc/<pid>/ctx` 查看 `ctx` 的值。截图见图1。

2 实验效果截图



```
root@ecs-433a:~# ps -e | grep test
3725 pts/0    00:00:00 test
root@ecs-433a:~# sudo cat /proc/3725/ctx
2
root@ecs-433a:~# sudo cat /proc/3725/ctx
3
root@ecs-433a:~# sudo cat /proc/3725/ctx
4
root@ecs-433a:~# sudo cat /proc/3725/ctx
5
```

图 1: 运行状态截图

3 实验心得

这次的代码作业需要对 Linux 的内核文件程序仔细地研读。对内核程序的流程和其中定义的宏有了更深的理解。与之前的作业不同，内核程序无法直接修改，更改源文件后，要对内核重新编译，耗时会高达几个小时。所以对于编程准确性要求非常高。同时内核编译中会占用很大的硬盘空间，因此掌握了临时增加磁盘空间的技术。