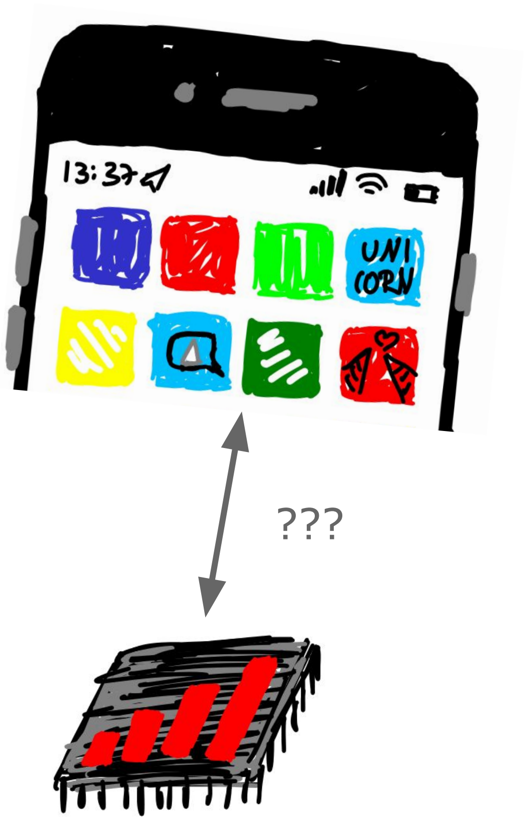
Interacting with *OS Hardware from User Space



Jiska Classen
Secure Mobile Networking Lab - SEEMOO
Technical University of Darmstadt, Germany

As a…

hardware hacker

I want to…

buy iPhones

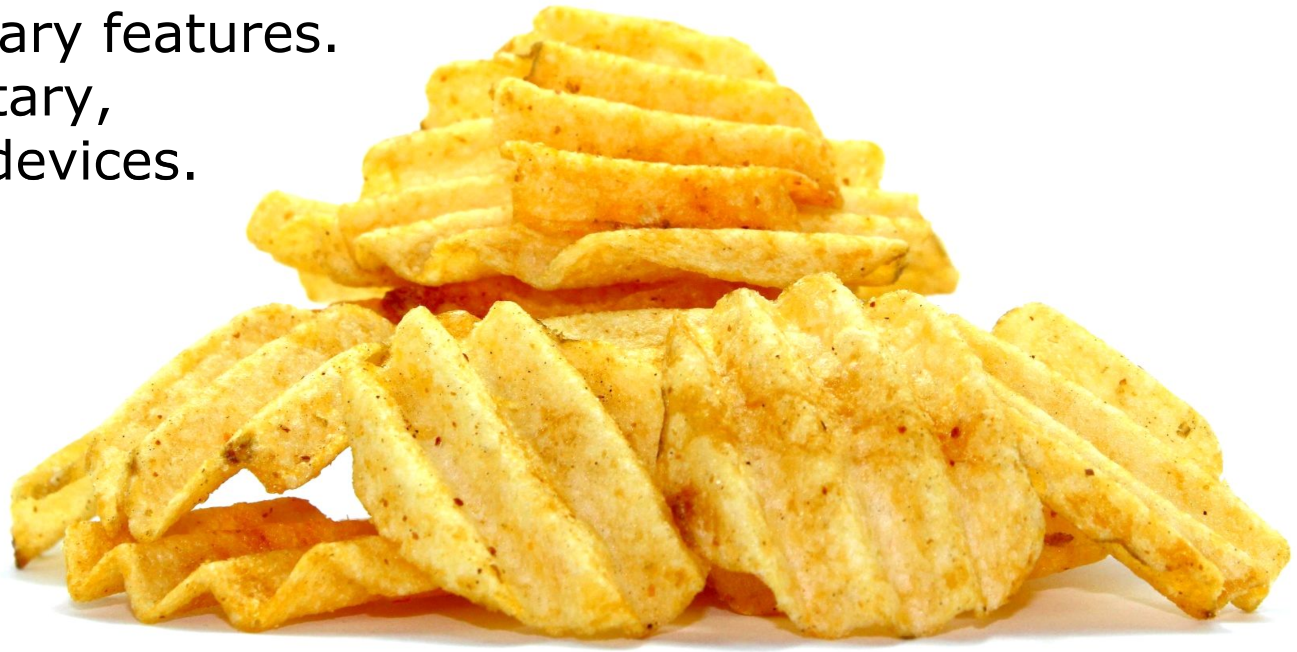
So that…

I can break the most recent chips

Wait, what?

- Official development kits often lag behind.
  - Bluetooth/Wi-Fi kits by Cypress (aka Broadcom) are stuck on a firmware state around 2016.
  - Ultra Wideband kits by Decawave added features like direction measurement much later than it was available in iPhones.

- Official development kits miss proprietary features.
  - Intel baseband chips use a proprietary, undocumented interface on Apple devices.



Bluetooth Chip Build Dates
iPhone 11: Oct 25 2018
iPhone 12: Oct 29 2019
Samsung Galaxy S21: April 13 2018
(S21+ Ultra probably got an update)

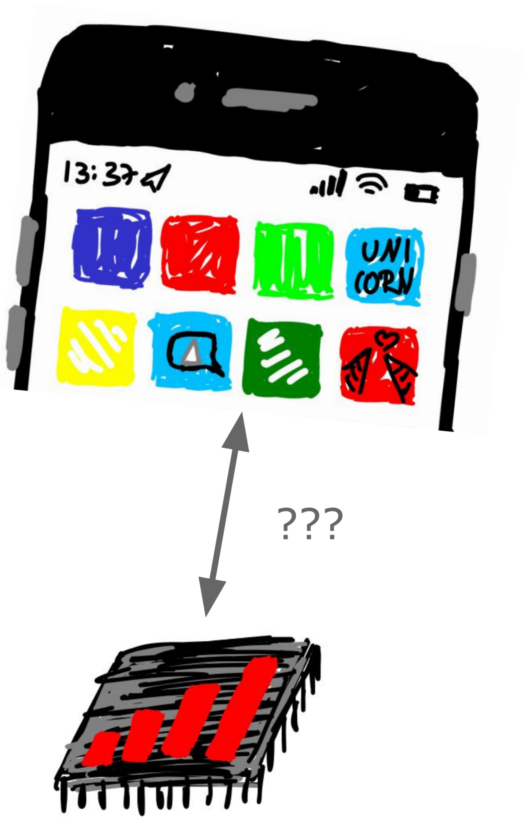
Hardware in an iPhone

- Bluetooth+Wi-Fi combo chip by Broadcom
- Baseband chip by Intel or Qualcomm
- U1 chip (in-house, since iPhone 11)
- NFC chip
- … and that’s just the wireless chips!

If I deal with iPhones, jailbreaks, etc.
I can access quite a lot of hardware 🎉

Goals

1. Find chip interfaces.
2. Find protocol handlers.
3. Decode proprietary protocols.
4. Inject custom payloads.

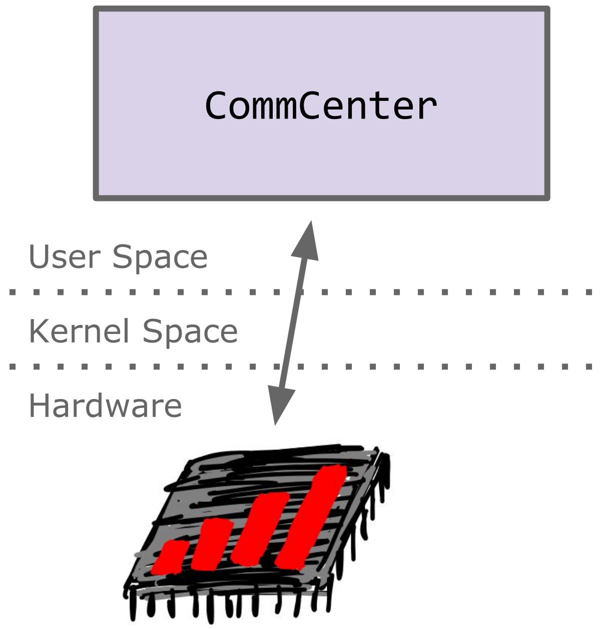


Why from user space?!

- The daemon that interacts with the chip already holds a correct state.
  - Chip initialization/activation on startup and when leaving flight mode.
  - Complex protocol internals are implemented here.

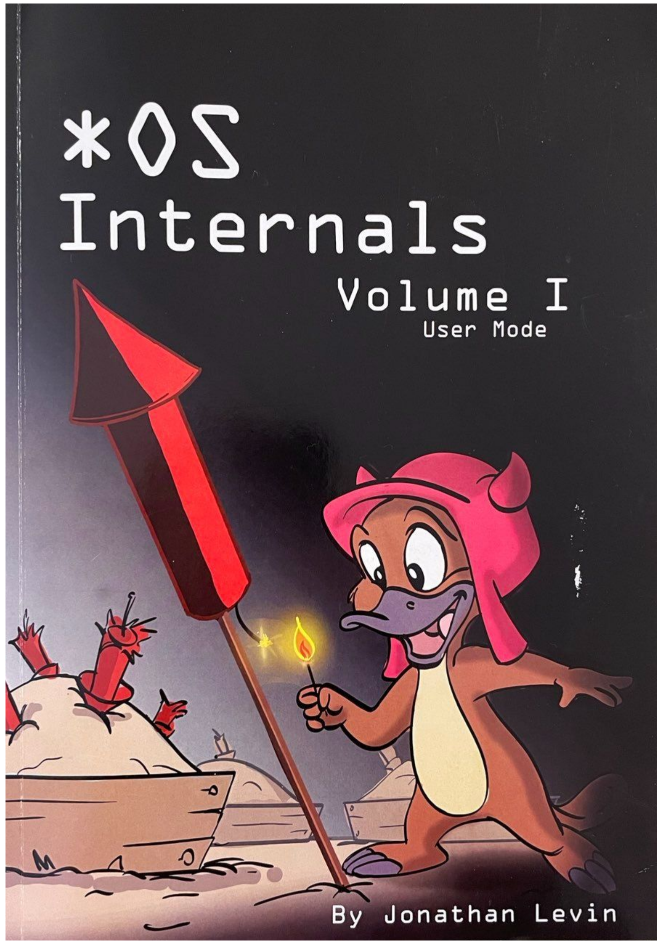
- The daemon's protocol implementation will typically:
  - parse crash logs,
  - acknowledge packets,
  - forward information to other daemons,
  - …

- FЯIDA only supports user space.

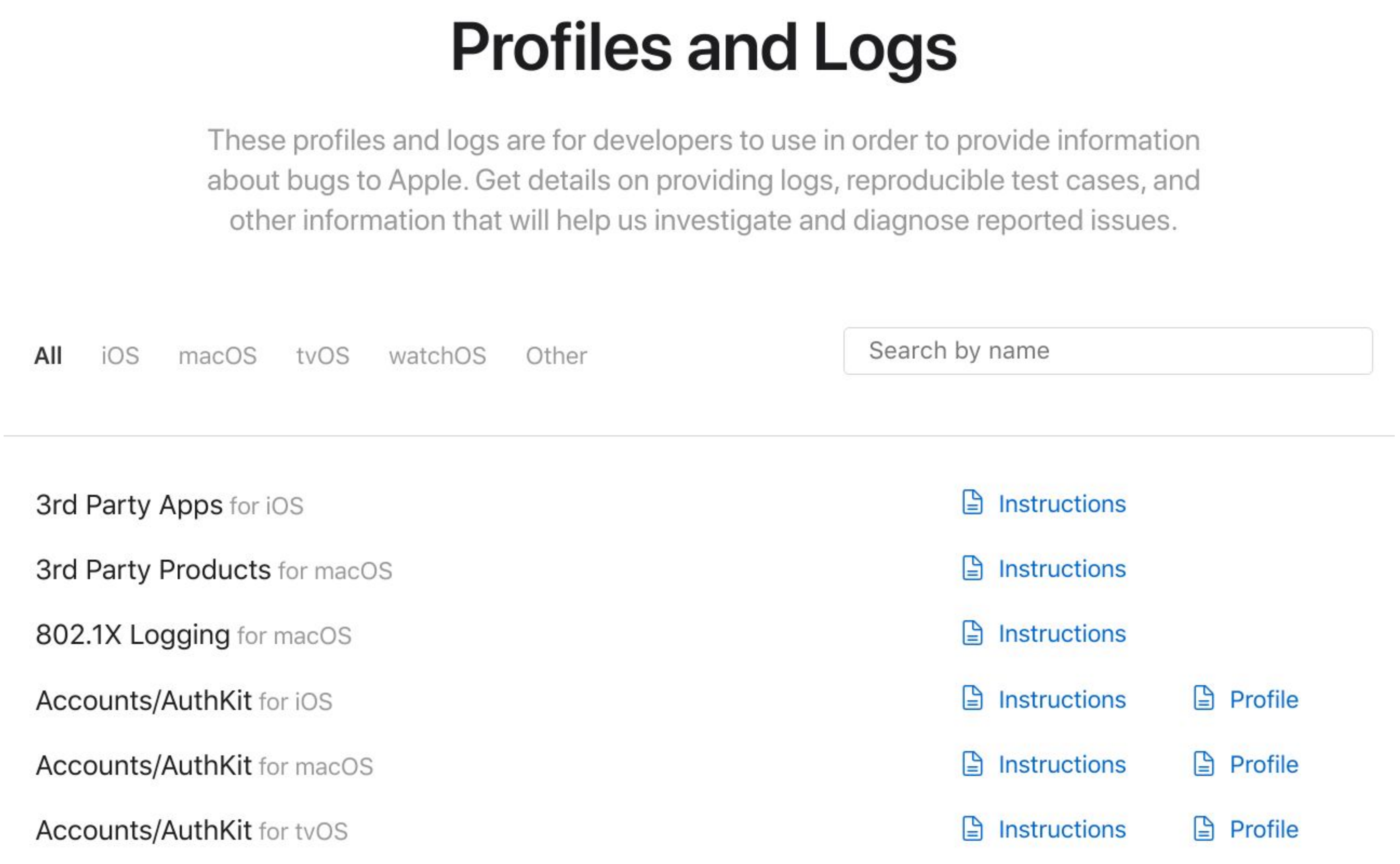


But you said *OS??!

- XNU kernel is very similar on MacOS, iOS, and the iOS derivatives like audioOS, watchOS, tvOS, …
- User space is also mostly similar.
- Everything in this talk should work on iOS and MacOS.
- Slightly inspired by the *"*OS Internals"* book series, definitely a good read if you want to dig deeper!



iOS Debugging Basics



Profiles and Logs

These profiles and logs are for developers to use in order to provide information about bugs to Apple. Get details on providing logs, reproducible test cases, and other information that will help us investigate and diagnose reported issues.

3rd Party Apps for iOS	Instructions	
3rd Party Products for macOS	Instructions	
802.1X Logging for macOS	Instructions	
Accounts/AuthKit for iOS	Instructions	Profile
Accounts/AuthKit for macOS	Instructions	Profile
Accounts/AuthKit for tvOS	Instructions	Profile

https://developer.apple.com/bug-reporting/profiles-and-logs/

When there is no profile...

```
iPhone# vim /Library/Preferences/Logging/com.apple.system.logging.plist
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST
1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Enable-Private-Data</key>
        <true/>
</dict>
</plist>
```

```
iPhone# killall -9 logd
```

https://github.com/EthanArbuckle/unredact-private-os_logs

Demo: `idevicesyslog`

```
idevicesyslog -p CommCenter
```

```
CommCenter(libARI.dylib)[4111] <Notice>: ari: (forwardIndication:123) Indication(0x25820000)
    for client(IndicationReregistrationActor) Type(GCD) size(110) dispq(AriHostIPC:0x101d712d0)
CommCenter(libARI.dylib)[4111] <Notice>: ind: Bin=['DE C0 7E AB … 00 00']
CommCenter(libARI.dylib)[4111] <Notice>: ari: (forwardIndication:123) Indication(0x25820000)
    for client(IndicationReregistrationActor) Type(GCD) size(110) dispq(AriHostIPC:0x101d712d0)
CommCenter(libARI.dylib)[4111] <Notice>: ind: Bin=['DE C0 7E AB … 00']
CommCenter[4111] <Notice>: #I CCXpcServer(1808[mediaserverd]:'Virtual Audio'/0x103616680)
    request: kSetActiveAudioSystemConfiguration.
```

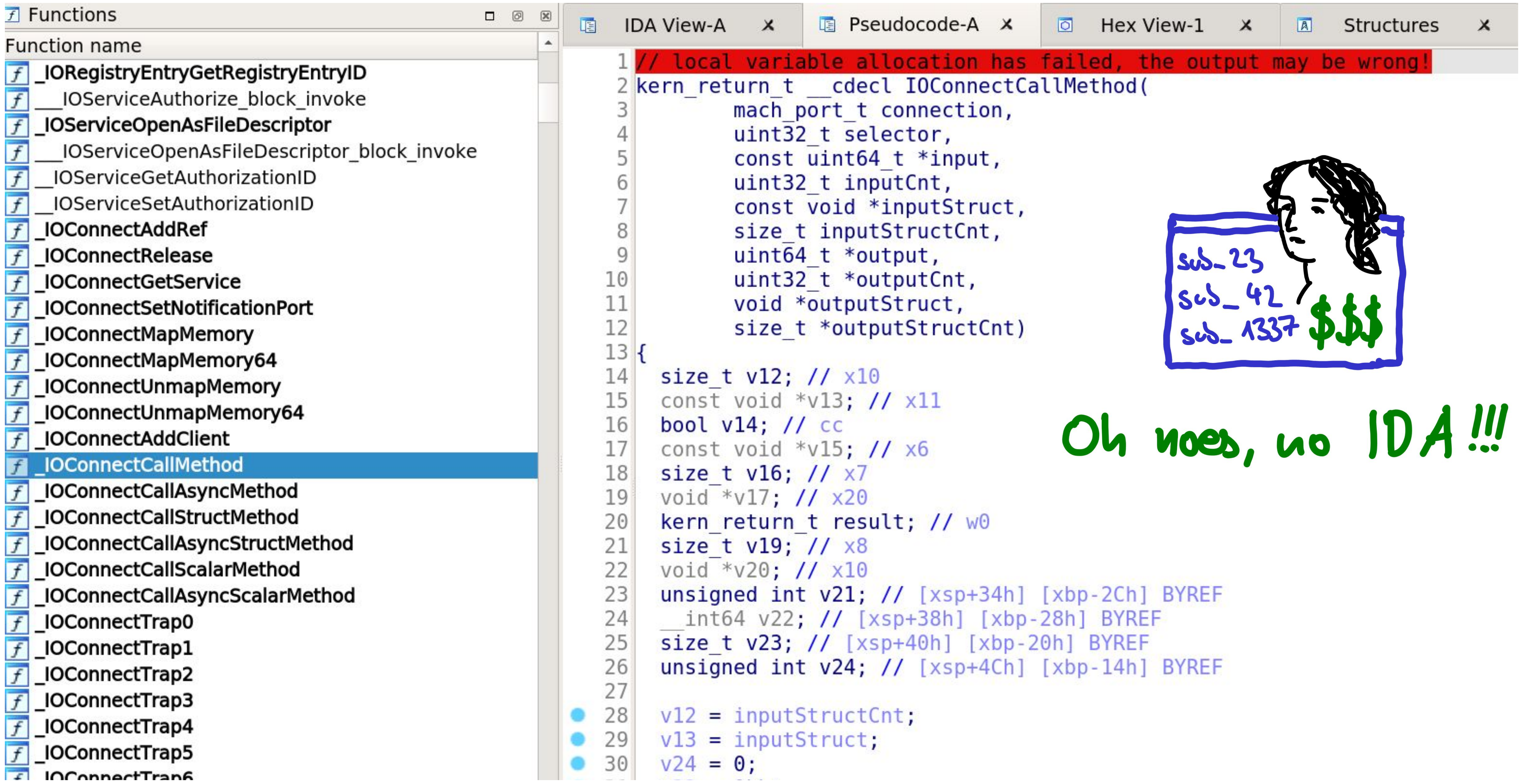
Shared Libraries

- Programs can use shared libraries.
- 3rd-party usage requires function name exports.
- DYLD shared cache is automatically extracted from an iPhone after connecting it to Xcode:
  `~/Library/Developer/Xcode/iOS DeviceSupport/`

- Lower levels of hardware access are often defined in shared libraries.
- `libARI.dylib` and `libARIServer.dylib` define almost every field of the Apple Remote Invocation (ARI) protocol for Intel baseband chips, Wireshark dissector to be released soon!

ARIstoteles

Dissector by Tobias Kröll



```
_IORegistryEntryGetRegistryEntryID
___IOServiceAuthorize_block_invoke
_IOServiceOpenAsFileDescriptor
___IOServiceOpenAsFileDescriptor_block_invoke
__IOServiceGetAuthorizationID
__IOServiceSetAuthorizationID
_IOConnectAddRef
_IOConnectRelease
_IOConnectGetService
_IOConnectSetNotificationPort
_IOConnectMapMemory
_IOConnectMapMemory64
_IOConnectUnmapMemory
_IOConnectUnmapMemory64
_IOConnectAddClient
_IOConnectCallMethod
_IOConnectCallAsyncMethod
_IOConnectCallStructMethod
_IOConnectCallAsyncStructMethod
_IOConnectCallScalarMethod
_IOConnectCallAsyncScalarMethod
_IOConnectTrap0
_IOConnectTrap1
_IOConnectTrap2
_IOConnectTrap3
_IOConnectTrap4
_IOConnectTrap5
_IOConnectTrap6
```

```
// local variable allocation has failed, the output may be wrong!
kern_return_t __cdecl IOConnectCallMethod(
        mach_port_t connection,
        uint32_t selector,
        const uint64_t *input,
        uint32_t inputCnt,
        const void *inputStruct,
        size_t inputStructCnt,
        uint64_t *output,
        uint32_t *outputCnt,
        void *outputStruct,
        size_t *outputStructCnt)
{
  size_t v12; // x10
  const void *v13; // x11
  bool v14; // cc
  const void *v15; // x6
  size_t v16; // x7
  void *v17; // x20
  kern_return_t result; // w0
  size_t v19; // x8
  void *v20; // x10
  unsigned int v21; // [xsp+34h] [xbp-2Ch] BYREF
  __int64 v22; // [xsp+38h] [xbp-28h] BYREF
  size_t v23; // [xsp+40h] [xbp-20h] BYREF
  unsigned int v24; // [xsp+4Ch] [xbp-14h] BYREF

  v12 = inputStructCnt;
  v13 = inputStruct;
  v24 = 0;
```

Oh noes, no IDA!!!

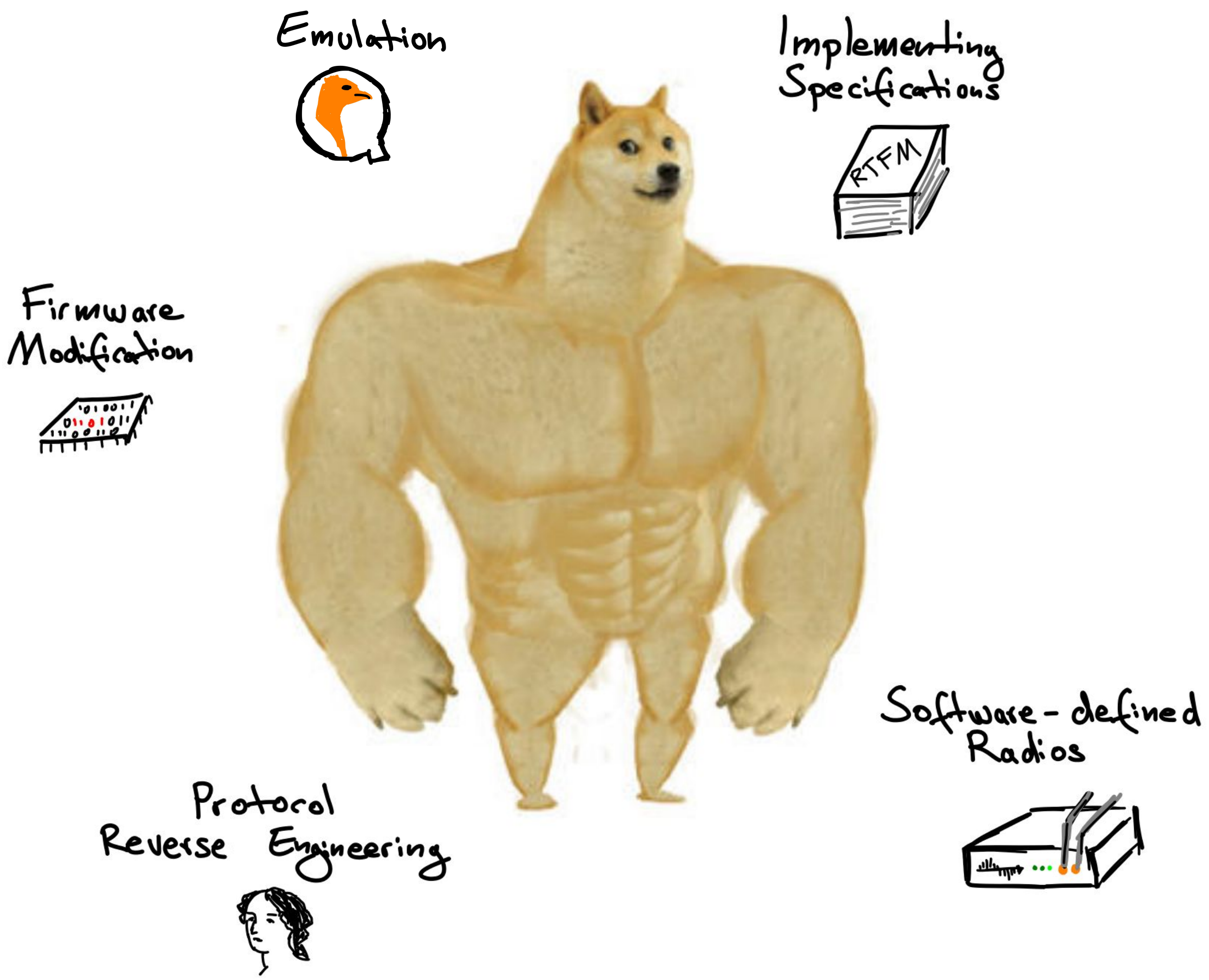
Symbols/System/Library/Frameworks/IOKit.framework/Versions/A/IOKit

iOS Debugserver

- Part of the developer tools.
- Copy it to `/usr/bin`, add more entitlements with `ldid`, attach to any process.
- Still not that intuitive to use, `lldb` is so-so, and even in combination with IDA Pro…

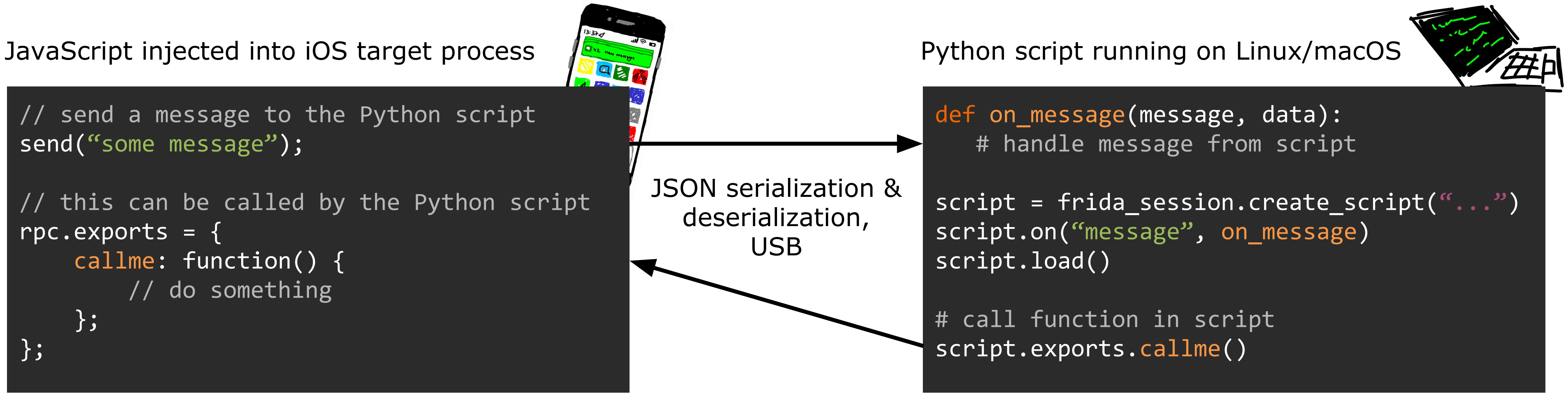
Not beginner-friendly, so we won’t use it here…

FRIDA Basics


Emulation
Implementing Specifications
RTFM
Firmware Modification
Software-defined Radios
Protocol Reverse Engineering

FRIDA

Running FRIDA on iOS



JavaScript injected into iOS target process

```
// send a message to the Python script
send(“some message”);

// this can be called by the Python script
rpc.exports = {
    callme: function() {
        // do something
    };
};
```

JSON serialization & deserialization, USB

Python script running on Linux/macOS

```
def on_message(message, data):
   # handle message from script

script = frida_session.create_script(“...”)
script.on(“message”, on_message)
script.load()

# call function in script
script.exports.callme()
```

frida-trace

```
mac# frida-trace -U CommCenter --decorate -i 'objc_msg*'
```

- The local `frida` and `frida-trace` command etc. run Python on the local machine and call something on the iPhone.

https://frida.re/docs/frida-trace/

Layer 8

```
mac# frida -U somedaemon --no-pause -l somescript.js
```

- You can open `somescript.js` in parallel and every time you save it, it will be executed again.
- The output console can also be used for input, so you can try single lines of code or call functions without reloading the `somescript.js`.
- You can even use the Chrome developer tools along with your scripts.

https://www.youtube.com/watch?v=PHa1F3tT130 Frida Scripting Simpsons by Tavis Ormandy

Analyzing Function Arguments

```
var _IOConnectCallMethod_addr = Module.getExportByName('IOKit', 'IOConnectCallMethod');

Interceptor.attach(_IOConnectCallMethod_addr, {
    onEnter: function(args) {
        var connection = args[0];
        // ...
    }
});
```

- The interceptor can change values `onEnter` and `onLeave` of a function.
- Function arguments are mapped to `args[0]`, `args[1]`, … (architecture-independent).
- Registers can also be accessed directly, like `this.context.x0` (architecture-dependent).

https://frida.re/docs/javascript-api/

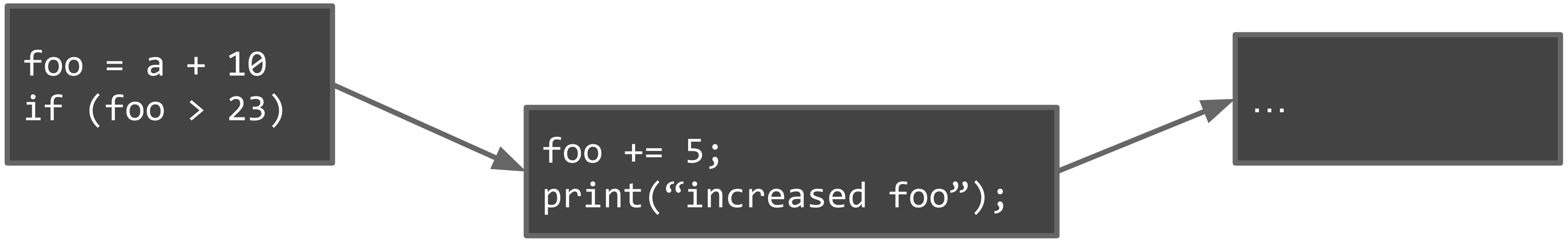
Backtrace

```
console.log('Called from:\n' +
            Thread.backtrace(this.context, Backtracer.ACCURATE)
            .map(DebugSymbol.fromAddress).join('\n') + '\n');
```

- Backtrace for the current thread.
- Even adds symbols if known via shared libraries.

https://frida.re/docs/javascript-api/

Stalker

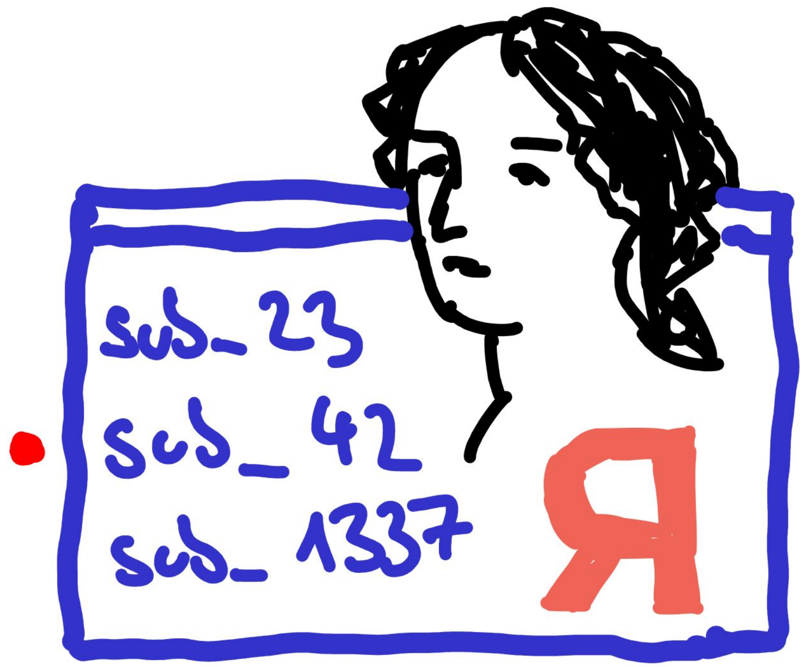

```
foo = a + 10
if (foo > 23)
```
```
foo += 5;
print("increased foo");
```
```
...
```

- Rewrite code during runtime to trace its execution.
- Supports different granularities, such as functions, or even basic blocks.
- Can be integrated for coverage-guided fuzzing.
- As of now, it only follows direct calls, e.g., no thread switches.

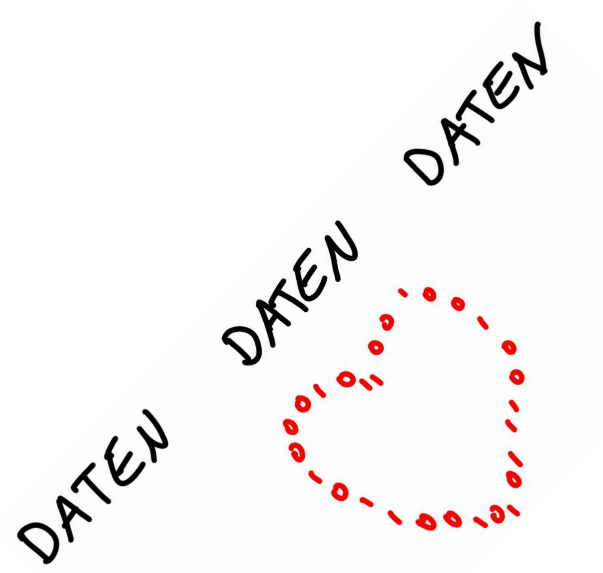
https://frida.re/docs/stalker/

FЯIDA 💕 iOS Debugserver

I really shouldn’t tell you this!
… but most of the time, this actually works without crashing the target.

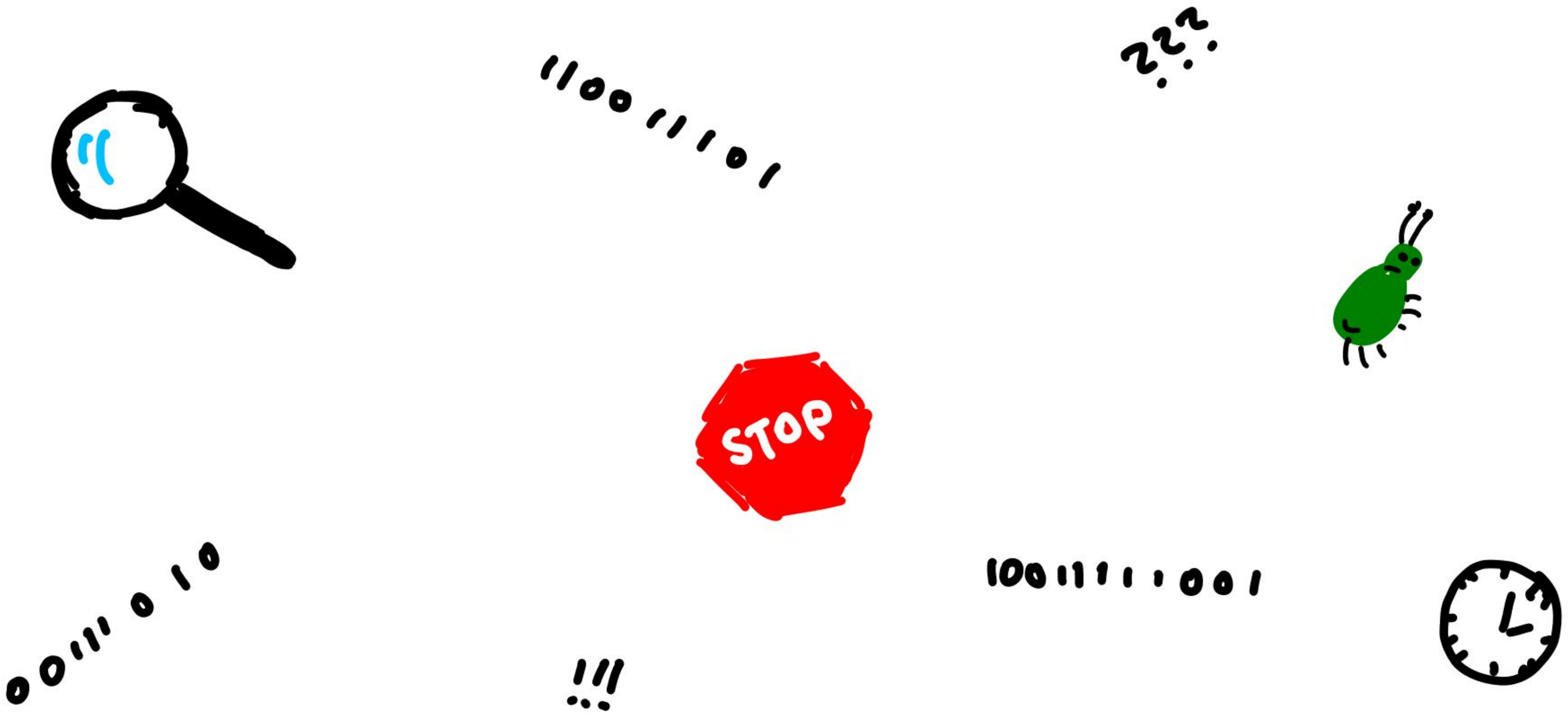


Data, Data, Data

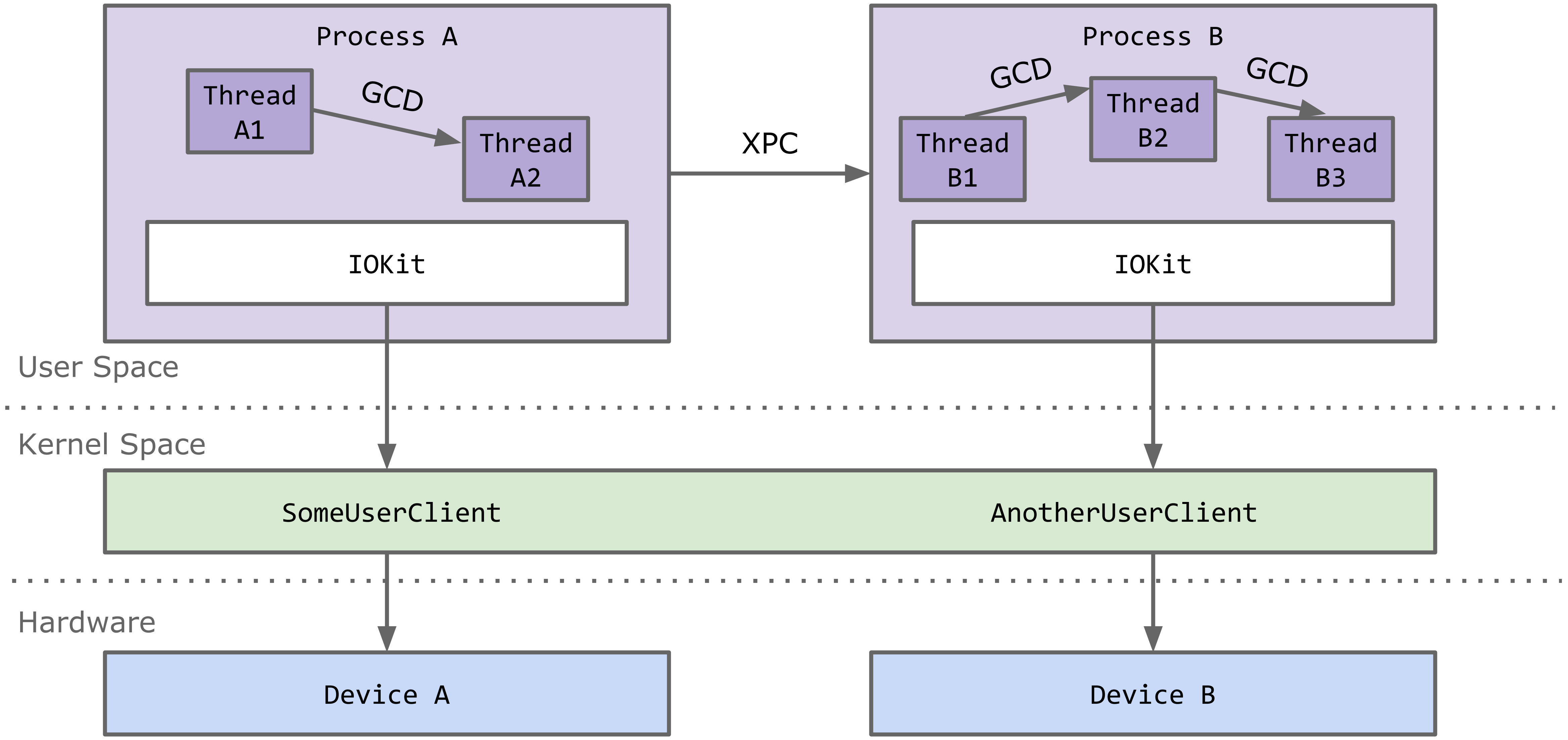


Data??

Attach IDA Pro to the iOS Debugserver, pause execution a few times while sending lots of data, set a few breakpoints, eventually locate functions that process data …



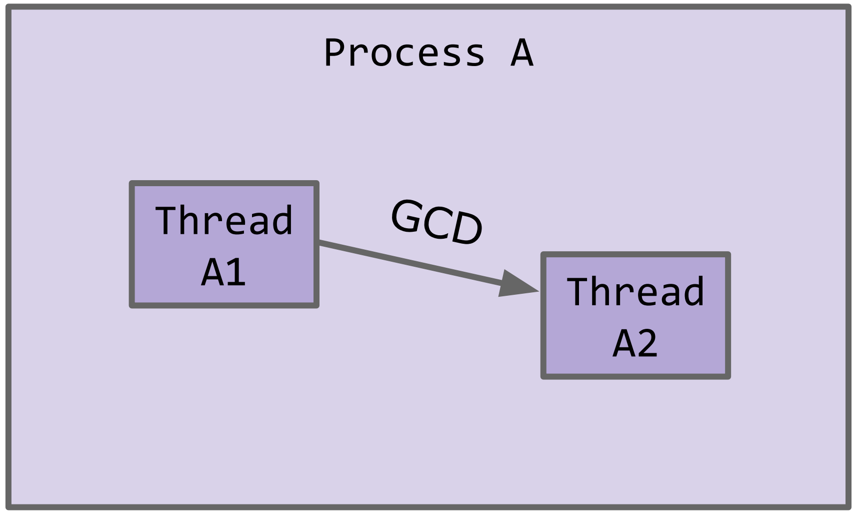
Data!

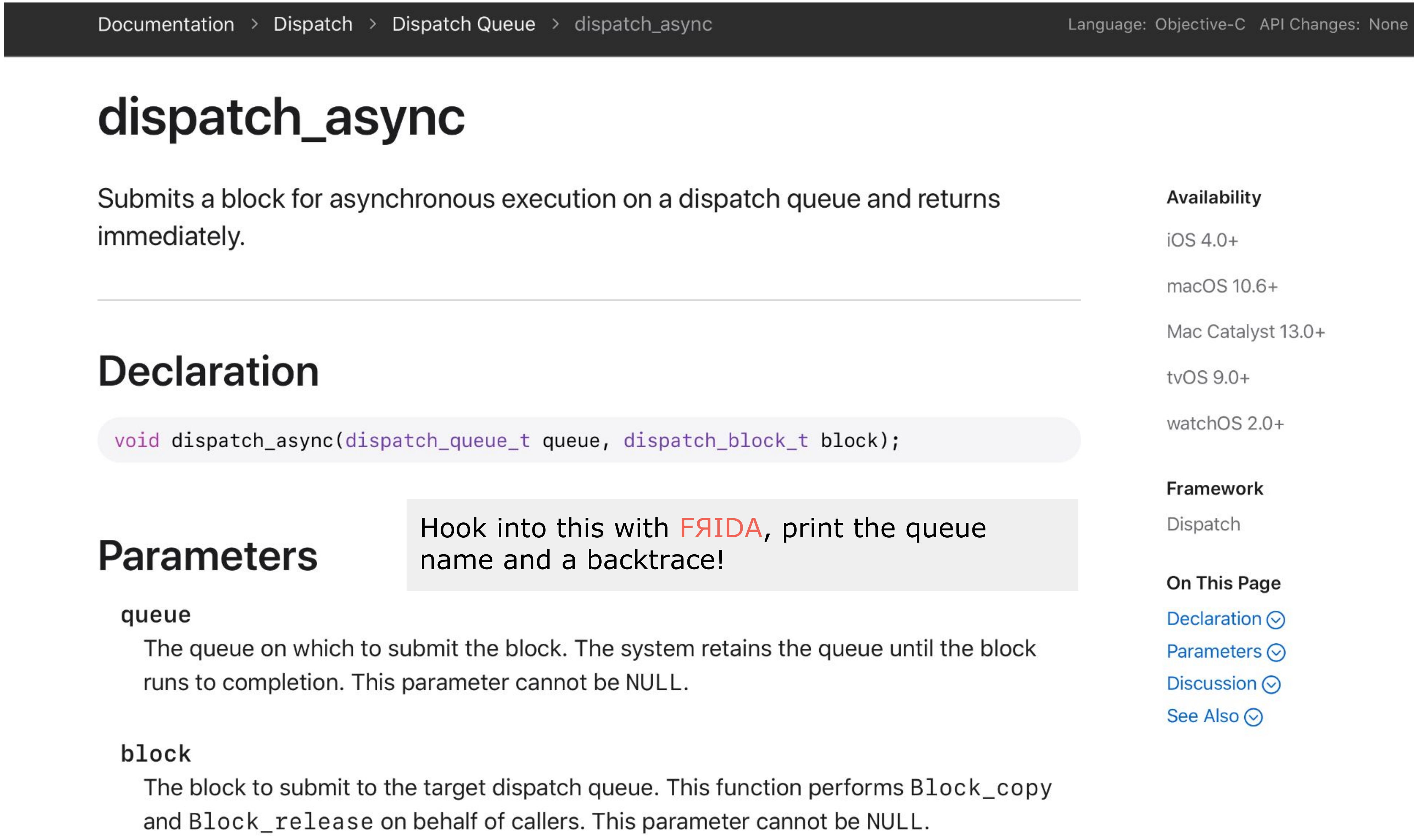

Process A
Thread A1
GCD
Thread A2
IOKit
XPC
Process B
GCD
GCD
Thread B1
Thread B2
Thread B3
IOKit
User Space
Kernel Space
SomeUserClient
AnotherUserClient
Hardware
Device A
Device B

Grand Central Dispatch

Grand Central Dispatch

- Apple’s implementation of threads.
- `libdispatch` is open-source, makes writing hooks easier.
- Thread switches often happen for data processing…
- … we want to find data handlers, this will help a lot!





Documentation › Dispatch › Dispatch Queue › dispatch_async

Language: Objective-C API Changes: None

dispatch_async

Submits a block for asynchronous execution on a dispatch queue and returns immediately.

Declaration

```
void dispatch_async(dispatch_queue_t queue, dispatch_block_t block);
```

Parameters

Hook into this with FЯIDA, print the queue name and a backtrace!

`queue`
The queue on which to submit the block. The system retains the queue until the block runs to completion. This parameter cannot be NULL.

`block`
The block to submit to the target dispatch queue. This function performs `Block_copy` and `Block_release` on behalf of callers. This parameter cannot be NULL.

Availability

iOS 4.0+

macOS 10.6+

Mac Catalyst 13.0+

tvOS 9.0+

watchOS 2.0+

Framework

Dispatch

On This Page

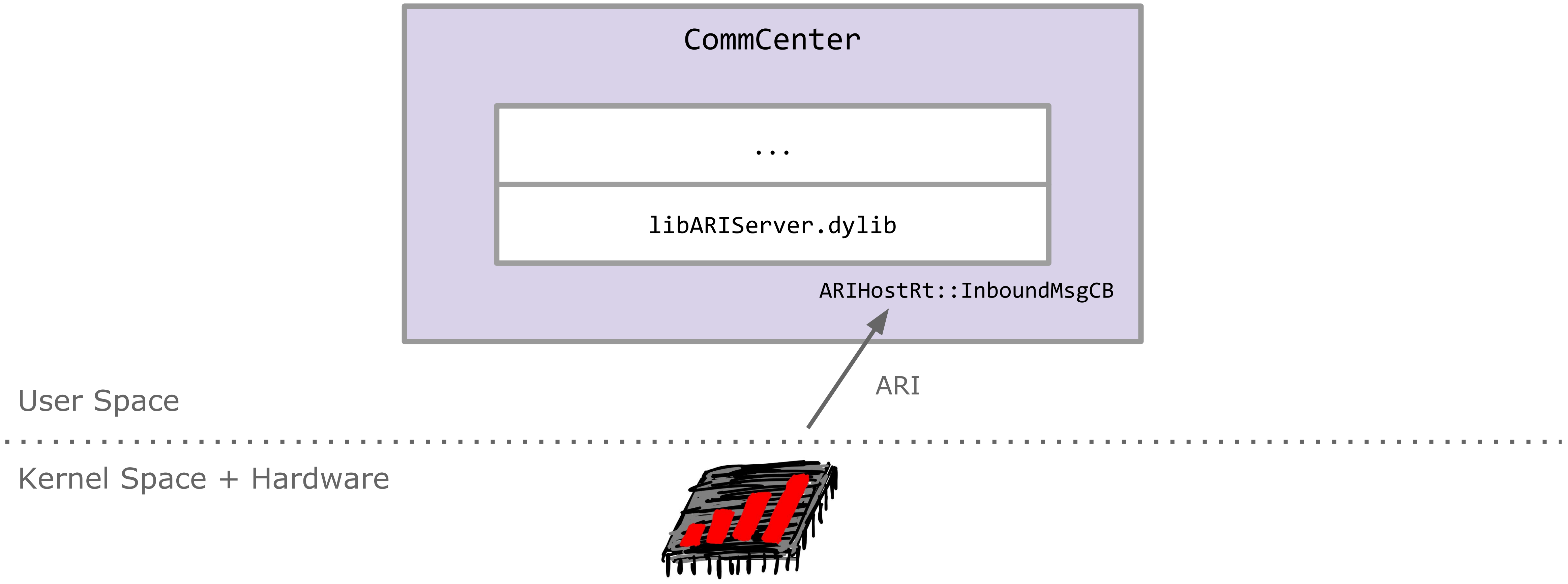
Declaration

Parameters

Discussion

See Also

https://github.com/seemoo-lab/frida-scripts/blob/main/scripts/libdispatch.js
https://developer.apple.com/documentation/dispatch/1453057-dispatch_async



Using `libdispatch.js` will help identifying functions like `ARIHostRt::InboundMsgCB`.

Demo: `libdispatch.js`

```
frida -U CommCenter --no-pause -l libdispatch.js
```

```
dispatch_async
Calling queue: AriHostRt
Callback function: 0xd47c1581e85b6dc8 libARIServer.dylib!invocation function for block in
AriHostRt::InboundMsgCB(unsigned char*, unsigned long)
Backtrace:
0x1e85b267c libARIServer.dylib!AriHostRt::InboundMsgCB(unsigned char*, unsigned long)
0x1e85aa4c0 libARIServer.dylib!AriFramer::fmrMsgCb(unsigned char*, unsigned long)
0x1e85aa10c libARIServer.dylib!AriFramer::IpcDataCb(unsigned char*, unsigned long, void*)
0x1a2316fd0 libdispatch.dylib!_dispatch_call_block_and_release
0x1a2318ac8 libdispatch.dylib!_dispatch_client_callout
0x1a231fc08 libdispatch.dylib!_dispatch_lane_serial_drain
0x1a2320734 libdispatch.dylib!_dispatch_lane_invoke
0x1a232a528 libdispatch.dylib!_dispatch_workloop_worker_thread
0x1ea60b908 libsystem_pthread.dylib!_pthread_wqthread
```

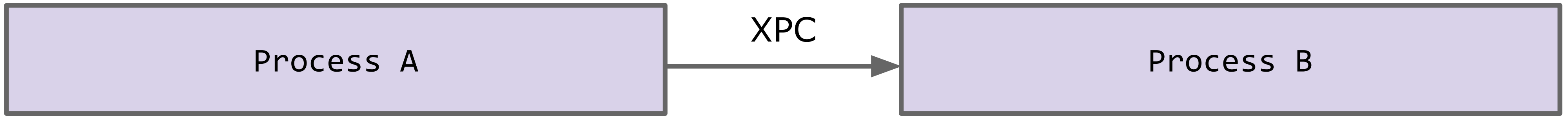
Demo: `libdispatch.js`

```
frida -U bluetoothd --no-pause -l libdispatch.js
```

```
dispatch_async
Calling queue: com.apple.AppleConvergedIPC.pci_client_2
Callback function: 0xd0779e01ec456e80 AppleConvergedTransport.dylib!invocation function for block in
pci::transport::th::readAsync()
Backtrace:
0x1ec456d24 AppleConvergedTransport.dylib!invocation function for block in
pci::transport::th::readAsync()
0x1ec45940c AppleConvergedTransport.dylib!pci::transport::bind::~bind()
0x1ec459ec0 AppleConvergedTransport.dylib!pci::transport::task::complete(int, unsigned int, unsigned
int)
0x1ec4507d4 AppleConvergedTransport.dylib!invocation function for block in
pci::transport::bh::generateCallback(std::__1::shared_ptr<pci::transport::task>, char const*, bool)
0x1ec4513d8 AppleConvergedTransport.dylib!pci::transport::bh::ioCompletion(void*, int, void*)
0x1ad20c134 IOKit!IODispatchCalloutFromCFMessage
0x1ad20c214 IOKit!_IODispatchCalloutWithDispatch
0x1a2333c18 libdispatch.dylib!dispatch_mig_server
0x1a2318ac8 libdispatch.dylib!_dispatch_client_callout
...
```

XPC

Cross-Process Communication



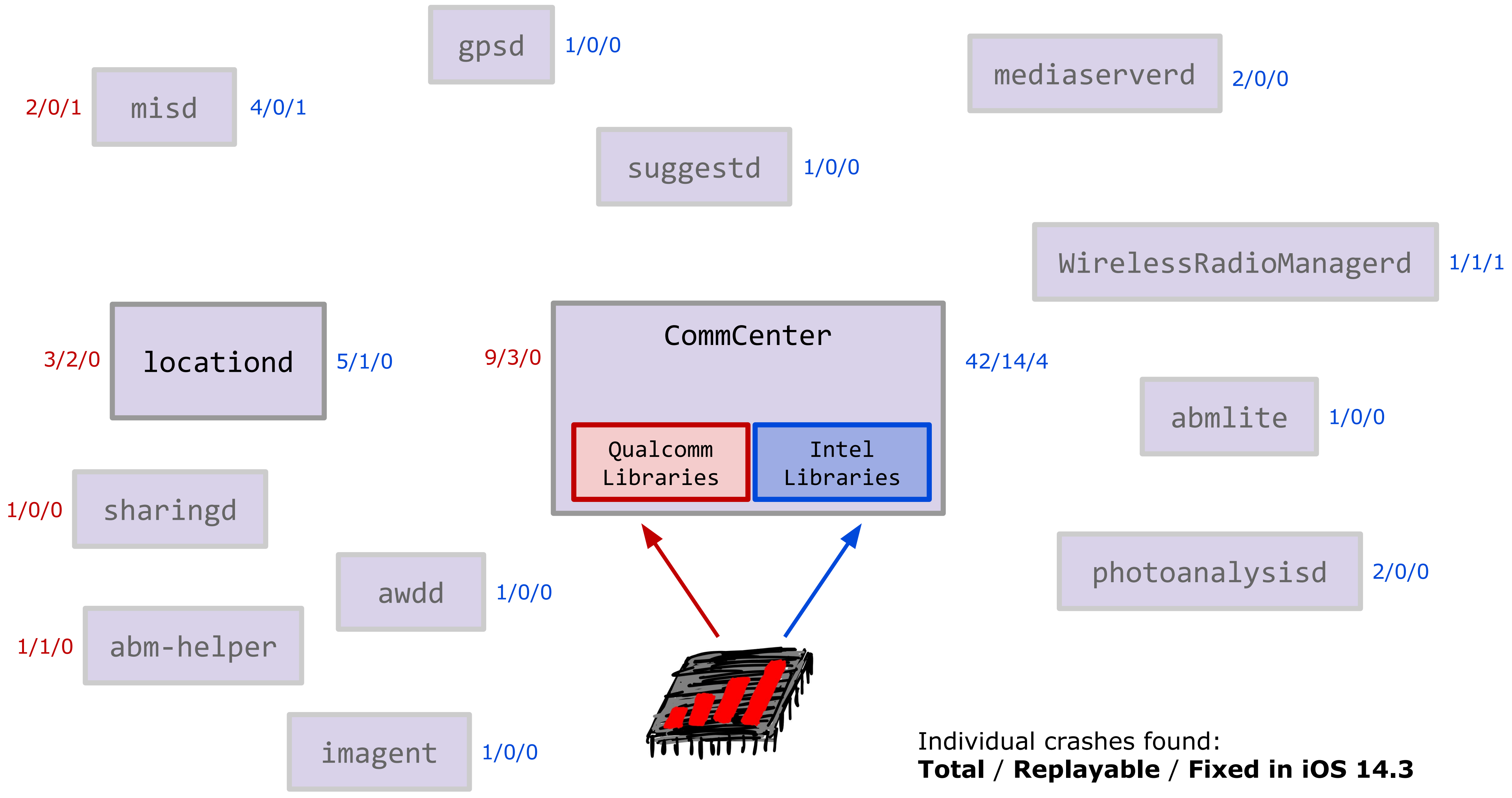
- Daemons interact a lot with each other.
- Also used between applications and daemons.
- Not necessarily wireless packets, but at least important state information is visible here.
- Hardware test functions might be implemented in XPC.
- Permissions are managed by `launchd`, which starts processes and bootstraps the underlying Mach ports for XPC.

https://github.com/evilpenguin/XPCSniffer
https://github.com/hot3eed/xpcspy

Demo: `xpcspy`

```
xpcspy -U -n bluetoothd
```

- Loooooooots of data!
- FЯIDA-based implementation crashes on XPC-heavy targets like `CommCenter`.



Individual crashes found:
Total / Replayable / Fixed in iOS 14.3

Example: SMS Simulator

- Remote Code Execution via SMS has a history since iOS 2.
- This was revisited for iOS 11.
- The `CommCenter` XPC interface `com.apple.commcenter.xpc` exposes test functions.
- SMS can be simulated using this (see `kSimulateSmsReceived`).

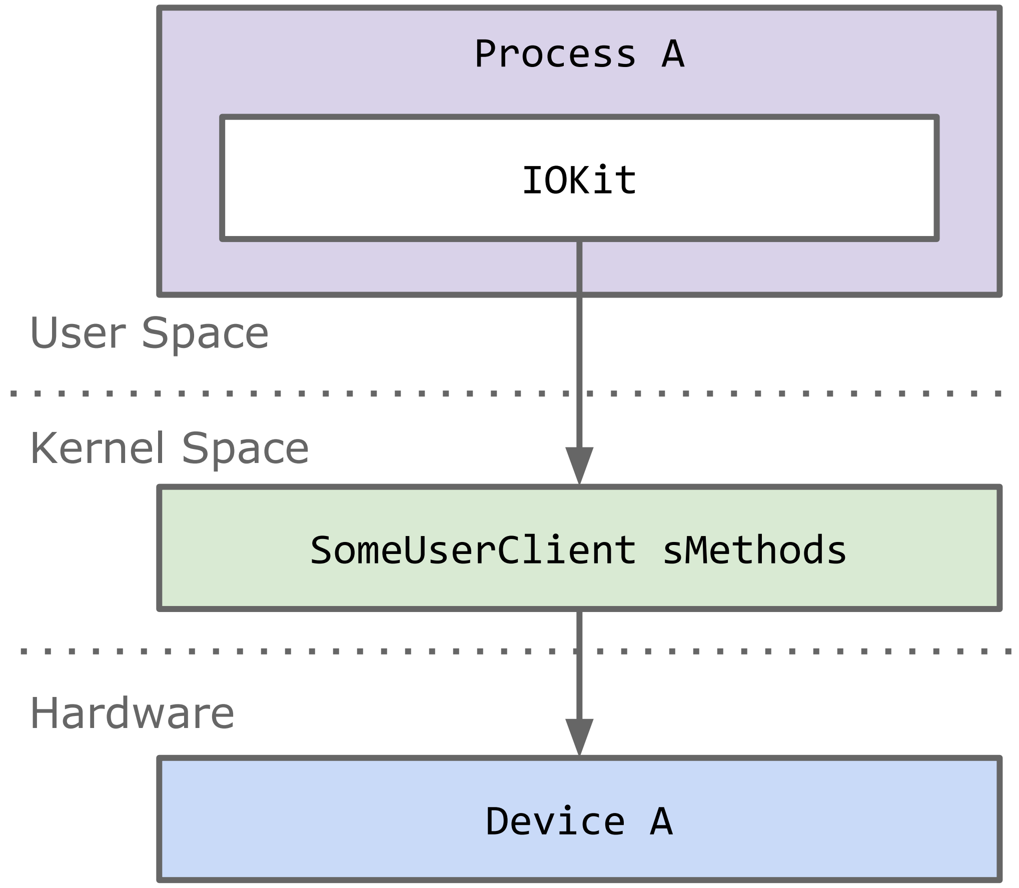
- Simulated SMS go a slightly different path in `CommCenter` than real SMS.
- SMS parsing has so many debug prints, it is probably the most tested functionality.

https://github.com/googleprojectzero/iOS-messaging-tools/tree/master/SmsSimulator

IOKit + Mach

IOKit

- IOKit drivers export functions from the kernel.
- Function parameter types are predefined and are verified.
- A process can call these functions via a `UserClient`.



https://github.com/robre/frida-scripts/blob/master/iotracy.js
https://github.com/unixb0y/IOKitLibAnalysis
"Strolling into Ring-0 via I/O Kit Drivers" talk by @patrickwardle

Demo: IOKit.js

```
frida -U nearbyd --no-pause -l IOKit.js
```

```
mach_task_self: 259
port: 18959 | desc: AppleSPURoseDriverUserClient(0x100027287)
port: 19215 | desc: AppleKeyStoreUserClient(0x100027286)
port: 19475 | desc: AppleKeyStoreUserClient(0x100027289)
port: 42755 | desc: AppleSPUUserClient(0x100027288)
done creating mappings.

{^-^} IOConnectCallMethod:
> mach_port_t connection: 0xa703 => AppleSPUUserClient(0x100027288)
> uint32_t selector: 0x3
> const uint64_t *input:
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  28                                               (
> uint32_t inputCnt: 0x1
> const void *inputStruct: 0x0
> size_t inputStructCnt: 0x0
> uint64_t *output: 0x0
> uint32_t *outputCnt: 0x0
> void *outputStruct: 0x16fd0e2f0
> size_t *outputStructCnt: 0x16fd0e2c8
```

AppleSPUUserClient::extPerformCommandMethod

(APCheckIn

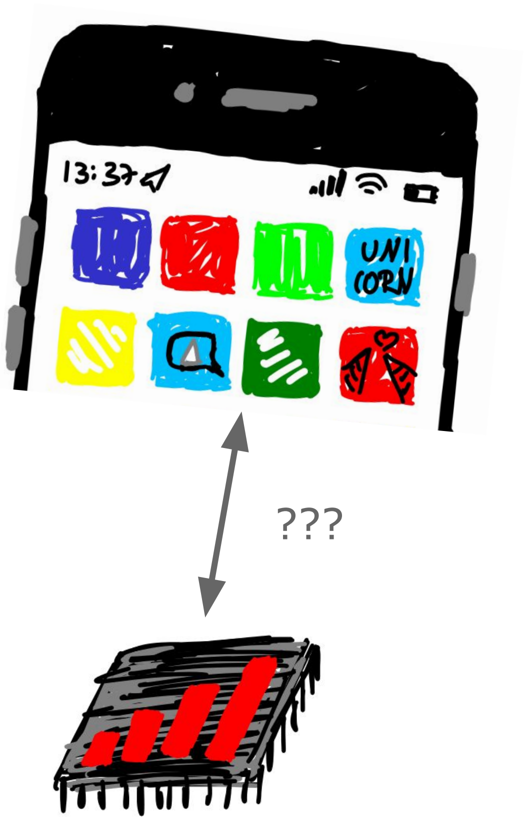
Mach Messages

- XPC and IOKit are internally just implemented as Mach messages.
- There might as well just be plain Mach messages communicating with the kernel etc.
- We can double-check what we are missing by hooking into them directly.
- … better only use this for debugging purposes!

https://github.com/seemoo-lab/frida-scripts/blob/main/scripts/mach_msg.js

Conclusion

1. Find chip interfaces: IOKit, Mach
2. Find protocol handlers: GCD
3. Decode proprietary protocols: Shared libraries, debug logs.
4. Inject custom payloads: Call functions with Frida or XPC.



Q&A

https://github.com/seemoo-lab

Twitter: @naehrdine

jiska@bluetooth.lol

Interacting with *OS Hardware from User Space