A Year Fuzzing XNU Mach IPC

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.

✅ Found some vulnerabilities in Apple from userland to kernel-level vulnerabilities

Outline

1. Previous XNU Mach IPC Vulnerabilities
2. Template-based XNU IPC Fuzzer
3. CVE-2022-32894
4. Q&A
5. References

1. Previous XNU Mach IPC Vulnerabilities

CVE-2020-27932 (1/2)

Issue 2107: XNU kernel type confusion in turnstiles

Reported by ianbeer@google.com on Fri, Oct 30, 2020, 2:56 AM GMT+8 Project Member

```
union {
        ipc_kobject_t kobject;                  <-- a kobject (like iokit userclient, host_notify port)
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;         <-- the destination port to which a special_reply_port was sent
        struct knote *sync_inheritor_knote;
        struct turnstile *sync_inheritor_ts;
} kdata;
```

} kdata;

There are a couple of weird edges cases which the turnstile code overlooked; specifically it's possible to turn a regular, user-owned port into a port with a kobject! Specifically, by calling host_request_notification on a send_once right we can change the type of a port to IKOT_HOST_NOTIFY and get the kobject field set to a struct host_notify_entry pointer.

CVE-2020-27932 (2/2)

```
101         entry = (host_notify_t)zalloc(host_notify_zone);
102         if (entry == NULL) {
                return KERN_RESOURCE_SHORTAGE;

        ip_lock(port);
        if (!ip_active(port) || port->ip_tempowner || port->ip_specialreply ||
            ip_is_kolabeled(port) || ip_kotype(port) != IKOT_NONE) {
                ip_unlock(port);

                lck_mtx_unlock(&host_notify_lock);
                zfree(host_notify_zone, entry);

                return KERN_FAILURE;
        }

117
118         entry->port = port;
119         ipc_kobject_set_atomically(port, (ipc_kobject_t)entry, IKOT_HOST_NOTIFY);
120         ip_unlock(port);
```

Added the check

assign port->kobject with entry pointer

CVE-2021-1782

Before the patch

patch in xnu 7195.81.3

```
switch (command) {
case MACH_VOUCHER_ATTR_REDEEM:

        /* redeem of previous values is the value */
        if (0 < prev_value_count) {
                elem = (user_data_element_t)prev_values[0];

                user_data_lock();
                assert(0 < elem->e_made);
                elem->e_made++;
                user_data_unlock();

                *out_value = (mach_voucher_attr_value_handle_t)elem;
                return KERN_SUCCESS;
        }
```

added the lock to prevent race conditon

Key takeaways

1. Mach IPC subsys eatures which is not easy.
2. ***The union* keywo** n mach port structure to store multiple typ

➔ ***These could lead NU kernel.***

2. Template-based XNU IPC Fuzzer

What are the three challenges ?

1. **Generating** test-case using **MACH IPC syscalls and their arguments in C language is very hard.**
2. The **code-base is huge**. In order to **reach these code paths**, you have to **satisfy may conditions**.
3. Mutating **Mach IPC syscalls and their arguments are not enough to expand the code coverage** and hard to develop.

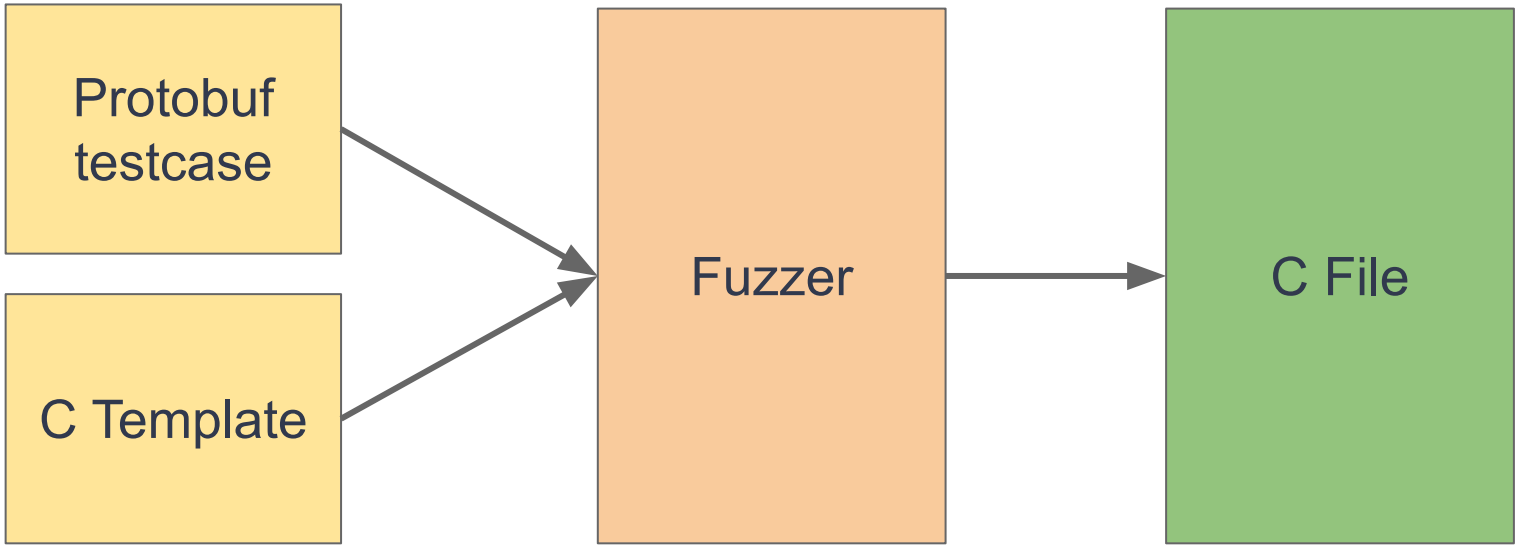
➔ ***We need more advanced fuzzer which has ability to generate complex test-case and reach in deeper in Mach IPC.***

What are template-based fuzzer?

My expectations

- The template based fuzzer have to **easily to modify and edit template in C language**.
- The template design must be **easy to parse and process** (less coding 😃).
- The template able to add some **custom functions beside mach ipc syscall functions**.

Fuzzer Architecture



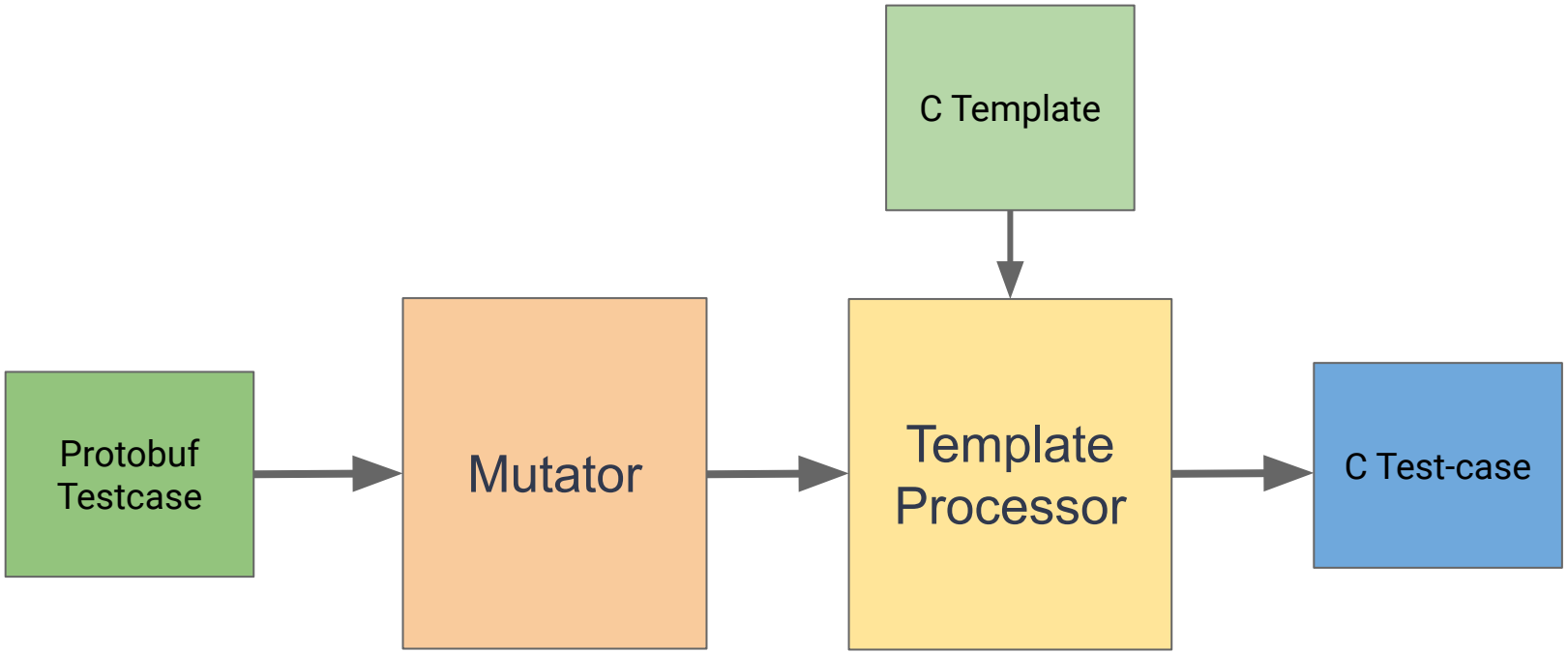
Fuzzer Architecture: Protobuf test-case

```
message mach_msg {
    string msgh_bits_remote = 1;
    string msgh_bits_local = 2;
    string msgh_bits_voucher = 3;
    string msgh_bits_other = 4;
    uint64 msgh_size = 5;
    uint32 msgh_remote_port = 6;
    uint32 msgh_local_port = 7;
    uint64 msgh_voucher_port = 8;
    int32  msgh_id = 9;
    uint64 msgh_descriptor_count = 10;
    repeated mach_msg_descriptor descriptors = 11;
    bytes inline_data = 12;
}
```

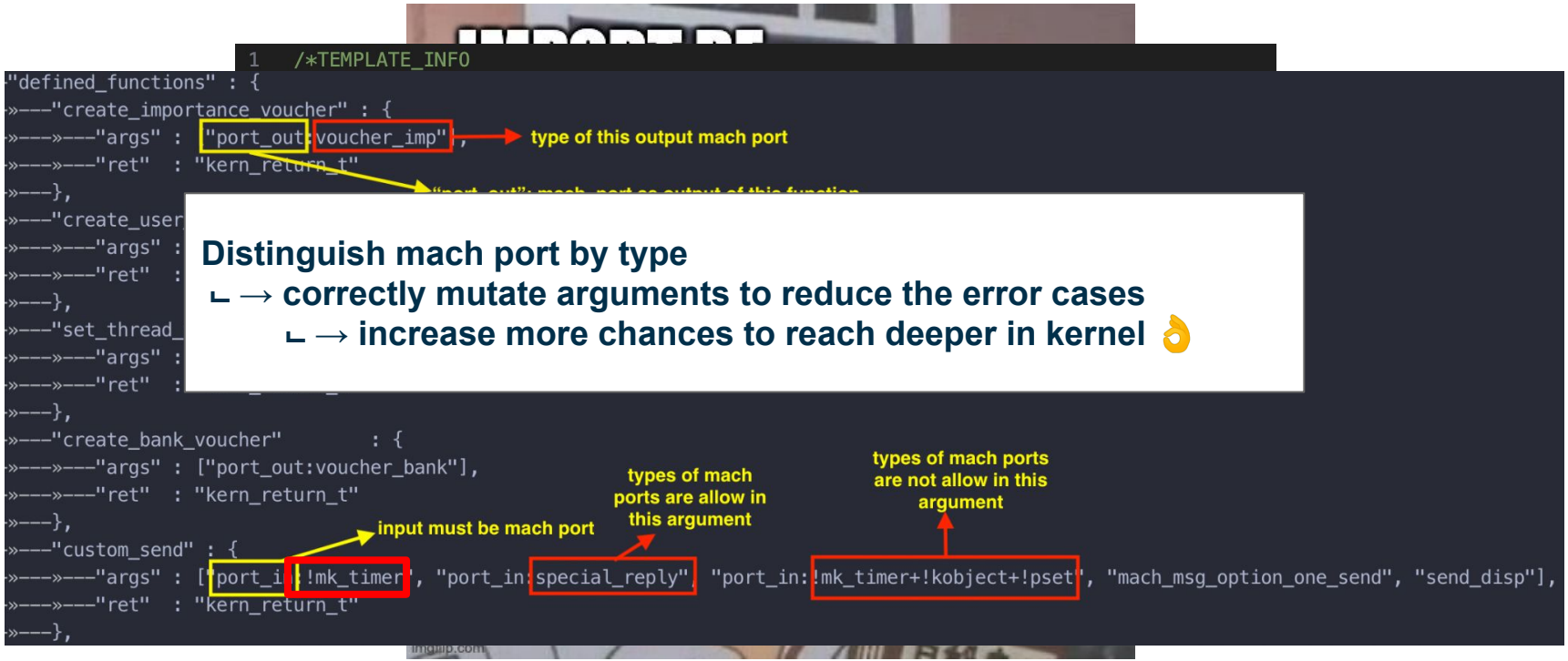
Fuzzer Architecture: C Template format

```
86  kern_return_t create_user_data_voucher(mach_port_t *port)
196 /*port_declare*/
197
198 void execute()
199 {
200     /*GENERATE_CODE*/
201 }
202
203 int main()
204 {
205     mach_msg_type_number_t my_size;
206     /*port_init*/
207
208     execute();
209
210     return 0;
211 }
```

Fuzzer Architecture: The Fuzzer



Template Processor (1/2)



Template Processor (2/2)

```
164        if template_info.get('defined_vars'):
```

```
273 port_25 = thread_get_special_reply_port();
274 port_26 = mach_thread_self();
275 mach_port_destruct(task_port_1, port_26, -1, 4660318827490212632);
276 kr = custom_send(port_3, port_25, port_2, port_3, MACH_SEND_SYNC_BOOTSTRAP_CHECKIN, MACH_MSG_TYPE_MAKE_SEND);
277 printf("[%s:%d] custom_send: %d (%s)\n", __FUNCTION__,  __LINE__, kr, mach_error_string(kr));
278 kr = custom_send(port_2, port_25, port_2, port_16, MACH_SEND_PROPAGATE_QOS, MACH_MSG_TYPE_MOVE_SEND_ONCE);
279 printf("[%s:%d] custom_send: %d (%s)\n", __FUNCTION__,  __LINE__, kr, mach_error_string(kr));
280 kr = mach_port_deallocate(task_port_1, port_9);
281 printf("[%s:%d] mach_port_deallocate: %d (%s)\n", __FUNCTION__,  __LINE__, kr, mach_error_string(kr));
282 kr = thread_set_mach_voucher(mach_thread_self(), port_2);
283 printf("[%s:%d] thread_set_mach_voucher: %d (%s)\n", __FUNCTION__,  __LINE__, kr, mach_error_string(kr));
284 basic_msg.header.msgh_remote_port   = port_7;
285 basic_msg.header.msgh_local_port    = port_13;
286 basic_msg.header.msgh_voucher_port  = MACH_PORT_NULL;
287 basic_msg.header.msgh_bits          = MACH_MSGH_BITS_SET(MACH_MSG_TYPE_MOVE_SEND_ONCE, MACH_MSG_TYPE_MOVE_SEND_ONCE, 0, 0);
288 basic_msg.header.msgh_id            = 24801;
289 basic_msg.header.msgh_size          = sizeof(basic_msg) - sizeof(mach_msg_trailer_t);
290 kr = mach_msg((mach_msg_header_t *)&basic_msg, MACH_SEND_MSG|MACH_SEND_TIMEOUT|MACH_RCV_MSG|MACH_RCV_TIMEOUT|MACH_RCV_LARGE_IDENTITY|
    IFY|MACH_RCV_SYNC_WAIT|MACH_RCV_VOUCHER, basic_msg.header.msgh_size, sizeof(basic_msg), port_10, 4000, port_26);
291 printf("[%s:%d] mach_msg: %d (%s)\n", __FUNCTION__,  __LINE__, kr, mach_error_string(kr));
```

Mutator

Mutator strategies:

1. Mutating **function arguments** base on **their types** (defined in C Template).
2. Mutating the **order of calling these functions**.
3. Mutating function **bases on their probabilities** defined in C Template.
4. Mutating the **order of each threads** to focus on **race condition cases**.

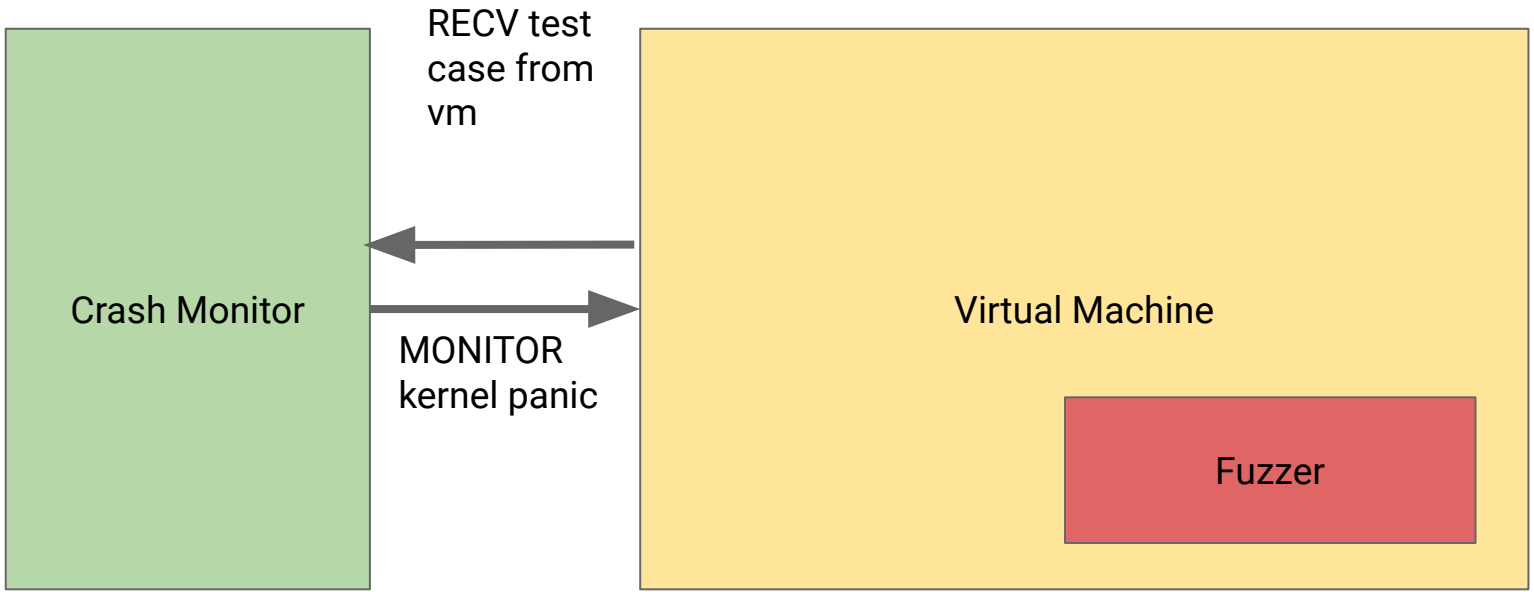
Mutator

```
for idx, arg_type_name in enumerate(defined_funcs[func_name]['args']):
    func_arg = ExtProtoArgument()
    func_arg.arg_type_name = arg_type_name
```

```
1078        if template.is_threads_binding:
1079            # this template has fixed number of thread.
1080            n_thread = len(template.threads_binding)
1081        else:
1082            n_thread = random.randint(*DEFAULT_NUM_THREAD_RANGE)
1083
1084        for _ in range(n_thread):
1085            # perform mutation for each thread
1086            self.mutator_thread(template, testcase)
```

```
        recipe = None
```

How do I handle kernel panic? (1/2)



How do I handle kernel panic? (2/2)

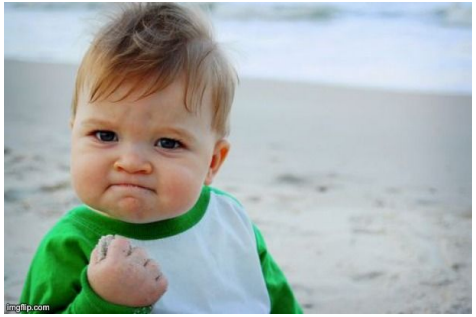
```
**** [IOBluetoothHCIUserClient][SetBTLPOffWL] -- mBluetoothFamily->mACPIMethods->SetBTLP (OFF) returned 0xE0000
2C7 error
**** [IOBluetoothHCIUserClient][SetBTRSWL] -- mBluetoothFamily->mACPIMethods->SetBTRS() returned 0xE00002C7 err
or
Panic(CPU 0, time 1686178019578498): NMIPI for spinlock acquisition timeout, spinlock: 0xffffff853190e9b8, spin
lock owner: 0xffffff85311bc2c0, current_thread: 0xffffff85311bc2c0, spinlock_owner_cpu: 0x0
RAX: 0x00000000000000de, RBX: 0x0000000000000000, RCX: 0x0000000000000830, RDX: 0x0000000000000001
RSP: 0xffffffd019d69d90, RBP: 0xffffffd019d69d90, RSI: 0x00000000000000de, RDI: 0x0000000000000001
R8:  0xffffff85311bc2c0, R9:  0xffffff8018337a80, R10: 0x0000000000000051, R11: 0xffffffd0153b6088
R12: 0x00000000000000d0, R13: 0xffffffd015558818, R14: 0x0000000000000001, R15: 0x00000000000000de
RFL: 0x0000000000000006, RIP: 0xffffff80174e12af, CS:  0x0000000000000008, SS:  0x0000000000000010
Panicked task 0xffffff85311a16c8: 147 threads: pid 0: kernel_task
Backtrace (CPU 0), panicked thread: 0xffffff85311bc2c0, Frame : Return Address
0xffffffd019d69be0 : 0xffffff80174d735a mach_kernel : _interrupt + 0x18a
0xffffffd019d69c90 : 0xffffff80172c7d69 mach_kernel : int_from_intstack + 0x38
0xffffffd019d69d90 : 0xffffff80174e2f5a mach_kernel : _lapic_send_ipi + 0x6a
0xffffffd019d69dc0 : 0xffffff80174e5a5a mach_kernel : _cpu_interrupt + 0x5a
0xffffffd019d69de0 : 0xffffff801735f454 mach_kernel : _thread_go + 0xe4
0xffffffd019d69e10 : 0xffffff80173a95b5 mach_kernel : _waitq_wakeup64_thread + 0xf5
0xffffffd019d69e60 : 0xffffff8017be01fa mach_kernel : __ZN10IOWorkLoop19signalWorkAvailableEv + 0x3a
0xffffffd019d69e80 : 0xffffff8017be232b mach_kernel : __ZN22IOInterruptEventSource23normalInterruptOccurredEPvP
9IOServicei + 0x7b
0xffffffd019d69ea0 : 0xffffff801a070b2e com.apple.iokit.IOPCIFamily : __ZN32IOPCIMessagedInterruptController15h
andleInterruptEPvP9IOServicei + 0x158
0xffffffd019d69ef0 : 0xffffff80185a0cc2 com.apple.driver.AppleACPIPlatform : __ZN23AppleACPIPlatformExpert23dis
patchGlobalInterruptEi + 0x2e
0xffffffd019d69f00 : 0xffffff80185a9e59 com.apple.driver.AppleACPIPlatform : __ZN31AppleACPICPUInterruptControl
```

Fuzzing Results

- My fuzzer is written in Python around 2500 lines of code .
- Auditing XNU kernel to create template and prepare for fuzzing and I have found a lot of crash after few days.
- Found a lot of crashes. But there are some valuable crashes: CVE-2022-32894, CVE-2023-32441 (which was patched in iOS 16.6).

Key takeaways

1. **The idea** of fuzzing is the most important factor to discover security vulnerabilities.
2. Stick with **code auditing** beside with fuzzing to improve the mutator.
3. Have a **deep knowledge about the target** before fuzzing and you will find your own bugs.



3. CVE-2022-32894

iOS 15.6.1 and iPadOS 15.6.1

Released August 17, 2022

Kernel

Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

Impact: An application may be able to execute arbitrary code with kernel privileges. Apple is aware of a report that this issue may have been actively exploited.

Description: An out-of-bounds write issue was addressed with improved bounds checking.

CVE-2022-32894: an anonymous researcher

Fuzzing Results

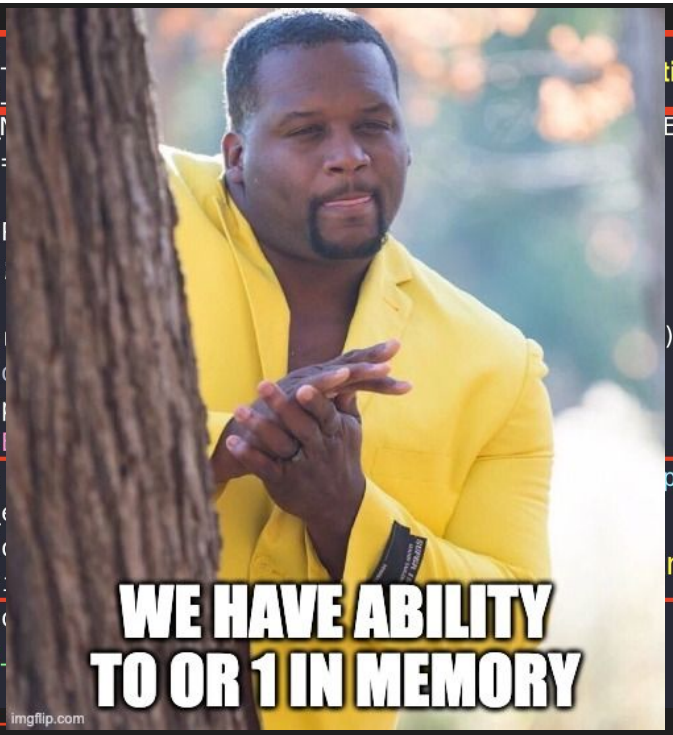
```
panic(cpu 2 caller 0xffffff801b0b523b): ipc_port.c:461 Assertion failed: table != IPR_NULL
Panicked task 0xffffff8542de96c8: 1 threads: pid 44699: table_null2
Backtrace (CPU 2), panicked thread: 0xffffff85358f3c20, Frame : Return Address
0xffffffd016dfb9e0 : 0xffffff801a73168d mach_kernel : _handle_debugger_trap + 0x3cd
0xffffffd016dfba30 : 0xffffff801a8e9df7 mach_kernel : _kdp_i386_trap + 0x177
0xffffffd016dfba70 : 0xffffff801a8d7dae mach_kernel : _kernel_trap + 0x3de
0xffffffd016dfbae0 : 0xffffff801a6c7a60 mach_kernel : trap_from_kernel + 0x26
0xffffffd016dfbb00 : 0xffffff801a731aea mach_kernel : _DebuggerTrapWithState + 0xba
0xffffffd016dfbc20 : 0xffffff801a731139 mach_kernel : _panic_trap_to_debugger + 0x2e9
0xffffffd016dfbc80 : 0xffffff801b0b528f mach_kernel : _panic + 0x54
0xffffffd016dfbcf0 : 0xffffff801b0b523b mach_kernel : _panic
0xffffffd016dfbd00 : 0xffffff801a70e914 mach_kernel : _ipc_port_request_sparm + 0x124
0xffffffd016dfbd20 : 0xffffff801a708229 mach_kernel : _ipc_kmsg_copyin_from_user + 0x1f19
0xffffffd016dfbe50 : 0xffffff801a721a8d mach_kernel : _mach_msg_overwrite_trap + 0x2ad
0xffffffd016dfbee0 : 0xffffff801a8b8153 mach_kernel : _mach_call_munger64 + 0x273
0xffffffd016dfbfa0 : 0xffffff801a6c8246 mach_kernel : _hndl_mach_scall64 + 0x16

Process name corresponding to current thread (0xffffff85358f3c20): table_null2
Boot args: -v keepsyms=1 tlbto_us=0 -zc zlog1=default.kalloc.768 zlog2=default.kalloc.256 vti=9 debug=0x814e kext-dev-mode=1 kcsuffix=de
velopment kdp_match_name=serial

Mac OS version:
21A559

Kernel version:
Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:22 PDT 2021; root:xnu-8019.41.5~1/DEVELOPMENT_X86_64
Kernel UUID: 212C1F60-0EC2-3E01-B533-F035E8B52DD7
```

Root Cause (1/3)



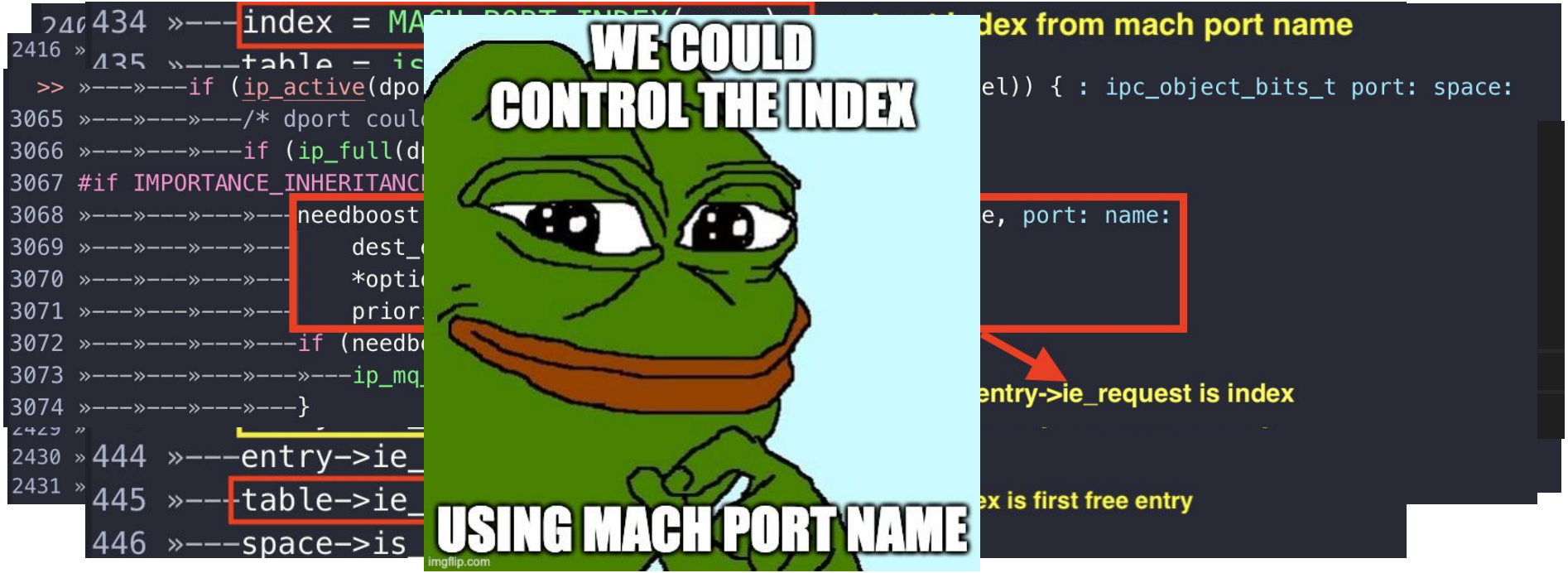
Root Cause (2/3)

```
111 struct ipc_entry {
112 »---union {
113 »---»---struct ipc_object *XN
114 »---»---struct ipc_object *XN
115 »---};
116 »---union {
117 »---»---ipc_entry_bits_t  ie_
118 »---»---ipc_entry_num_t   ie_
119 »---};
120 »---uint32_t              ie_di
121 »---mach_port_index_t     ie_in
122 »---union {
123 »---»---mach_port_index_t ie_
124 »---»---ipc_table_index_t ie_
125 »---};
126 };
```


```
ie_object;
volatile ie_volatile_object;
```

is union field with ie_next

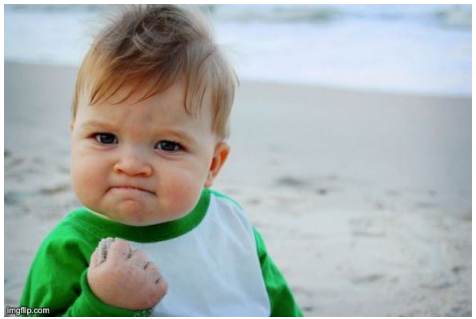
Root Cause (3/3)



Mach Port Spraying

Key takeaways

1. Mach port is very complex and it use a lot of “**union**” **in their structures** which could lead to some **side effects**.
2. With ability to generate test-case from custom template, my fuzzer able to explore some **potential code paths** which **haven’t reach before**.



Q&A

References

- Mach Overview
- CVE-2020-27932.
- https://www.fortinet.com/blog/threat-research/inspecting-mach-messages-in-macos-kernel-mode--part-i--sniffing-
- https://medium.com/@ali.pourhadi/ipc-mach-message-cab64ff1b569
- MOSEC 2021: Exploitation of XNU Port Type Confusions
- https://googleprojectzero.blogspot.com/2022/04/cve-2021-1782-ios-in-wild-vulnerability.html
- Understanding Mach IPC from MOSEC 2022
- CVE-2021-1870 Analysis
- https://github.com/kholia/OSX-KVM.

THANK YOU !