Programowanie Systemowe

Laboratorium 4 - Debugowanie jądra

Magdalena Pastuła

Zadanie 1 - debugowanie modułów.

Moduł nr 1

Poniżej znajduje się screen części informującej o błędzie przez Oops:

```
[ 2051.215609] broken_module: version magic '4.9.13-200.fc25.x86_64 SMP mod_unload ' should be '4.9.12-200.fc25.x86_64 S
MP mod_unload '
[ 2067.244246] broken_module: loading out-of-tree module taints kernel.
[ 2067.244859] broken_module: module verification failed: signature and/or required key missing - tainting kernel
[ 2067.250542] The BROKEN module has been inserted.
[ 2141.871677] perf: interrupt took too long (9143 > 9078), lowering kernel.perf_event_max_sample_rate to 21000
[ 2348.999679] BUG: unable to handle kernel paging request at fffffc7047682920
[ 2348.999686] IP: [<ffffffffbe230653>] kfree+0x53/0x170
[ 2348.999694] PGD 0

[ 2348.999698] Oops: 0000 [#1] SMP
```

Natomiast następny screen to dalszy ciąg komunikatów z Oops:

```
[ 2348.999751] CPU: 3 PID: 3850 Comm: cat Tainted: G           OE   4.9.12-200.fc25.x86_64 #1
[ 2348.999753] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS 6.00 07/22/202
0
[ 2348.999755] task: ffff9fe5665abd00 task.stack: ffffb4a803134000
[ 2348.999757] RIP: 0010:[<ffffffffbe230653>]  [<ffffffffbe230653>] kfree+0x53/0x170
[ 2348.999761] RSP: 0018:ffffb4a803137df8  EFLAGS: 00010286
[ 2348.999763] RAX: 0000000000000000 RBX: 00007fb71a0a4000 RCX: 0000000000000000
[ 2348.999764] RDX: 0000000000000003 RSI: ffff9fe5759c22a3 RDI: 0000601a40000000
[ 2348.999766] RBP: ffffb4a803137e10 R08: 31203a657a697320 R09: 0000000000000080
[ 2348.999767] R10: fffffc7047682900 R11: ffff9fe5759c22a2 R12: 00007fb71a0a4000
[ 2348.999769] R13: ffffffffc07230c1 R14: 0000000000000000 R15: ffffb4a803137f18
[ 2348.999771] FS:  00007fb71a0c5700(0000) GS:ffff9fe5fa6c0000(0000) knlGS:0000000000000000
[ 2348.999773] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 2348.999774] CR2: fffffc7047682920 CR3: 000000004f7fd000 CR4: 00000000003406e0
[ 2348.999810] Stack:
[ 2348.999812]  0000000000000023 00007fb71a0a4000 ffffb4a803137f18 ffffb4a803137e40
[ 2348.999815]  ffffffffc07230c1 ffff9fe5429bcd00 ffffb4a803137f18 ffff9fe5429bcd00
[ 2348.999818]  00007fb71a0a4000 ffffb4a803137ec8 ffffffffbe258db7 0000000000000000
[ 2348.999821] Call Trace:
[ 2348.999828]  [<ffffffffc07230c1>] broken_read+0xb1/0xe0 [broken_module]
[ 2348.999832]  [<ffffffffbe258db7>] __vfs_read+0x37/0x150
[ 2348.999836]  [<ffffffffbe37316b>] ? security_file_permission+0x9b/0xc0
[ 2348.999839]  [<ffffffffbe259f36>] vfs_read+0x96/0x130
[ 2348.999841]  [<ffffffffbe25b425>] SyS_read+0x55/0xc0
[ 2348.999845]  [<ffffffffbe81dc37>] entry_SYSCALL_64_fastpath+0x1a/0xa9
[ 2348.999847] Code: 00 80 49 01 da 0f 82 27 01 00 00 48 c7 c7 00 00 00 80 48 2b 3d c7 2e c0 00 49 01 fa 49 c1 ea 0c 49
c1 e2 06 4c 03 15 a5 2e c0 00 <49> 8b 42 20 48 8d 50 ff a8 01 4c 0f 45 d2 49 8b 52 20 48 8d 42
[ 2348.999881] RIP  [<ffffffffbe230653>] kfree+0x53/0x170
[ 2348.999885]  RSP <ffffb4a803137df8>
[ 2348.999886] CR2: fffffc7047682920
[ 2348.999889] ---[ end trace 9c215688d06aef5b ]---
[ 2481.484596] perf: interrupt took too long (11454 > 11428), lowering kernel.perf_event_max_sample_rate to 17000
```

Jak można zauważyć, błąd sygnalizowany przez Oops to problem z dostępem do strony w pamięci przestrzeni jądra. Na drugim screenie RIP podpowiada, że błąd wystąpił przy użyciu funkcji kfree pod adresem 0x170 w segmencie 0x53, a z zapisów Call trace można wyczytać, że błąd wystąpił w module broken_module w funkcji broken_read. Poniżej znajduje się kod funkcji, w której wystąpił błąd.

```
ssize_t broken_read(struct file *filp, char *user_buf, size_t count,
    loff_t *f_pos)
{
    char *mybuf = NULL;
```

```
        int mybuf_size = 100;
        int len, err;

        mybuf = kmalloc(mybuf_size, GFP_KERNEL);
        if (!mybuf) {
            return -ENOMEM;
        }

        fill_buffer(mybuf, mybuf_size);

        len = strlen(mybuf);
        err = copy_to_user(user_buf, mybuf, len);
        kfree(user_buf);

        read_count++;

        if (!err && *f_pos == 0) {
            *f_pos += len;
            return len;
        }
        return 0;
    }
```

Po przeanalizowaniu kodu okazuje się, że błędem jest próba zwolnienia pamięci z przestrzeni użytkownika, do której procesy z przestrzeni jądra nie mają bezpośredniego dostępu, co wyjaśnia problem z dostępem do pamięci zgłoszony w Oops. Po zmianie fragmentu `kfree(user_buf)` na `kfree(mybuf)` moduł już działa poprawnie i nie zgłasza żadnych błędów. Poniżej znajduje się screen komunikatów jądra po załadowaniu wykonaniu podanych instrukcji i usunięciu poprawionego modułu, a także screen komunikatu po wywołaniu komendy 'cat /dev/broken`.

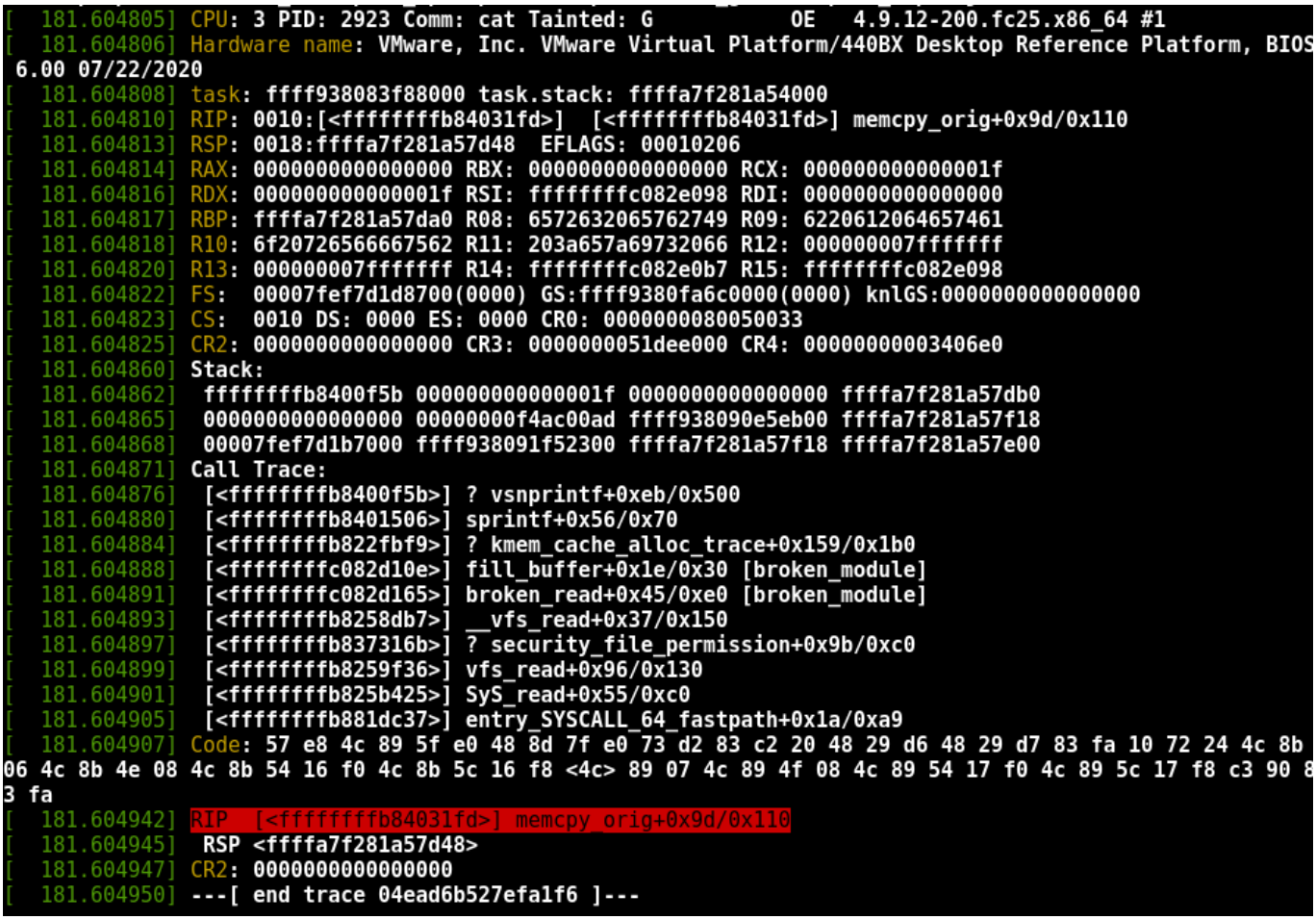
```
[   15.472681] fuse init (API version 7.26)
[  486.813022] broken_module: loading out-of-tree module taints kernel.
[  486.813130] broken_module: module verification failed: signature and/or required key missing - taintin
g kernel
[  486.816026] The BROKEN module has been inserted.
[  724.843738] The BROKEN module has been removed
```

```
/dev » cat /dev/broken
I've created a buffer of size: 100
```

Moduł nr 2

Poniżej znajduje się screeny przedstawiające komunikaty Oops po załadowaniu i uruchomieniu modułu drugiego.

```
[   83.262675] broken_module: loading out-of-tree module taints kernel.
[   83.262724] broken_module: module verification failed: signature and/or required key missing - taintin
g kernel
[   83.263473] The BROKEN module has been inserted.
[  181.604741] BUG: unable to handle kernel NULL pointer dereference at           (null)
[  181.604748] IP: [<ffffffffb84031fd>] memcpy_orig+0x9d/0x110
[  181.604754] PGD 0

[  181.604758] Oops: 0002 [#1] SMP
```



Podobnie jak w poprzednim module, w tym również błąd pojawił się w funkcji `broken_read` oraz `fill_buffer`, jednakże tym razem sygnalizowanym błędem jest problem w dereferencji wskaźnika. Poniżej znajduje się kod funkcji `broken_read` oraz `fill_buffer`.

```
int fill_buffer(char *buf, int buf_size)
{
    sprintf(mybuf, "I've created a buffer of size: %d\n", buf_size);
    return strlen(mybuf);
}

ssize_t broken_read(struct file *filp, char *user_buf, size_t count,
    loff_t *f_pos)
{
    char *buf;
    int buf_size = 100;
    int len, err;

    buf = kmalloc(buf_size, GFP_KERNEL);
    if (buf == 0) {
        return -ENOMEM;
    }
    fill_buffer(buf, buf_size);

    len = strlen(buf);
    err = copy_to_user(user_buf, buf, len);
    kfree(buf);
```

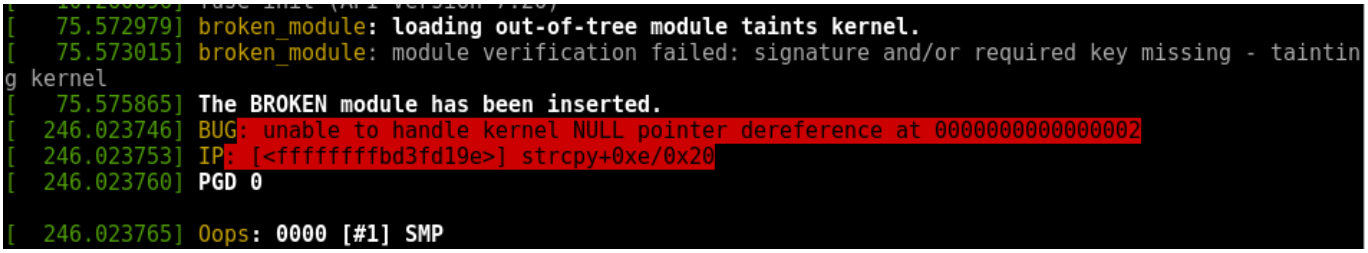
```
    read_count++;


    if (!err && *f_pos == 0) {
        *f_pos += len;
        return len;
    }
    return 0;
}
```

Jak widać, sama funkcja `broken_read` nie zawiera żadnego błędu pod względem referencji do niezaalokowanego wskaźnika. Natomiast sytuacja taka ma miejsce w funkcji `fill_buffer`, która do funkcji sprintf jako argument przekazuje globalny wskaźnik `mybuf`, który jest niezaalokowany, zamiast przekazanego argumentu. Również linijkę niżej do funkcji strlen jako argument przekazany jest zły wskaźnik.

Po poprawieniu błędu i ponownym uruchomieniu modułu otrzymujemy komunikaty wypisywane po wywołaniu komend `dmesg` i `cat /dev/broken` są takie same jak w poprzednim zadaniu.

Moduł nr 3

Poniżej znajduje się screeny przedstawiające komunikaty Oops po załadowaniu i uruchomieniu modułu drugiego.



```
[  246.023817] CPU: 3 PID: 2957 Comm: bash Tainted: G           OE   4.9.12-200.fc25.x86_64 #1
[  246.023819] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS
 6.00 07/22/2020
[  246.023821] task: ffff9128dead3d00 task.stack: ffffb378c2eec000
[  246.023823] RIP: 0010:[<ffffffffbd3fd19e>]  [<ffffffffbd3fd19e>] strcpy+0xe/0x20
[  246.023827] RSP: 0018:ffffb378c2eefdf8  EFLAGS: 00010206
[  246.023829] RAX: ffff91290e495000 RBX: 0000000000000002 RCX: 0000000000000034
[  246.023830] RDX: ffff91290e495000 RSI: 0000000000000003 RDI: ffff91290e495000
[  246.023832] RBP: ffffb378c2eefdf8 R08: 0000000000000002 R09: 0000000000000001
[  246.023833] R10: 000000000000000a R11: f000000000000000 R12: 0000000000000002
[  246.023835] R13: 000055bc678735d0 R14: ffff91290e495000 R15: 0000000000000000
[  246.023837] FS:  00007f8a1bd2d700(0000) GS:ffff91293a6c0000(0000) knlGS:0000000000000000
[  246.023839] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  246.023840] CR2: 0000000000000002 CR3: 0000000089e26000 CR4: 00000000003406e0
[  246.023877] Stack:
[  246.023879]  ffffb378c2eefe10 ffffffffc0857304 0000000000000002 ffffb378c2eefe40
[  246.023883]  ffffffffc0857398 ffff912909fdb300 ffffb378c2eeff18 ffff912909fdb300
[  246.023886]  000055bc678735d0 ffffb378c2eefec8 ffffffffbd259747 0000000000000002
[  246.023888] Call Trace:
[  246.023895]  [<ffffffffc0857304>] fill_buffer_with_process_name+0x34/0x50 [broken_module]
[  246.023898]  [<ffffffffc0857398>] broken_write+0x78/0xce0 [broken_module]
[  246.023902]  [<ffffffffbd259747>] __vfs_write+0x37/0x160
[  246.023906]  [<ffffffffbd37d577>] ? selinux_file_permission+0xd7/0x110
[  246.023910]  [<ffffffffbd37310b>] ? security_file_permission+0x3b/0xc0
[  246.023912]  [<ffffffffbd25a085>] vfs_write+0xb5/0x1a0
[  246.023915]  [<ffffffffbd25b4e5>] SyS_write+0x55/0xc0
[  246.023919]  [<ffffffffbd81dc37>] entry_SYSCALL_64_fastpath+0x1a/0xa9
[  246.023921] Code: ff 89 ca 4c 63 c9 83 c2 20 43 f6 04 08 01 0f b6 d2 0f 45 ca 39 c8 74 c2 29 c8 5d c3
90 55 48 89 f8 48 89 fa 48 89 e5 48 83 c6 01 <0f> b6 4e ff 48 83 c2 01 84 c9 88 4a ff 75 ed 5d c3 90 55 4
8 85
[  246.023954] RIP  [<ffffffffbd3fd19e>] strcpy+0xe/0x20
[  246.023957]  RSP <ffffb378c2eefdf8>
[  246.023959] CR2: 0000000000000002
[  246.023962] ---[ end trace b0685bd200fdf4e1 ]---
```

W tym przypadku błąd sugerowany przez Oops to dereferencja wskaźnika równego `NULL` podczas użycia funkcji `strcpy` w funkcji `broken_write` i `fill_buffer_with_process_name`. Poniżej znajduje się kod tych dwóch funcji.

```
void fill_buffer_with_process_name(long pid)
{
    struct pid *selected_pid = find_get_pid(pid);
    struct task_struct *selected_proc = pid_task(selected_pid,
PIDTYPE_PID);

    if (selected_proc != NULL)
        strcpy(buf1, (char *) selected_proc->pid);
    else
        sprintf(buf1, "The process with PID: %ld cannot be found", pid);
}

ssize_t broken_write(struct file *filp, const char *buf, size_t count,
             loff_t *f_pos)
{
    int error = 0;
    long pid = 0;
    int copy_size = count;

    if (count > buf1_size)
        copy_size = buf1_size - 1;

    error = copy_from_user(buf1, buf, copy_size);
```

```
        buf1[copy_size] = 0;

        pid = simple_strtol(buf1, buf1 + copy_size, 10);

        if (pid < 1)
            printk(KERN_WARNING "Invalid PID number\n");
        else
            fill_buffer_with_process_name(pid);

        write_count++;
        return copy_size;
    }
```

Jak widać, błąd znajduje się w funkcji `fill_buffer_with_process_name` przy użyciu funkcji `strcpy`: podany jako drugi argument pid jest liczbą całkowitą, a nie napisem, zatem po rzutowaniu na wskaźnik na char otrzymujemy wskaźnik na niekontrolowany fragment pamięci. Ponieważ przekazany pid był równy 2, to program próbuje dostać się do adresu równego 2, co widać w komunikacie Oops.

Moduł nr 4

Poniżej znajduje się screeny przedstawiające komunikaty Oops po załadowaniu i uruchomieniu modułu drugiego.

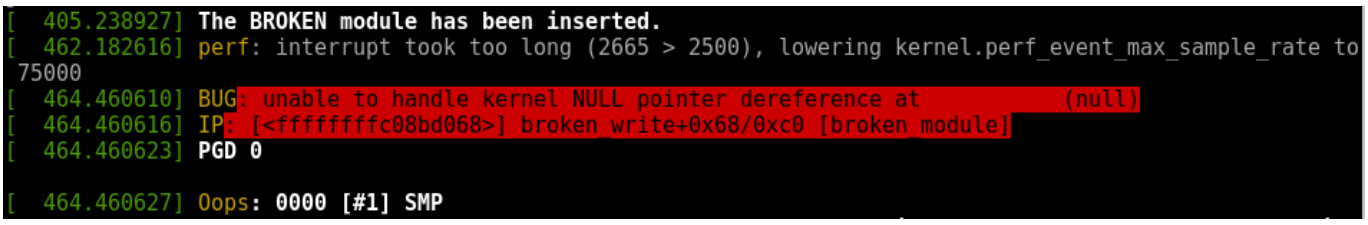
```
[  115.308525] The BROKEN module has been inserted.
[  142.469407] hrtimer: interrupt took 3542602 ns
[  218.929798] perf: interrupt took too long (2522 > 2500), lowering kernel.perf_event_max_sample_rate to
 79000
[  294.940348] usercopy: kernel memory overwrite attempt detected to          (null) (<null>) (8 bytes)
[  294.940394] ------------[ cut here ]------------
[  294.940396] kernel BUG at mm/usercopy.c:75!
[  294.940399] invalid opcode: 0000 [#1] SMP
```

```
[  294.940453] CPU: 2 PID: 3327 Comm: bash Tainted: G           OE   4.9.12-200.fc25.x86_64 #1
[  294.940454] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS
 6.00 07/22/2020
[  294.940457] task: ffff93bb1dd78000 task.stack: ffffbc4f035b0000
[  294.940458] RIP: 0010:[<ffffffff8d2556c7>]  [<ffffffff8d2556c7>] __check_object_size+0x77/0x1d6
[  294.940467] RSP: 0018:ffffbc4f035b3df8  EFLAGS: 00010282
[  294.940468] RAX: 0000000000000059 RBX: 0000000000000000 RCX: 0000000000000000
[  294.940470] RDX: 0000000000000000 RSI: ffff93bbba68e0a8 RDI: ffff93bbba68e0a8
[  294.940471] RBP: ffffbc4f035b3e18 R08: 2820296c6c756e28 R09: 20293e6c6c756e3c
[  294.940473] R10: ffff93bba2e9cb80 R11: 0000000000000667 R12: 0000000000000008
[  294.940474] R13: 0000000000000000 R14: 0000000000000008 R15: 0000000000000000
[  294.940476] FS:  00007f0a2edbd700(0000) GS:ffff93bbba680000(0000) knlGS:0000000000000000
[  294.940478] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  294.940480] CR2: 00005589e82b1820 CR3: 00000000859a1000 CR4: 00000000003406e0
[  294.940518] Stack:
[  294.940521]  0000000000000008 0000000000000008 00005589e83845c0 00005589e83845c0
[  294.940525]  ffffbc4f035b3e40 ffffffffc07b1044 ffff93bb15ec8c00 ffffbc4f035b3f18
[  294.940528]  ffff93bb15ec8c00 ffffbc4f035b3ec8 ffffffff8d259747 0000000000000002
[  294.940531] Call Trace:
[  294.940539]  [<ffffffffc07b1044>] broken_write+0x44/0xb0 [broken_module]
[  294.940542]  [<ffffffff8d259747>] __vfs_write+0x37/0x160
[  294.940546]  [<ffffffff8d37d577>] ? selinux_file_permission+0xd7/0x110
[  294.940550]  [<ffffffff8d37310b>] ? security_file_permission+0x3b/0xc0
[  294.940553]  [<ffffffff8d25a085>] vfs_write+0xb5/0x1a0
[  294.940555]  [<ffffffff8d25b4e5>] SyS_write+0x55/0xc0
[  294.940559]  [<ffffffff8d81dc37>] entry_SYSCALL_64_fastpath+0x1a/0xa9
[  294.940561] Code: 48 0f 44 d1 48 c7 c6 6f a5 c5 8d 48 c7 c1 5b e5 c4 8d 48 0f 44 f1 4d 89 e1 49 89 c0
48 89 d9 48 c7 c7 d0 69 c5 8d e8 78 ad f6 ff <0f> 0b e8 32 9e fb ff 85 c0 75 72 48 89 df e8 96 41 e1 ff 8
4 c0
[  294.940594] RIP  [<ffffffff8d2556c7>]  __check_object_size+0x77/0x1d6
[  294.940598]  RSP <ffffbc4f035b3df8>
[  294.940602] ---[ end trace 8121a90746ed4f4a ]---
```

W przypadku tego modułu Oops zwraca informację, że nastąpił błąd w pliku `usercopy.c`, co może oznaczać, że wystąpił błąd podczas kopiowania danych z przestrzeni użytkownika. Dodatkowo, błąd miał się pojawić w funkcji `broken_write`. Poniżej znajduje się kod tej funkcji.

```
ssize_t broken_write(struct file *filp, const char *user_buf, size_t count,
    loff_t *f_pos)
{
    char *mybuf = NULL;
    int mybuf_size = 100;
    int real_count = count;
    int err;

    // Initialize the memory
    kmalloc(mybuf_size, GFP_KERNEL);

    // Take the max(mybuf_size, count)
    if (real_count > mybuf_size)
        real_count = mybuf_size;

    // Copy the buffer from user space
    err = copy_from_user(mybuf, user_buf, real_count);
    mybuf[mybuf_size] = 0;

    if (!err && real_count > 0) {
        // Count number of digits in buffer
        numbers_count = count_numbers(mybuf);
    } else {
        printk(KERN_WARNING "BROKEN: error occured in write function");
    }

    // Free the memory
    if (mybuf != NULL)
        kfree(mybuf);

    write_count++;
    return real_count;
}
```

Po przeanalizowaniu tego kodu okazuje się, że faktycznie nastąpił tu błąd. Wprawdzie używana jest funkcja kmalloc, ale jej wynik jest ignorowany, zatem wskaźnik mybuf przekazany do funkcji kopiowania z przestrzeni użytkownika jest równy `NULL`. Po poprawieniu tego błędu i ponownym uruchowieniu modułu otrzymujemy następujące komunikaty Oops:



```
[  464.460710] task: ffff8db45df6bd00 task.stack: ffffac9f02f84000
[  464.460711] RIP: 0010:[<ffffffffc08bd068>]  [<ffffffffc08bd068>] broken_write+0x68/0xc0 [broken_module
]
[  464.460716] RSP: 0018:ffffac9f02f87e20  EFLAGS: 00010202
[  464.460717] RAX: 0000000000000000 RBX: 0000000000000008 RCX: 0000000000000000
[  464.460719] RDX: 0000000000000000 RSI: 0000562aaeba55d8 RDI: ffff8db47639b308
[  464.460720] RBP: ffffac9f02f87e40 R08: 0a33323164636261 R09: 0000000000000080
[  464.460721] R10: 0000000000000080 R11: ffff8db45df6bd00 R12: 0000000000000008
[  464.460723] R13: ffff8db47639b300 R14: 0000562aaeba55d0 R15: 0000000000000000
[  464.460725] FS:  00007f85da586700(0000) GS:ffff8db4fa6c0000(0000) knlGS:0000000000000000
[  464.460726] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  464.460727] CR2: 0000000000000000 CR3: 0000000060387000 CR4: 00000000003406e0
[  464.460758] Stack:
[  464.460759]  ffff8db45db59f00 ffffac9f02f87f18 ffff8db45db59f00 0000562aaeba55d0
[  464.460762]  ffffac9f02f87ec8 ffffffff83259747 0000000000000002 ffffac9f02f87e90
[  464.460765]  ffffffff8337d577 ffffffff83ea1140 ffff8db45db59f00 0000000000000002
[  464.460767] Call Trace:
[  464.460773]  [<ffffffff83259747>] __vfs_write+0x37/0x160
[  464.460777]  [<ffffffff8337d577>] ? selinux_file_permission+0xd7/0x110
[  464.460780]  [<ffffffff8337310b>] ? security_file_permission+0x3b/0xc0
[  464.460782]  [<ffffffff8325a085>] vfs_write+0xb5/0x1a0
[  464.460784]  [<ffffffff8325b4e5>] SyS_write+0x55/0xc0
[  464.460788]  [<ffffffff8381dc37>] entry_SYSCALL_64_fastpath+0x1a/0xa9
[  464.460790] Code: 89 ef 31 d2 49 63 dc 48 89 de e8 04 86 99 c2 44 89 e2 4c 89 f6 4c 89 ef e8 76 4b b4
c2 85 c0 41 c6 45 64 00 75 4f 45 85 e4 7e 4a <0f> b6 14 25 00 00 00 00 31 c9 31 f6 eb 12 48 83 c1 01 0f b
e 01
[  464.460818] RIP  [<ffffffffc08bd068>] broken_write+0x68/0xc0 [broken_module]
[  464.460821]  RSP <ffffac9f02f87e20>
[  464.460822] CR2: 0000000000000000
```

Okazało się, że w module znajduje się następny błąd w funkcji `count_numbers`, której kod znajduje się poniżej. Ponownie występuje tutaj odwołanie się do wskaźnika, który jest równy `NULL` i który nie jest przekazanym argumentem.

```
int count_numbers(char *str)
{
        int numbers = 0;
        char *ptr = 0;

        while (*ptr != 0) {
                ptr++;
        if (isdigit(*ptr))
                        numbers++;
        }

        return numbers;
}
```

Po naprawieniu tego błędu moduł działał już poprawnie.

Zadanie 2 - GDB.

Zadanie nie zostało wykonane.