Drill the Apple Core: Up & Down
Fuzz Apple Core Component in Kernel and User Mode for Fun and Profit
Juwei Lin

- @panicall
- Joined TrendMicro Since 2013
- Windows Kernel/Rootkit/Bootkit
- Ransomware Decryption
- iOS/Android/Mac Vulnerability Hunting
Lilang Wu

- @Lilang_Wu
- Joined Trend Micro Since 2016
- Mac/iOS Vulnerability/Malware
- iOS/Android Exploit Detection
Moony Li

- @Flyic
- 8 years security
- Sandcastle
- Deep Discovery
- Exploit Detection
- Mac/Windows Kernel
- iOS/Android Vulnerability
Agenda

• Smart Fuzz XPC
  • XPC Internals
  • Fuzz Strategy
  • Reproduce Strategy
  • Output
• Smart Fuzz XNU
  • Introduction
  • Architecture and Sanitizer Support
  • Syntax Engine and Corpus
  • Sanitizers
  • Root Case Study
Smart Fuzzing XPC
• What is XPC?
  • low-level (libSystem) interprocess communication mechanism
  • simple messages and complex messages

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgh_bits</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_size</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_remote_port</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_local_port</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_voucher_port</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_id</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>msgh_descriptor_count</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>descriptor #1</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>descriptor #2</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>...</td>
<td>c</td>
<td>Simple messages</td>
</tr>
<tr>
<td>Trailer</td>
<td>c</td>
<td>Simple messages</td>
</tr>
</tbody>
</table>

Trailer

```c
descriptor_count
```

```c
24 bytes
```

```c
mach_msg_type_descriptor_t
```

```c
ool_descriptor #1
ool_descriptor #2
... Trailer
```

```c
ool_descriptor #1
ool_descriptor #2
... Trailer
```

```c
24 bytes
```
• Message Binary Format
• XPC Services
  • `launchctl dumpstate`
• Attack Surface
  • serialize/deserialize
  • libxpc
  • services code
• How to trigger these bugs?
• Proactive fuzz

1) body count
2) message descriptor
3) dictionary data
• Fuzz Strategy
  • Easy to control
  • Easy to mutate
  • Easy to monitor
  • Easy to reproduce
XPC Fuzz Architecture

[1] Mutation with seed
- Mutation Engine
  - Seed reproducer
  - Randomizer with Seed
- Fuzzer agent
- Mach Message Executor
- Crash Monitor
- Reproduce

[2] Fuzz target with seed
- Foundation.framework / NSXPC*
- Libxpc.dylib / xpc_*


[4] Reproduce by seed

Services
- Audio
- CoreMedia
- Windowserver
- Iohideventsystem
- ...

Seed server
• Fuzz Controller

  ✓ Wrap the xpc interfaces by python

```cpp
def XpcCreateConnection(self, init):
def mach_connect(self):
def XpcHandler(self):
def mach_msg(self):
def XpcSendMessage(self):

class XpcConnection:

BOOST_PYTHON_MODULE(xpcconnection) {
    PyEval_InitThreads();
}
```

✓ Python fuzz Engine
• Mutation
  • Pseudo-Random Number Generator with Mersenne Twister Algorithm
• **Crash Monitor**
  - Monitor the processes IDs cluster status
  - Monitor exits signal value

```
zuffdemac-pro:~ zuff$ launchctl list
PID  Status   Label
-    0        com.apple.SafariHistoryServiceAgent
 307   0        com.apple.Finder
 336   0        com.apple.homed
 578   0        com.apple.SafeEjectGPUAgent
-    0        com.apple.quicklook
-    0        com.apple.parentalcontrols.check
-    0        com.apple.PackageKit.InstallStatus
 345   0        com.apple.mediaremoteagent
-    0        com.apple.FontWorker
 321   0        com.apple.bird
-    0        com.apple.familycontrols.useragent
-    0        com.apple.AssetCache.agent
 666   0        com.apple.universalaccessAuthWarn
 312   0        com.apple.nsurlsessiond
-    0        com.apple.mobileactivationd
-    0        com.apple.syncservices.uichandler
 352   0        com.apple.iconservices.iconservicesagent
```
**Comparison between different Reproduce Methods**

<table>
<thead>
<tr>
<th></th>
<th>Typical Example</th>
<th>Storage Cost</th>
<th>Speed Cost</th>
<th>Support Complex Scenario</th>
<th>Reproduce Rate</th>
<th>Dev Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>Trinity</td>
<td>High (Execution Log)</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Case(File)</td>
<td>AFL</td>
<td>Middle (Files Causing Crash)</td>
<td>Low</td>
<td>Middle</td>
<td>Middle</td>
<td>High</td>
</tr>
<tr>
<td>Crash Dump</td>
<td>-</td>
<td>High (Every Crash Context)</td>
<td>High</td>
<td>-</td>
<td>Very Low</td>
<td>No</td>
</tr>
<tr>
<td>Seed</td>
<td>JS Fun Fuzz</td>
<td>Low (Integer)</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Case Study - CVE-2018-4411

Target 0: (fontd) stopped.

(lldb) bt

* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x7ffee1934000)
* frame #0: 0x00007fff55a06f49 libsystem_platform.dylib`_platform_memmove$VARIANT$Haswell + 41
  frame #1: 0x00007fff2b8b597a libATSServer.dylib`FODBWriteToAnnex + 246
  frame #2: 0x00007fff2b8d0157 libATSServer.dylib`HandleFontManagementMessage + 5403
  frame #3: 0x00007fff2b8cd2d1 libATSServer.dylib`serverMainHandler(__CFMachPort*, FontMgrMessage*, long, void*) + 263
  frame #4: 0x00007fff2d3e4596 CoreFoundation`__CFMachPortPerform + 310
  frame #5: 0x00007fff2d3e4449 CoreFoundation`__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__ + 41
  frame #6: 0x00007fff2d3e4395 CoreFoundation`__CFRunLoopDoSource1 + 533
  frame #7: 0x00007fff2d3dbf50 CoreFoundation`__CFRunLoopRun + 2848
  frame #8: 0x00007fff2d3dbf50 CoreFoundation`CFRunLoopRunSpecific + 483
  frame #9: 0x00007fff2d419c33 CoreFoundation`CFRunLoopRun + 99
  frame #10: 0x00007fff2b8cc91c libATSServer.dylib`main_handler + 4510
  frame #11: 0x00007fff556f5015 libdyld.dylib`start + 1
  frame #12: 0x00007fff556f5015 libdyld.dylib`start + 1

• Case Study - CVE-2018-4411
Smart Fuzzing XNU
Smart Fuzzing XNU

• Introduction of Smart Fuzzing XNU
• Architecture and Sanitizer Support
• Syntax Engine and Corpus
• Sanitizers
• Root Case Study
What I will introduce today

1. Port Syzkaller to Support macOS XNU Fuzzing.

2. Modify XNU to add support some features.
**Fuzzer**
- 530 BSD API Patterns
- VM Fusion Support
- macOS Executor

**XNU**
- Add Code Coverage
- Add Kernel Memory Sanitizer
- Enable Kernel Address Sanitizer
1. Key modules are in GREEN

2. Also add some other modules, e.g. vmfusion
My Efforts

• Syntax Engine is directly from Syzkaller; But I developed the XNU BSD API patterns.
• Kasan is from XNU, but it does not work well after compilation.
• I developed coverage sanitizer.
• I developed kmsan.
Syntax Engine & Corpus
Quick glance at syzkaller’s syntax engine

syz-extract & syz-sysgen
  • description in txt written by human

 generation engine
  • description in go made by previous 2 tools

 executor
  • program made by previous generation engine
Corpus

• More than 500 syscalls in XNU kernel

• Refer to syzkaller’s syscall descriptions syntax: https://github.com/google/syzkaller/blob/master/docs/syscall_descriptions_syntax.md

• Refer to sample txt files in syzkaller project
## Basic Concepts 1: User Mode Sanitizers

<table>
<thead>
<tr>
<th>Name</th>
<th>Features</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressSanitizer²</td>
<td>• Out-of-bounds accesses to heap, stack and globals</td>
<td>• compiler instrumentation module</td>
</tr>
<tr>
<td></td>
<td>• Use-after-free</td>
<td>• run-time library</td>
</tr>
<tr>
<td></td>
<td>• Use-after-return</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Use-after-scope</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Double-free, invalid free</td>
<td></td>
</tr>
<tr>
<td>MemorySanitizer³</td>
<td>• uninitialized reads</td>
<td></td>
</tr>
<tr>
<td>SanitizerCoverage⁴</td>
<td>• get function/block/edge coverage</td>
<td>• Instrumentations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Default callbacks provided</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ThreadSanitizer⁵</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UndefinedBehaviorSanitizer⁶</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DataflowSanitizer⁷</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LeakSanitizer⁸</td>
</tr>
</tbody>
</table>
# Basic Concepts 2: Kernel Mode Sanitizers

<table>
<thead>
<tr>
<th>Name</th>
<th>Features</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Kernel Sanitizer Coverage                | • get function/block/edge coverage                                       | • Has instrumentations support
|                                           |                                                                          | • NO existing callbacks implementation       |
| KASAN (kernel address sanitizer)          | • Out-of-bounds accesses Use-after-free                                 | • Has instrumentations support
|                                           | • Use-after-return                                                       | • Has callbacks/module support                |
|                                           | • Use-after-scope                                                        |                                               |
|                                           | • Double-free, invalid free                                              |                                               |
| KMSAN (kernel memory sanitizer)           | • uninitialized reads                                                    | • Not implemented                             |
Sanitizer Coverage

• We need to develop a new module in XNU to:
  • Support sanitizer callback function
  • Read the coverage data back to user fuzzing program
Callback Implementation

1. callback name: __sanitizer_cov_trace_pc
2. just support single-thread mode
3. store coverage structure into task_t

```c
struct task {
    ...
    enum kcov_mode kcov_mode;
    unsigned kcov_size;
    void *kcov_area;
    struct kcov *kcov;
    uint32_t refcount;
}

void __attribute__((noinline)) __sanitizer_cov_trace_pc()
{
    ...
}
```
After Compilation
**KASAN**

- latest XNU has KASAN support
  - KDK now provides kernel.kasan which works well.
  - It does not work if you compile it, VM cannot boot.

- It consists of *guard pages*, *shadow memory* and *operations*.

- It can protect Globals, Stack and Heap memory.
How KASAN protects memory

1) memory operations are called, e.g. __asan_strlcpy

2) __asan_strlcpy checks shadow memory

3) KASAN panics the kernel if shadow memory is illegal (shadow value < 0)
Guard Pages & Shadow Memory

Memory

Left Red Zone  Using Memory  Right Red Zone

Shadow Memory
(1/8 size of memory)
## Operations

<table>
<thead>
<tr>
<th>Heap Memory Operations</th>
<th>Stack Memory Operations</th>
<th>Other Memory Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>__asan_bcopy</td>
<td>__asan_stack_malloc_0</td>
<td>__asan_load1</td>
</tr>
<tr>
<td>__asan_memmove</td>
<td>__asan_stack_malloc_1</td>
<td>__asan_load2</td>
</tr>
<tr>
<td>__asan_memcpy</td>
<td>__asan_stack_malloc_2</td>
<td>__asan_load4</td>
</tr>
<tr>
<td>__asan_memset</td>
<td>__asan_stack_malloc_3</td>
<td>__asan_load8</td>
</tr>
<tr>
<td>__asan_bzero</td>
<td>__asan_stack_malloc_4</td>
<td>__asan_load16</td>
</tr>
<tr>
<td>__asan_memcpy</td>
<td>__asan_stack_malloc_5</td>
<td>__asan_loadN</td>
</tr>
<tr>
<td>__asan_memcmp</td>
<td>__asan_stack_malloc_6</td>
<td></td>
</tr>
<tr>
<td>__asan_strlcpy</td>
<td>__asan_stack_malloc_7</td>
<td></td>
</tr>
<tr>
<td>__asan_strlcat</td>
<td>__asan_stack_malloc_8</td>
<td></td>
</tr>
<tr>
<td>__asan_strncpy</td>
<td>__asan_stack_malloc_9</td>
<td></td>
</tr>
<tr>
<td>__asan_strncat</td>
<td>__asan_stack_malloc_10</td>
<td></td>
</tr>
<tr>
<td>__asan_strnlen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__asan_strlen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#define strlcpy __asan_strlcpy

-fsanitizer=address

buildin calls in xnu source code
Example: Detect UAF

- When new memory is allocated and aligned with 8
Example cont. 1

• When the memory is freed

![Diagram showing memory and shadow memory with values in hexadecimal]

- Memory
- 8 bytes
- 8 bytes
- 7 bytes
- Shadow Memory
- ff ff ff
Example cont. 2

• When the memory is used after free, any related operation will check its shadow memory and then panic the system.
  • 0xff is illegal
KMSAN

• Kernel memory sanitizer is used to detect uninitialized memory.

• We worked on how to initialize all uninitialized memory allocated in kernel, e.g. kalloc_canblock
kalloc_canblock

```c
assert(size <= z->elem_size);

#if VM_MAX_TAG_ZONES
    if (z->tags && site)
    {
        tag = vm_tag_alloc(site);
        if (!canblock && !vm_allocation_zone_totals[tag])
            tag = VM_KERN_MEMORY_KALLOC;
    }
#endif

addr = zalloc_canblock_tag(z, canblock, size, tag);

#if KASAN_KALLOC
    /* fixup the return address to skip the redzone */
    addr = (void *)kasan_alloc((vm_offset_t)addr, z->elem_size, req_size, KASAN_GUARD_SIZE);
    /* For KASan, the redzone lives in any additional space, so don't
        * expand the allocation. */
#else
    psize = z->elem_size;
#endif

    // add by @panicall
    if (addr)
        memset(addr, 0xde, *psize);
    return addr;
```
Check in fuzzer

1. check continuous `0xDE`
2. add a syscall `panicall_report`
Conclusion

- About 530 API patterns
- Corpus

- Coverage Sanitizer
- KASAN
- KMSAN

Syntax Engine + Sanitizers = Smart XNU Fuzzer
macOS Root Case Study
CVE-2018-4413
• Uninitialized heap memory leak
• Fixed in macOS 10.14.1 and iOS 12.1
• Can be used to leak ipc_port object address

CVE-2018-4425
• NECP type confusion
• Fixed in macOS 10.14
• Can be used to write arbitrary kernel address
• Can be used to free arbitrary kernel address
CVE-2018-4413

sysctl_procargsx is used to retrieve process args information by calling sysctl.

at location (a):
• p->p_argslen is usually around 0x300;
• I set my arg_size to 0x200 so that arg_size will not be round_paged.
At location (b):
- Stack information is copied to new allocated page at offset 0 with arg_size (0x200).
- The new allocated page is not zeroed. So this operation leaves the rest of this page filled with uninitialized heap data.

At location (c):
- copy_end is round_paged, parameter data points to the last 0x200 bytes of the page.

At location (d):
- copyout the 0x200 bytes leaked heap information to user buffer.
### Args Information
- **Arg Size**: 0x200

### Data
- **Uninitialized Heap Data 1**
- **Uninitialized Heap Data 2** (arg_size, 0x200)

**Leaked!!!**
Exploit CVE-2018-4413 to leak ipc_port object address:

MACH_MSG_OOL_PORTS_DESCRIPTOR

Destroy the ports memory:

mach_port_destroy(mach_task_self(), q);

Trigger the vulnerability to leak the ports memory:
CVE-2018-4413

Apple fixed it by calling bzero.
CVE-2018-4425
NECP Attack Surface 1

necp_open → necp_fd_data → necp_client_action →
necp_client_add
necp_client_remove
necp_client_copy
necp_client_list
necp_client_agent_action
necp_client_copy_agent
necp_client_agent_use
necp_client_copy_interface
necp_client_copy_route_statistics
necp_client_update_cache
necp_client_copy_client_update
CVE-2018-4425
NECP Attack Surface 1

necp_open assigns necp_fd_data to fg_data:
- user-mode syscall gets returned fd handle
- fd is an index to kernel fp object
- fp object contains necp_fd_data object as fg_data
necp_client_action operates on fg_data:

- at (a), call necp_find_fd_data to find necp_fd_data with given handle
- dispatch methods operates on necp_fd_data

```
int necp_client_action(struct proc *p, struct necp_client_action_args *uap, int *retval)
{
    #pragma unused(p)
    int error = 0;
    int return_value = 0;
    struct necp_fd_data *fd_data = NULL;
    error = necp_find_fd_data(uap->necp_fd, &fd_data);  ---(a)
    if (error != 0) {
        NECPLOG(LOG_ERR, "necp_client_action find fd error (%d)", error);
        return (error);
    }

    u_int32_t action = uap->action;
    switch (action) {
        ...
    }
}
```

CVE-2018-4425
NECP Attack Surface 1
necp_find_fd_data finds fd_data:

• call fp_lookup to get fp of given fd
• at (b), verify if the fp is of type necp_fd_data by checking fo_type

```
static int
necp_find_fd_data(int fd, struct necp_fd_data **fd_data)
{
    proc_t p = current_proc();
    struct fileproc *fp = NULL;
    int error = 0;

    proc_fdlock_spin(p);
    if ((error = fp_lookup(p, fd, &fp, 1)) != 0) {
        goto done;
    }
    if (fp->f_fglob->fg_ops->fo_type != DTYPE_NETPOLICY) {  ---(b)
        fp_drop(p, fd, fp, 1);
        error = ENODEV;
        goto done;
    }
    *fd_data = (struct necp_fd_data *)fp->f_fglob->fg_data;

done:
    proc_fdunlock(p);
    return (error);
}
```
Normal Process:

- necp_open creates necp_fd_data object in kernel and returns handle to user mode
- necp_client_action finds the necp_fd_data by given handle, it internally checks if corresponding fo_type equals DTYPE_NETPOLICY
- dispatch methods of necp_client_action operates on found necp_fd_data
CVE-2018-4425
NECP Attack Surface 2

`necp_session_open` → `necp_session` → `necp_session_action`

- `necp_session_add_policy`
- `necp_session_get_policy`
- `necp_session_delete_policy`
- `necp_session_apply_all`
- `necp_session_list_all`
- `necp_session_delete_all`
- `necp_session_set_session_priority`
- `necp_session_lock_to_process`
- `necp_session_register_service`
- `necp_session_unregister_service`
- `necp_session_dump_all`
necp_session_open assigns necp_session to fg_data:
- user-mode syscall gets returned fd handle
- fd is an index to kernel fp object
- fp object contains necp_session object as fg_data
necp_session_action operates on fg_data:
- at (aa), call necp_session_find_from_fd to find necp_session with given handle
- dispatch methods operates on necp_session object
necp_session_find_from_fd finds fd_data:
  • call fp_lookup to get fp of given fd
  • at (bb), verify if the fp is of type necp_session by checking fo_type

```c
static int
necp_session_find_from_fd(int fd, struct necp_session **session)
{
    proc_t p = current_proc();
    struct fileproc *fp = NULL;
    int error = 0;

    proc_fdlock_spin(p);
    if ((error = fp_lookup(p, fd, &fp, 1)) != 0) {
        goto done;
    }
    if (fp->f_fglob->fg_ops->fo_type != DTYPE_NETPOLICY) {  ---(bb)
        fp_drop(p, fd, fp, 1);
        error = ENODEV;
        goto done;
    }
    *session = (struct necp_session *)fp->f_fglob->fg_data;

    done:
    proc_fdunlock(p);
    return (error);
}
```
Normal Process:

- `necp_session_open` creates `necp_session` object in kernel and returns handle to user mode
- `necp_session_action` finds the `necp_session` by given handle, it internally checks if corresponding `fo_type` equals `DTYPE_NETPOLICY`
- `dispatch` methods of `necp_session_action` operates on found `necp_session`
CVE-2018-4425
Type Confusion

What we learn so far:

Attack surface 1: if fp->...->fo_type == DTYPE_NETPOLICY , fp is of type `necp_fd_data`

Attack surface 2: if fp->...->fo_type == DTYPE_NETPOLICY , fp is of type `necp_session`

`necp_fd_data` is totally different from `necp_session`!!!
CVE-2018-4425
Exploit: arbitrary address free

Method:
1. create necp_fd_data object and call necp_session_action to operate on it
2. create necp_session object and call necp_client_action to operate on it
CVE-2018-4425

Exploit: arbitrary address free

Step 1: call ncep_open to create ncep_fd_data object:
- fd_data->update_list is initialized by TAILQ_INIT
  +20: 0
  +28: update_list address

```c
struct ncep_fd_data {
    +0x00 u_int8_t ncep_fd_type;
    +0x08 LIST_ENTRY(ncep_fd_data) chain;
    +0x18 struct _ncep_client_tree_clients;
    +0x20 TAILQ_HEAD(_ncep_client_update_list, ncep_client_update) update_list;
    +0x30 int update_count;
    +0x34 int flags;
    +0x38 int proc_pid;
    +0x40 decl_lck_mtx_data,( fd_lock);
    +0x50 struct selinfo si;
};
```
CVE-2018-4425
Exploit: arbitrary address free

necp_open →
+0x20: 0
+0x28: update_list address
CVE-2018-4425
Exploit: arbitrary address free

Step 2 call `necp_session_action` on the object
at location (b), if `session->proc_locked` is false(0),
`session->proc_uuid` and `session->proc_pid` will be
updated.

```c
int
necp_session_action(struct proc *p, struct necp_session_action_args *uap, int *retval)
{
    #pragma unused(p)
    int error = 0;
    int return_value = 0;
    struct necp_session *session = NULL;
    error = necp_session_find_from_fd(uap->necp_fd, &session);
    if (error != 0) {
        NECPLOG(LOG_ERR, "necp_session_action find fd error (\%d)\", error);
        return (error);
    }
    NECP_SESSION_LOCK(session);
    if (session->proc_locked) {
        // Verify that the calling process is allowed to do actions
        uid_t proc_uuid;
        proc_getexecutableuuid(current_proc(), proc_uuid, sizeof(proc_uuid));
        if (uid_compare(proc_uuid, session->proc_uuid) != 0) {
            error = EPERM;
            goto done;
        }
    } else {
        // If not locked, update the proc_uuid and proc_pid of the session
        proc_getexecutableuuid(current_proc(), session->proc_uuid, sizeof(session->proc_uuid));
        session->proc_pid = proc_pid(current_proc());  ---(b)
    }
    ...
}
```
CVE-2018-4425
Exploit: arbitrary address free

- `session->proc_locked` at offset 0x20 overlaps `update_list` which is 0 in `necp_fd_data`.
- `session->proc_uuid` at offset 0x21 is updated with macho UUID
- `session->proc_pid` is updated with current pid
CVE-2018-4425
Exploit: arbitrary address free

+0x20: 0
+0x28: update_list address

necp_session_action

+0x20: 0
+0x21: UUID, low 7Bytes
+0x28: UUID, high 9Bytes
+0x34: pid
CVE-2018-4425
Exploit: arbitrary address free

Step 3 call nectp_client_action on the object
• we use action 15(nectp_client_copy$client_update) on the object
• at location (f), client_update is freed
• client_update is the first element of update_list which is UUID now
CVE-2018-4425
Exploit: arbitrary address free

For Example, we set MachO UUID (16 bytes) as 41414141414141414141414141414141, here we get 0x4141414141414100 freed. We can control high 7 bytes of the address to be freed.
CVE-2018-4425

Apple Fix

Add sub type check:
necep_session has sub type 1
necep_fd_data has sub type 2
Future Plan of Our Fuzzing Tool

• Support kernel extension
• Support IOKit(+code coverage)
• Support Passive Fuzzing
• More and More Corpus
IOKit Code Coverage Example
macOS <= 10.14 Root

- mach_portal: all details [https://bugs.chromium.org/p/project-zero/issues/detail?id=1417](https://bugs.chromium.org/p/project-zero/issues/detail?id=1417)
- Demo(10.13.6)
More Information

• follow me on twitter: @panicall
Acknowledge

• Google Project Syzkaller\textsuperscript{1}
ANY QUESTIONS?