black hat USA 2022

August 10-11, 2022

BRIEFINGS

TruEMU: an extensible, open-source, whole-system iOS emulator

Speaker: Trung Nguyen

Team members: Antonio Binachi, Kyungtae Kim, Dave Jing Tian

whoami

- Trung Nguyen Hoang - @ntrung03
- Undergraduate CS student at Purdue University
- Focus on macOS/iOS research
- Used to blog about CTF challenges
  - https://trungnguyen1909.github.io/blog/

PURDUE UNIVERSITY | Department of Computer Science

Agenda

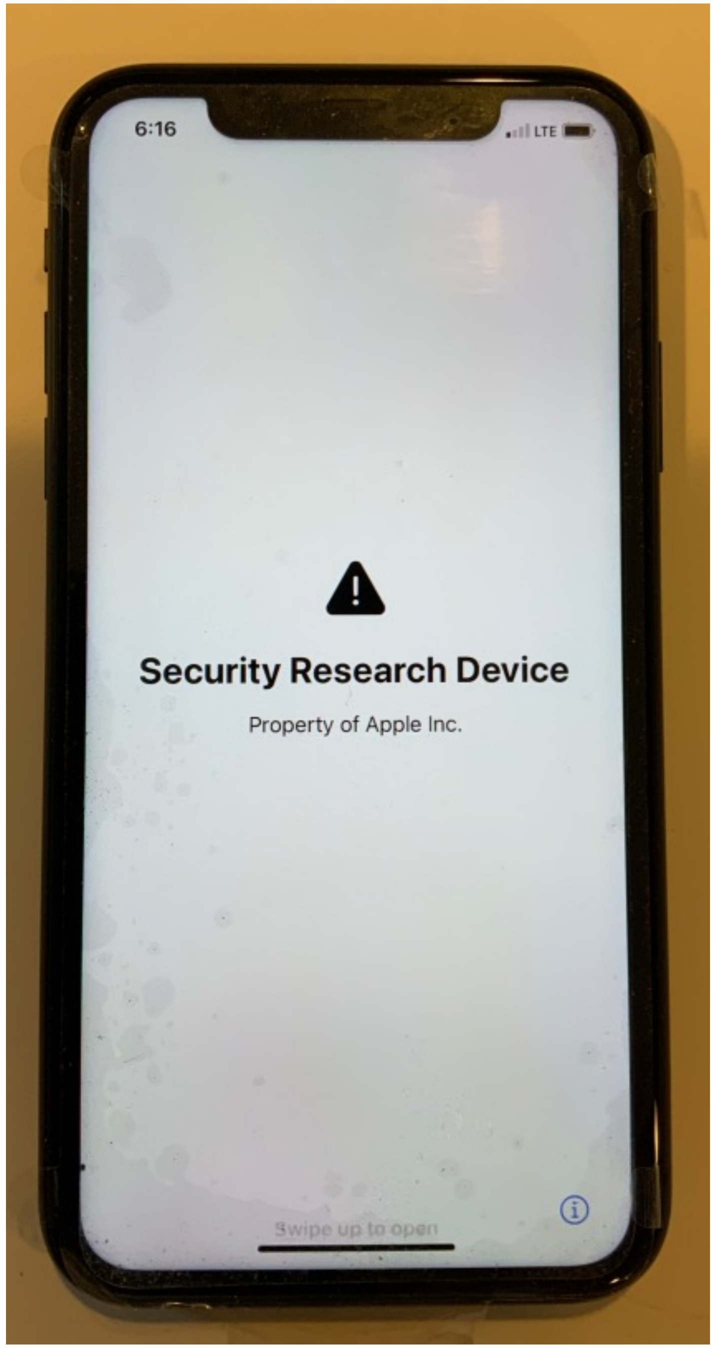
- Current state of iOS Research
- TruEmu's design goal
- Implementing TruEmu
- Using TruEmu for Research
- TruEmu's Future and Roadmap

We need to talk about iOS research

How current iOS research is done

Using real devices

- Security Research Device Program by Apple



How current iOS research is done

Using real devices

How current iOS research is done

Using real devices

- Security Research Device Program by Apple
- Apple Internal devices (dev-fused devices)
- Off-the-shelf jailbroken devices

How current iOS research is done

Using real devices

Pinned Tweet

ax🔥🌸mX @axi0mX · 9/27/19

EPIC JAILBREAK: Introducing checkm8 (read "checkmate"), a permanent unpatchable bootrom exploit for hundreds of millions of iOS devices.

Most generations of iPhones and iPads are vulnerable: from iPhone 4S (A5 chip) to iPhone 8 and iPhone X (A11 chip).

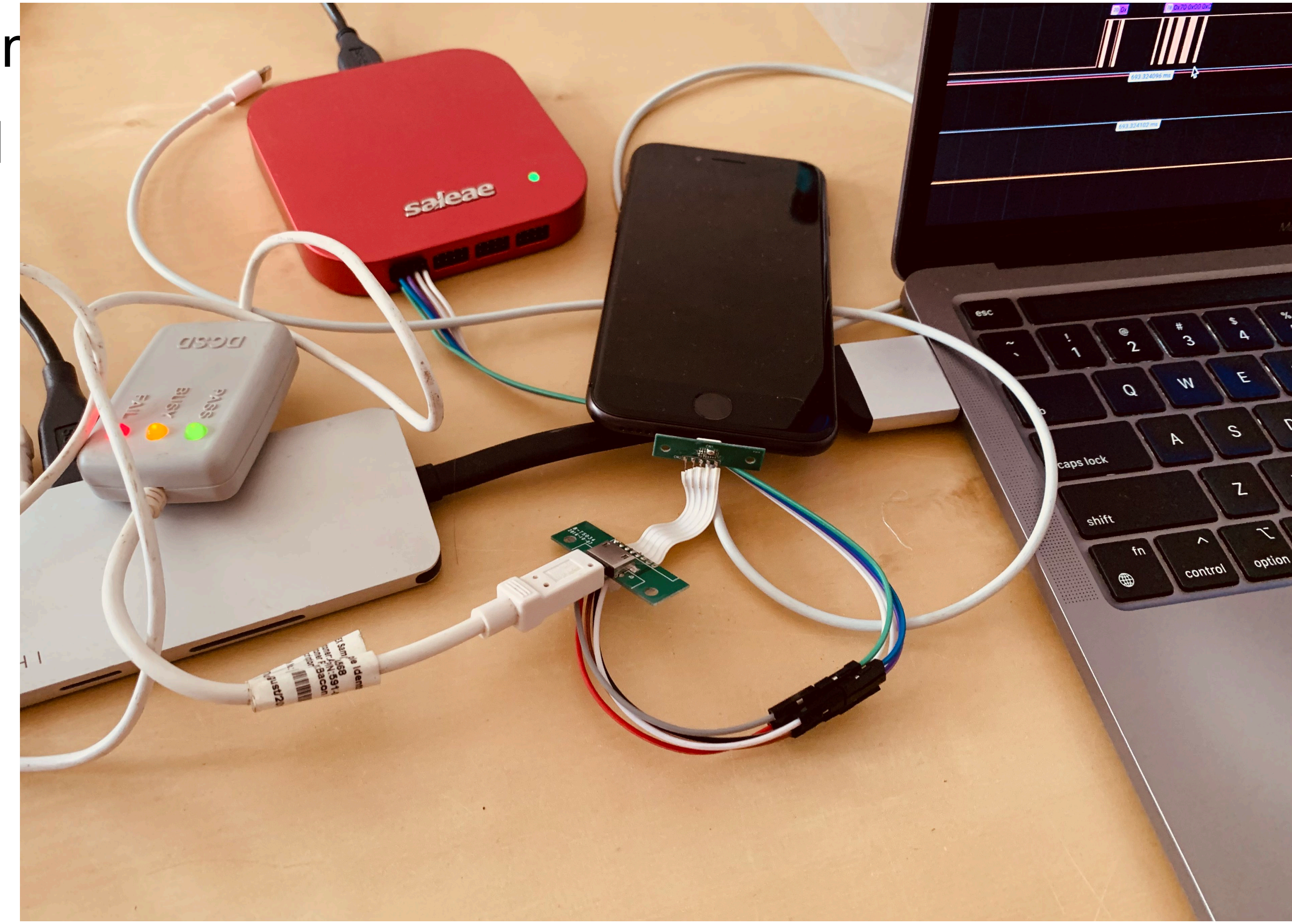
axi0mX/ipwndfu

open-source jailbreaking tool for many iOS devices

2 Contributors · 144 Issues · 7k Stars · 2k Forks

github.com
GitHub - axi0mX/ipwndfu: open-source jailbreaking tool for many iOS devices

982 · 7,870 · 15.7K



How current iOS research is done

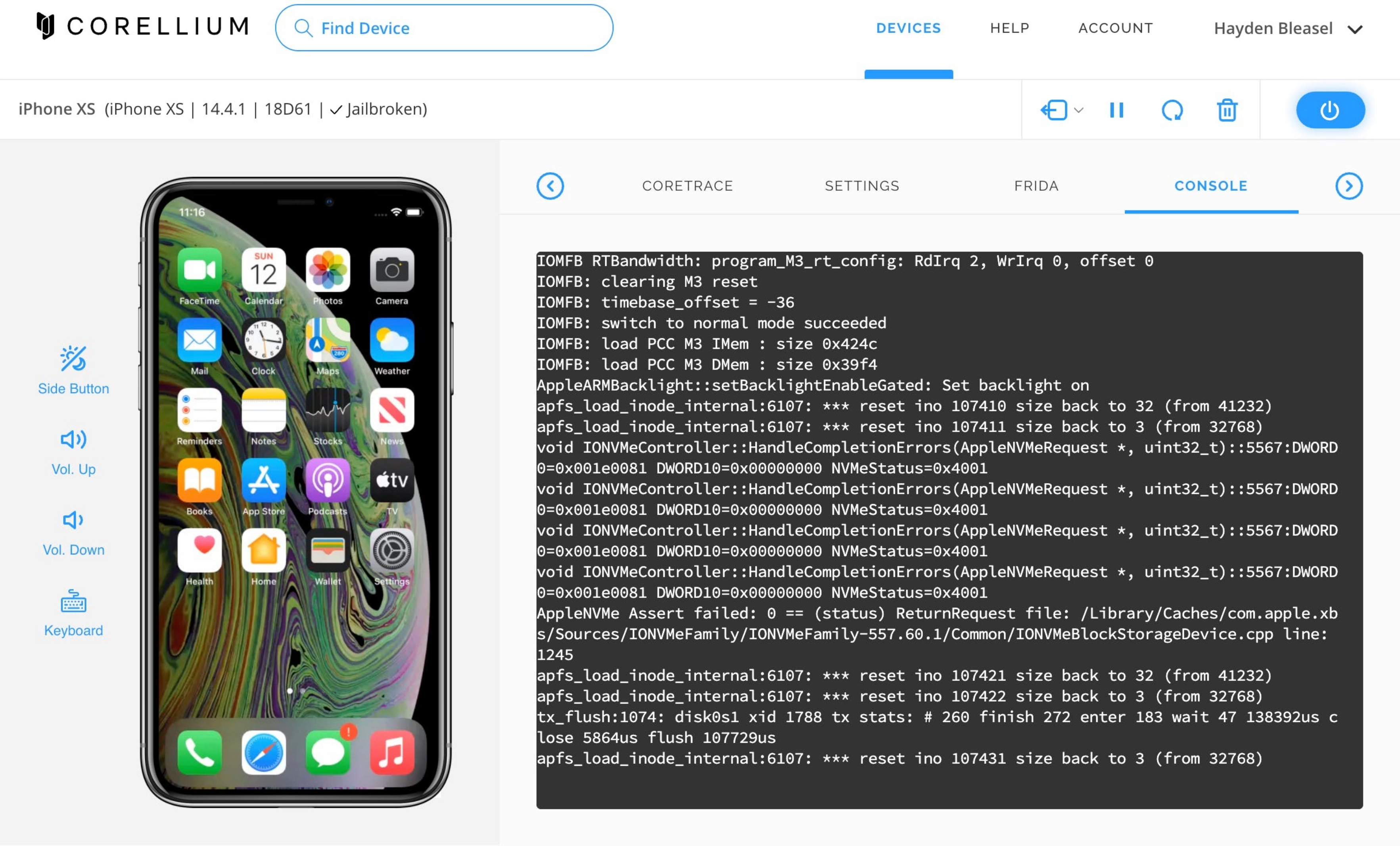
Using real devices

- Security Research Device Program by Apple
- Apple Internal devices (dev-fused devices)
- Off-the-shelf jailbroken devices
- Off-the-shelf non-jailbroken devices
- ARM Macs

How current iOS research is done

Emulation comes to the rescue

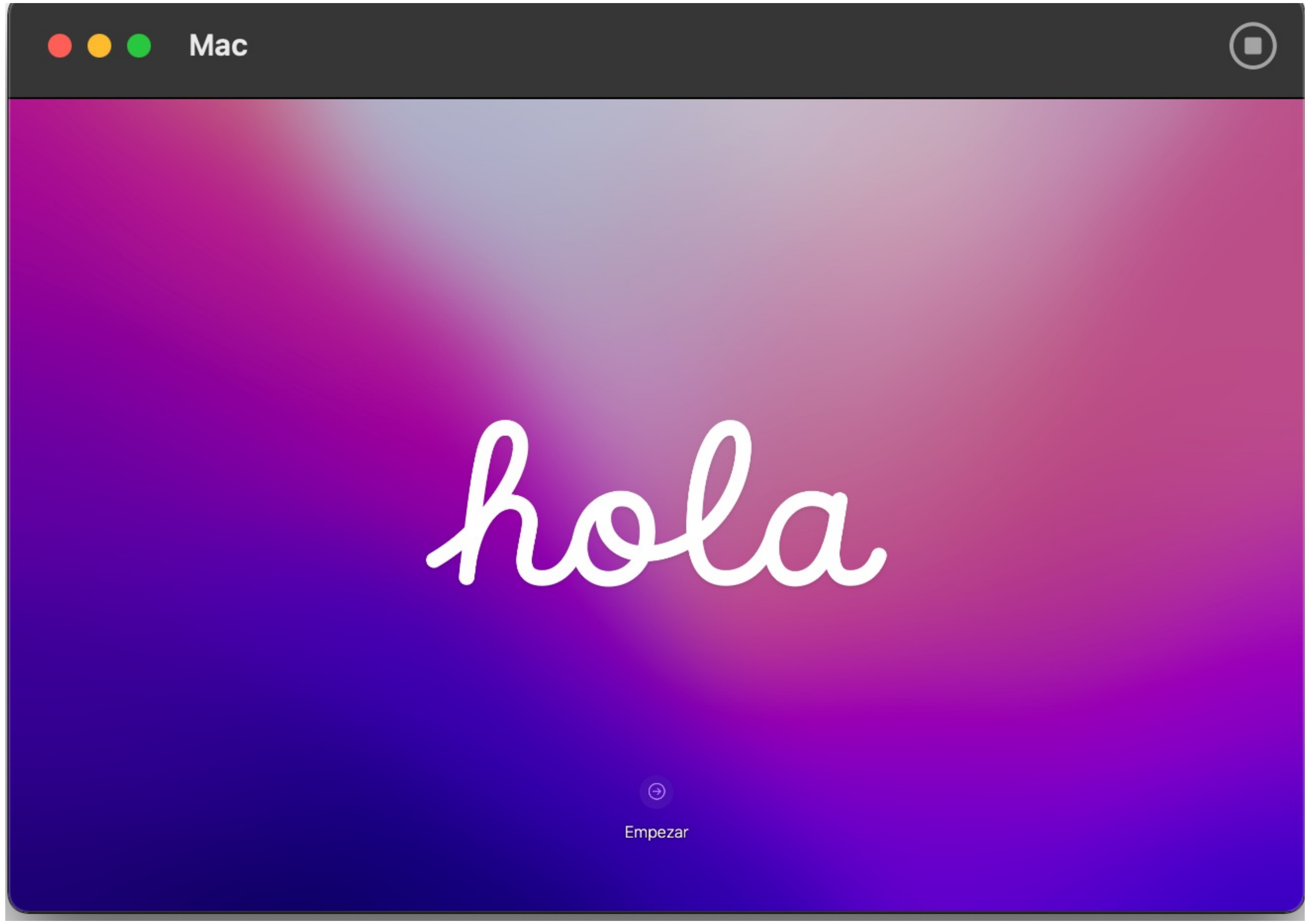
- Third party commercial iOS emulator



How current iOS research is done

Emulation comes to the rescue

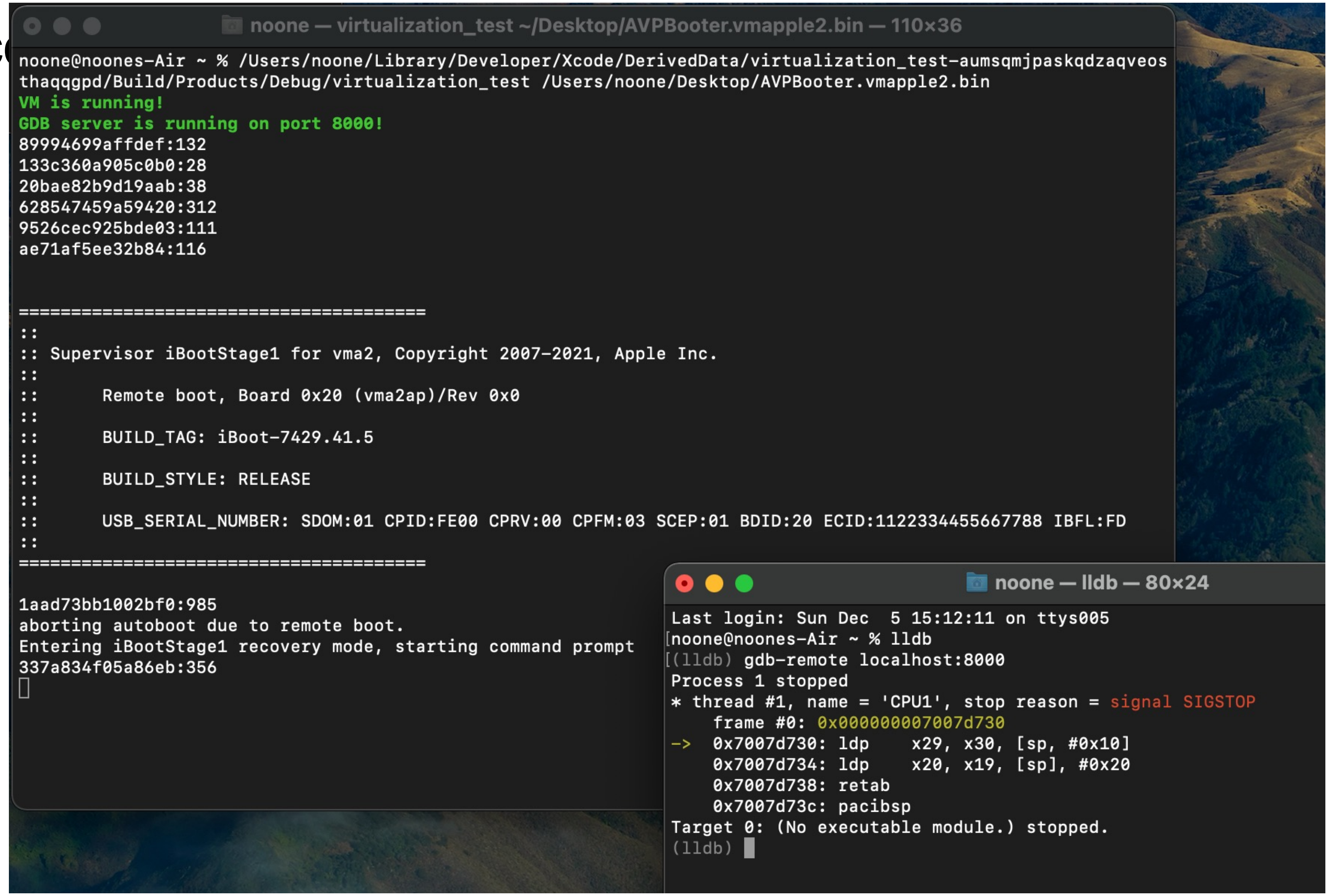
- Third party commercial iOS emulator
- VMApple



How current iOS research is done

Emulation comes to the rescue

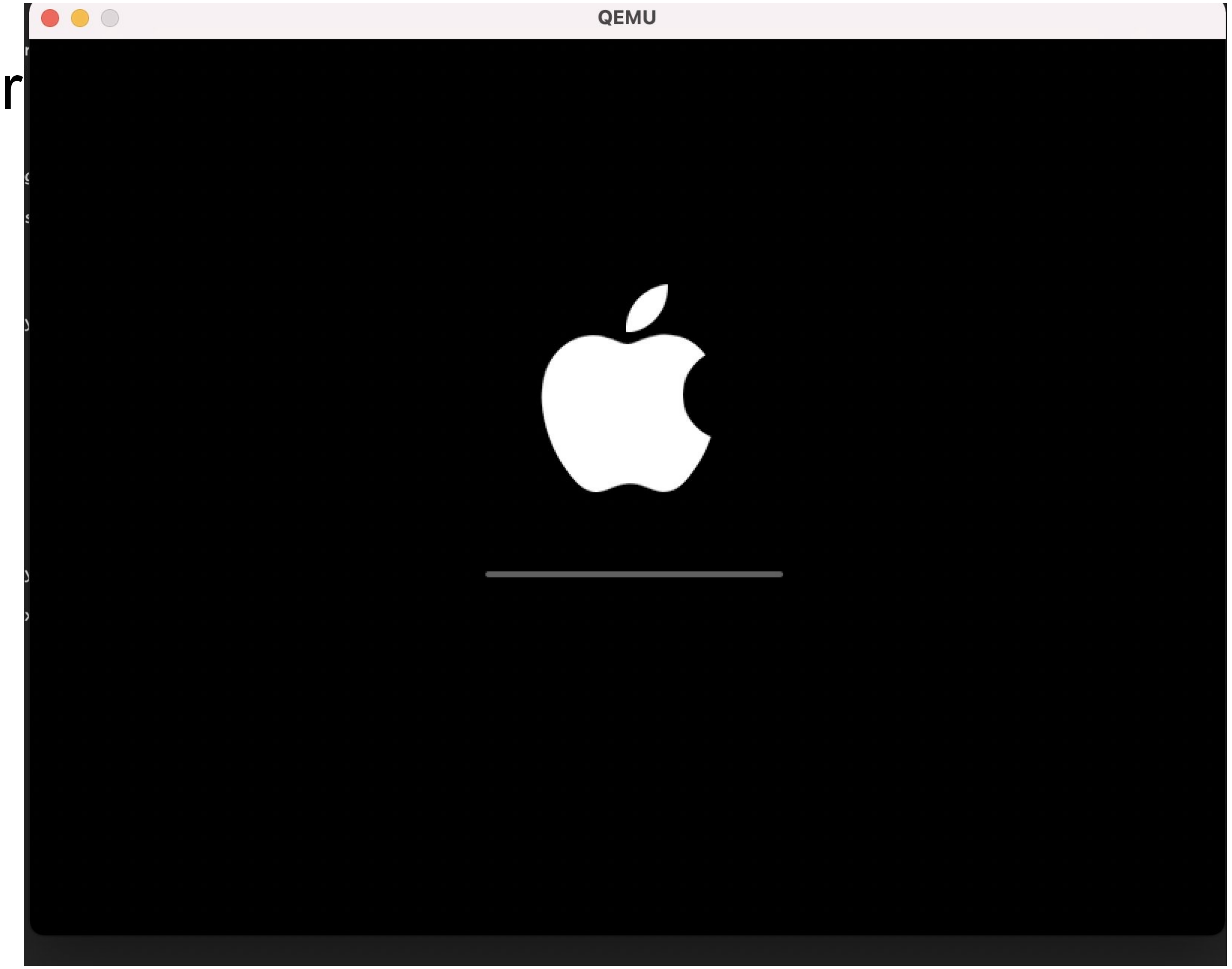
- Third party c
- VMApple



How current iOS research is done

Emulation comes to the rescue

- Third party commercial iOS emulator
- VMApple
- Aleph Security's xnu-qemu-arm64



How current iOS research is done

Shortcomings of Aleph Security's xnu-qemu-arm64

- Supports only 2 iOS version
- Limited hardware support
- Hard to maintain and also abandoned

TruEmu came to the rescue

TruEmu came to the rescue

TruEmu’s design goal

- Free-to-use iOS emulator for security research
- Out-of-box support for a wide range of iOS versions
- Easy to debug
- Can be used for fuzzing

TruEmu came to the rescue

TruEmu's notable features

- Model actual hardware
- Support from iOS 14 to the latest iOS 16
- iPhone 6S SecureROM
- Out-of-box Kernel debugging support
- USB support (with Firmware Restore)
- Apple's custom CPU features (SPRR/GXF, custom PAC)
- We are Open source
  - http://github.com/TrungNguyen1909/qemu-t8030

Implementing TruEmu

Implementing TruEmu

How does a new device get modeled

1. Look for information from the device tree
2. Build a stub model and log MMIO accesses
3. A mix of dynamic and static reverse engineering the protocol
4. Write code to emulate needed responses
5. Profit

Implementing TruEmu

1. Reading the device tree

- Can be found in iOS IPSW
- Contains a rich amount of peripherals information for iOS
- Used to match driver

Implementing TruEmu

1. Reading the device tree

- Contains a rich amount of peripherals information for iOS
- Used to match driver

```
~/Projects/iOSQEMU
❯ ./dt/dt Firmware/all_flash/DeviceTree.n104ap.im4p.out gpio
#interrupt-cells                 0x00000002
interrupt-controller                                                                  ||
compatible                       67 70 69 6f 2c 74 38 30  33 30 00 67 70 69 6f 2c    |gpio,t8030.gpio,|
                                 73 35 6c 38 39 36 30 78  00                         |s5l8960x.|
interrupt-parent                 0x0000001a
interrupts                       83 00 00 00 84 00 00 00  85 00 00 00 86 00 00 00    |................|
                                 87 00 00 00 88 00 00 00  89 00 00 00                |............|
#gpio-int-groups                 0x00000007
reg                              00 00 10 3c 00 00 00 00  00 00 10 00 00 00 00 00    |...<............|
#gpio-pins                       0x000000d4
AAPL,phandle                     0x00000023
device_type                      interrupt-controller
#address-cells                   0x00000000
role                             AP
name                             gpio
```

2. Building the stub model

- Map a dummy memory region to the MMIO address
- Log accesses and back trace, disassemble the related code
- Try driving the interrupt lines to see how iOS responses

```
disp0: base_reg_write @ 0x0000000000050030 value: 0x00000000016c0000
stacktrace: pc: 0xfffffff00977b740 tid: 0xffffffe19b6a9d10
0xfffffff00977b740,0xfffffff00977b6a8,0xfffffff00977b6a8,0xfffffff00975973c,0xfffffff00975bd90,0xfffffff0097473a8,0xfffffff0097b8528,0xfffffff0097bb3a4,0xfffffff0096f830c,0xfff
ffff008050900,0xfffffff0096f7b4c,0xfffffff0096f6be4,0xfffffff0096df594,0xfffffff0096e213c,0xfffffff00808cb78,0xfffffff0096e14e0,0xfffffff00809a98c,0xfffffff007b25190,0xfffffff0
07a30e9c,0xfffffff007a021d8,0xfffffff007a1d810,0xfffffff007b4a434,0xfffffff007b57094,0xfffffff00811c5f4,

disp0: base_reg_write @ 0x0000000000050040 value: 0x0000000001c70000
stacktrace: pc: 0xfffffff00977b740 tid: 0xffffffe19b6a9d10
0xfffffff00977b740,0xfffffff00977b6a8,0xfffffff00977b6a8,0xfffffff00975975c,0xfffffff00975bd90,0xfffffff0097473a8,0xfffffff0097b8528,0xfffffff0097bb3a4,0xfffffff0096f830c,0xfff
ffff008050900,0xfffffff0096f7b4c,0xfffffff0096f6be4,0xfffffff0096df594,0xfffffff0096e213c,0xfffffff00808cb78,0xfffffff0096e14e0,0xfffffff00809a98c,0xfffffff007b25190,0xfffffff0
07a30e9c,0xfffffff007a021d8,0xfffffff007a1d810,0xfffffff007b4a434,0xfffffff007b5700xfffffff00977b740,0xfffffff00977b6a8,0xfffffff00977b6a8,0xfffffff0097597a0,0xfffffff00975bd90
,0xfffffff0097473a8,0xfffffff0097b8528,0xfffffff0097bb3a4,0xfffffff0096f830c,0xfffffff008050900,0xfffffff0096f7b4c,0xfffffff0096f6be4,0xfffffff0096df594,0xfffffff0096e213c,0xff
fffff00808cb78,0xfffffff0096e14e0,0xfffffff00809a98c,0xfffffff007b25190,0xfffffff007a30e9c,0xfffffff007a021d8,0xfffffff007a1d810,0xfffffff007b4a434,0xfffffff007b57094,0xfffffff
00811c5f4,
```

TruEmu’s implementation

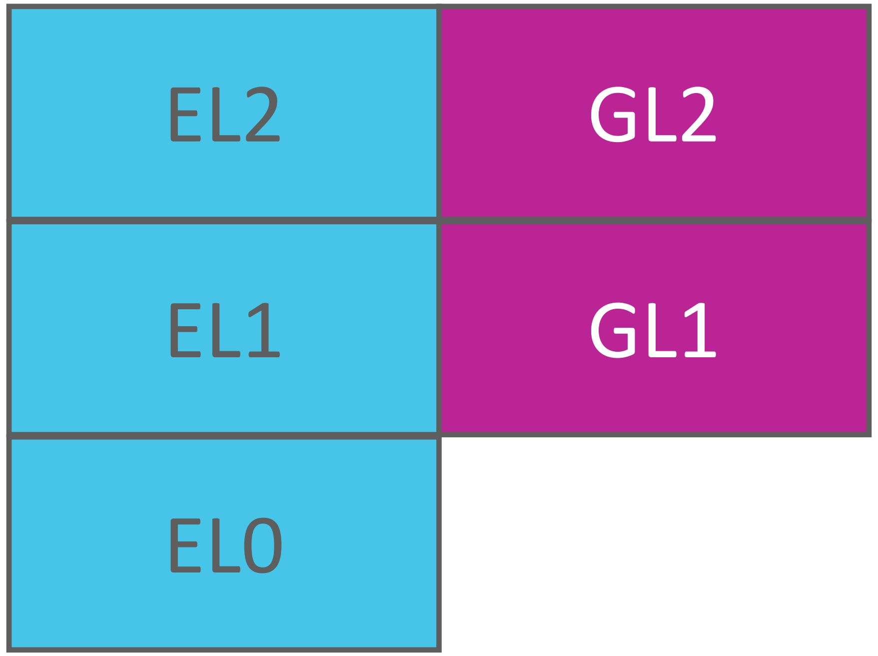
SPRR/GXF

- Used in both iOS kernel and browser
- Apple’s custom privilege-level
- New levels are created laterally from ARM’s

Implementing TruEmu

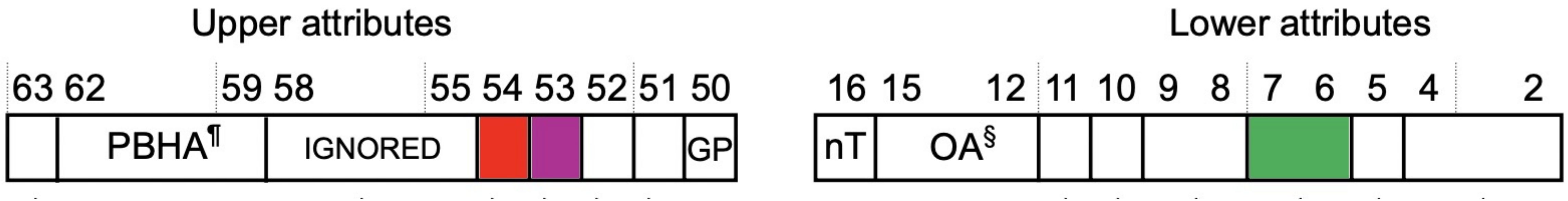
SPRR/GXF

- Used in both iOS kernel and browser
- Apple's custom privilege-level
- New levels are created laterally from ARM's
- GXF: Guarded eXecution Feature
- GENTER: ELx to GLx
- GEXIT: GLx to ELx
- Guarded mode can have different page permission



SPRR/GXF

Attribute fields for VMSAv8-64 stage 1 Block and Page descriptors



Index: 0bNNNN

Permission register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Permission: 0bGGEE

Permission bits on page table becomes index in a system register

SPRR/GXF

Permission: 0bGGEE

- Jumping to GLx code from ELx code causes a GXF abort
- Except: No write in ELx if exec in GLx

0b00	---
0b01	r-x
0b10	r--
0b11	rw-

Page Protection Layer

- PPL: Page Protection Layer
- Security-sensitive code (Page table, TrustCache) are in PPL
- Normal kernel code (__TEXT, __TEXT_EXEC): 0x24ac000 bytes (≈ 37MiB)
- PPL kernel code (__PPLTEXT): 0x19844 bytes (≈ 102KiB) (368x smaller)
- PPL runs in Guarded mode
- PPL can jump to normal kernel code, but not the other way around

Bulletproof JIT

- Browsers use JIT to compile JavaScript code into native code to speed up execution
- It creates a page that is both writable and executable to store the result and execute

Problem with normal JIT

- JIT pages constantly need to change between write and execute mode
- Changing permission would normally require trapping to kernel and some TLB flushes
- Those are slow and hurt performance

SPRR/GXF + iOS

SPRR comes to the rescue

- Just flip the permission bit from userspace
- pthread_jit_write_protect_np:

Read-Execute

```
movk x0, 0xc118
movk x0, 0xffff, lsl 16
movk x0, 0xf, lsl 32
movk x0, 0, lsl 48
ldr x0, [x0]                ; 0xd8
msr s3_6_c15_c1_5, x0
isb
movk x1, 0xc118
movk x1, 0xffff, lsl 16
movk x1, 0xf, lsl 32
movk x1, 0, lsl 48
ldr x8, [x1]                ; 0xd9
mrs x9, s3_6_c15_c1_5
bics xzr, x8, x9
b.eq 0x24d0
```

Read-Write

```
movk x0, 0xc110
movk x0, 0xffff, lsl 16
movk x0, 0xf, lsl 32
movk x0, 0, lsl 48
ldr x0, [x0]                    ; 0xd8
msr s3_6_c15_c1_5, x0
isb
movk x1, 0xc110
movk x1, 0xffff, lsl 16
movk x1, 0xf, lsl 32
movk x1, 0, lsl 48
ldr x8, [x1]                    ; 0xd9
mrs x9, s3_6_c15_c1_5
b 0x24c8
```

Implementing TruEmu

SPRR/GXF

- We implemented these custom CPU logics in TCG
- New instructions need to be decoded
- Page table permission logic needs to be modified
- Limitation: Changes to permission register requires an expensive TLB flush due to QEMU TLB's limitation

Implementing TruEmu

Why we want USB Emulation?

- Restoring: We can now install iOS like a real device
- Networking: SSH?
- Connect to Xcode: Install and run apps (not yet)

Implementing TruEmu

Challenges of USB Emulation

- Problem 1: iOS only has drivers for Synopsys USB controllers

Implementing TruEmu

SYNOPSYS®

Solutions Products Support Company

Home ▾ / DesignWare IP ▾ / Interface IP ▾ / USB ▾ / DesignWare Hi-Speed USB 2.0 On-the-Go Controller

DesignWare Hi-Speed USB 2.0 On-the-Go Controller

The DesignWare® Hi-Speed USB 2.0 On-The-Go (HS OTG) Controller provides designers with high-quality USB IP for the most demanding USB 2.0 peripherals. The controller performs as a standard Hi-Speed Dual-Role Device (DRD), operating as either a USB 2.0 Hi-Speed peripheral, or Hi-Speed USB 2.0 Host. Based on Synopsys' success in building and deploying Hi-Speed USB 2.0 Host, Device and PHY designs, the DesignWare USB 2.0 HS OTG Controller incorporates Synopsys expertise in Reuse Methodology, Constrained Random Verification, and USB PHY interoperability to deliver flexible, quality IP in Verilog source. The controller is optimized for area- and power-sensitive markets such as Internet of Things (IoT).

- DesignWare IP Prototyping Kit for USB 2.0 HS OTG
- DesignWare IP Prototyping Kits
- DesignWare USB 2.0 Controller IP

SYNOPSYS

SYNOPSYS.COM | REGISTRATION HELP | 帮助 | 幫助 | ヘルプ

SYNOPSYS

Home ▼ / Desi

DesignWa

The DesignWare®
USB IP for the mos
Role Device (DRD),
Synopsys' success
DesignWare USB 2
Constrained Rand
source. The contr

- DesignWare IP F
- DesignWare IP F
- DesignWare USI

SolvNetPlus

Sign In

Username

Please enter a username

Password

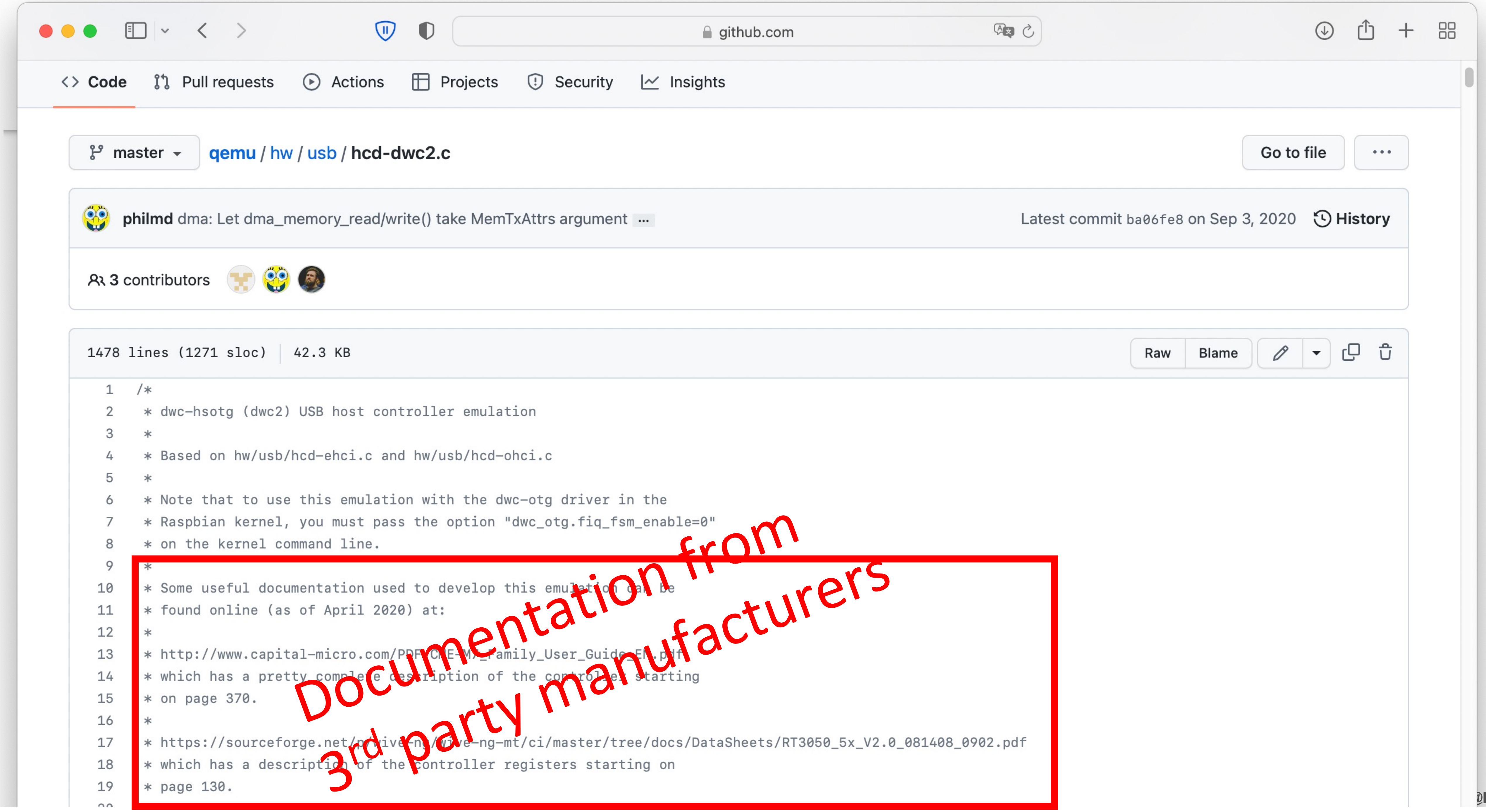
Sign In

Need help signing in?

REGISTER - CREATE ACCOUNT   FORGOT PASSWORD

SIGN IN SUPPORT | TERMS OF USE | PRIVACY POLICY

Implementing TruEmu



Documentation from 3rd party manufacturers

Implementing TruEmu

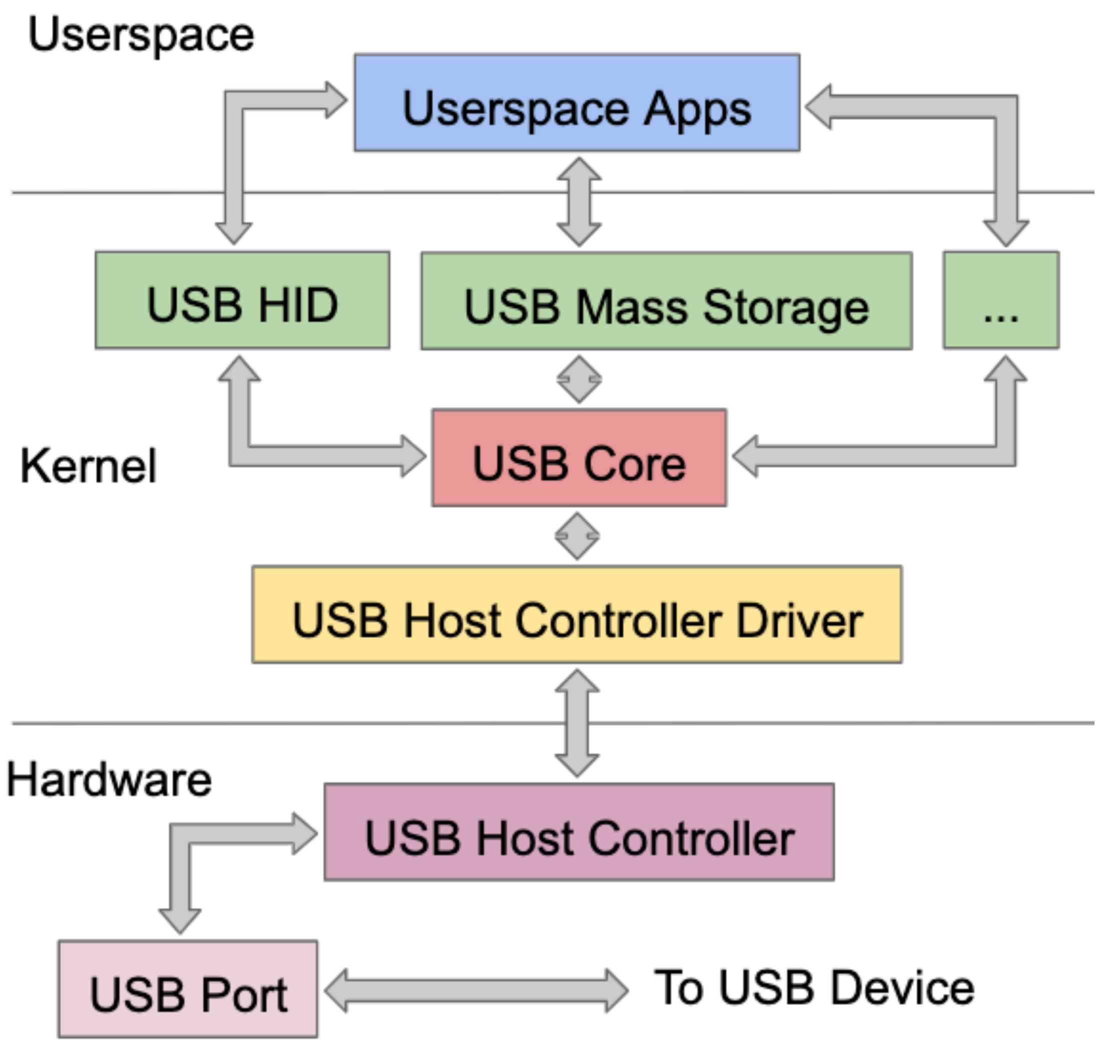
Challenges of USB Emulation

- Problem 1: But iOS only has drivers for Synopsys USB controllers
- Problem 2: Actual iPhone 11 uses newer Synopsys Dual-Role-Device, but documents are sparse for those
  - → We used to modify device tree to make iOS loads old drivers for Synopsys OTG
- We eventually implemented the new Synopsys USB controller

Implementing TruEmu

USB bus

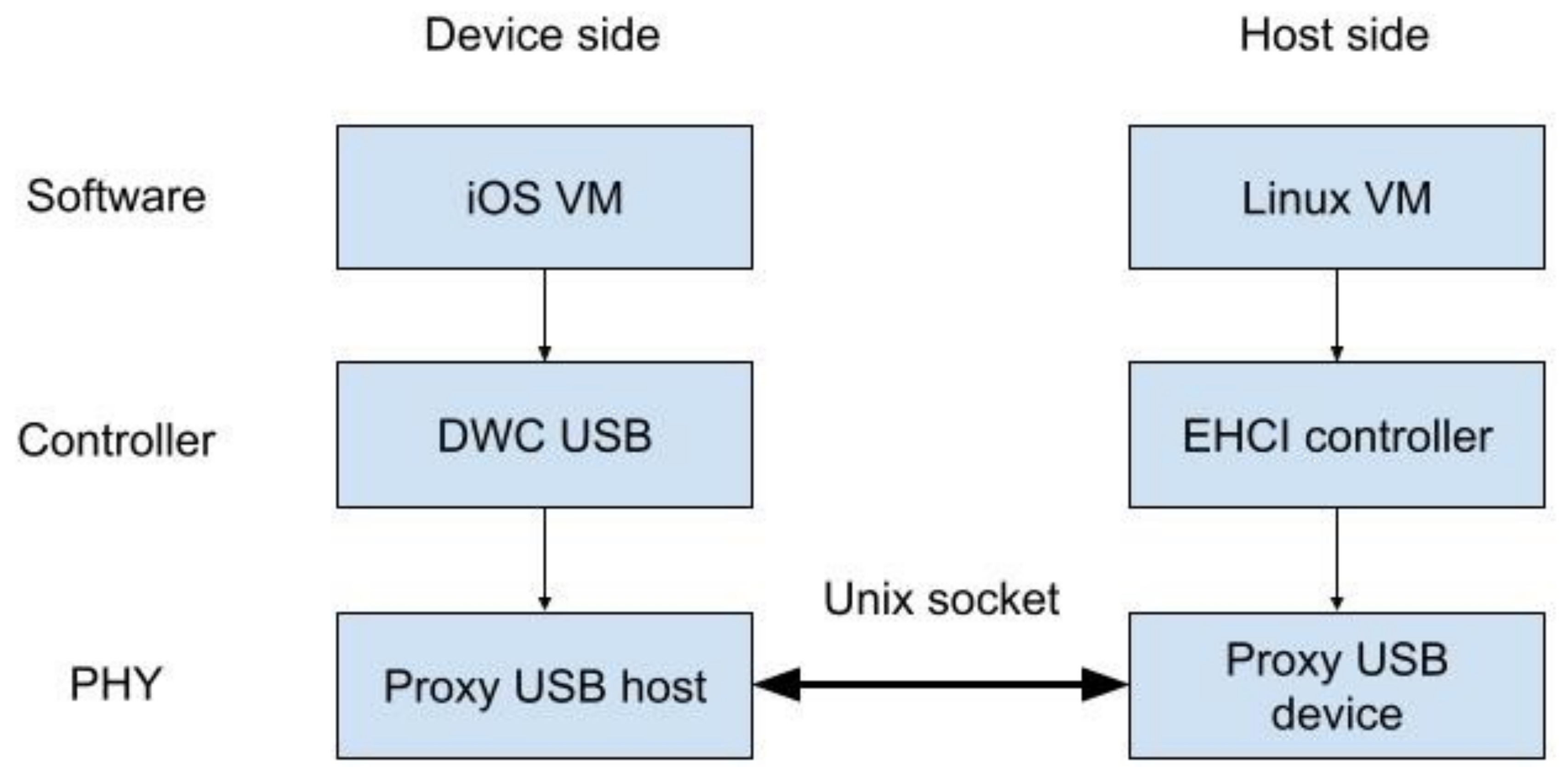
- There are 2 USB sides: host and device
- iOS supports both
- iOS uses device mode to connect with PCs
- QEMU does not support device mode



Implementing TruEmu

USB bus

- We connect the iOS VM to a Linux VM using UNIX pipes



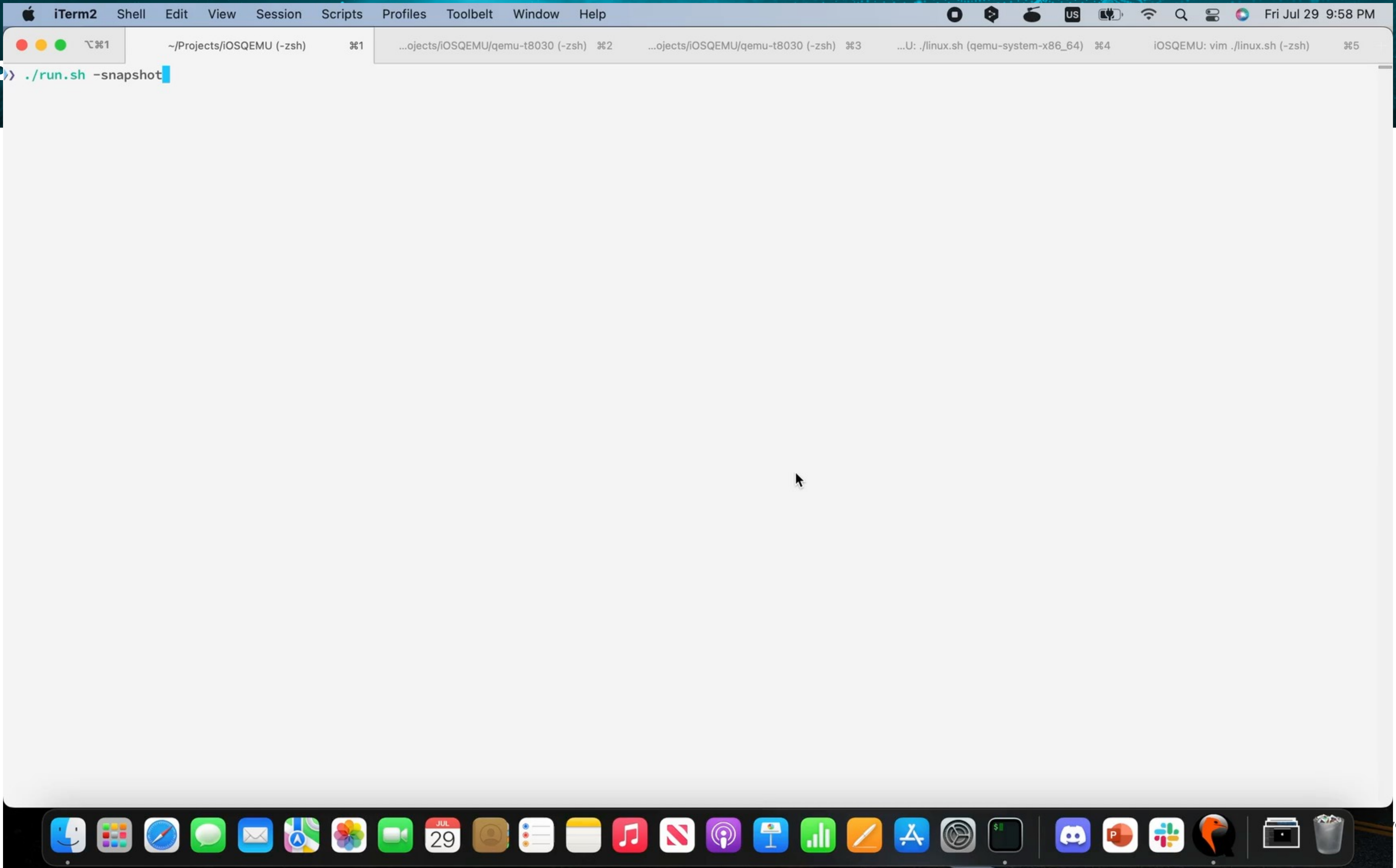
Using TruEmu for research

Using TruEmu for research

Emulation - Demo

⌥⌘1 | iOSQEMU: ./run_demo.sh (-zsh) ⌘1 | iOSQEMU: caffeinate -dism (-zsh) ⌘2 | iOSQEMU: ./linux.sh (qemu-system-x86_64) ⌘3

```
❯ ./run_demo.sh
```


iTerm2
Shell
Edit
View
Session
Scripts
Profiles
Toolbelt
Window
Help
Fri Jul 29 9:58 PM
~/Projects/iOSQEMU (-zsh)
...ojects/iOSQEMU/qemu-t8030 (-zsh)
...ojects/iOSQEMU/qemu-t8030 (-zsh)
...U: ./linux.sh (qemu-system-x86_64)
iOSQEMU: vim ./linux.sh (-zsh)
./run.sh -snapshot

Using TruEmu for research

Emulation - Demo

- We went through the Restore process of iOS
- We got a bash shell and explored iOS using various commands
- We SSHed into our iOS machine

Using TruEmu for research

Reverse Engineer - Demo

Using TruEmu for research

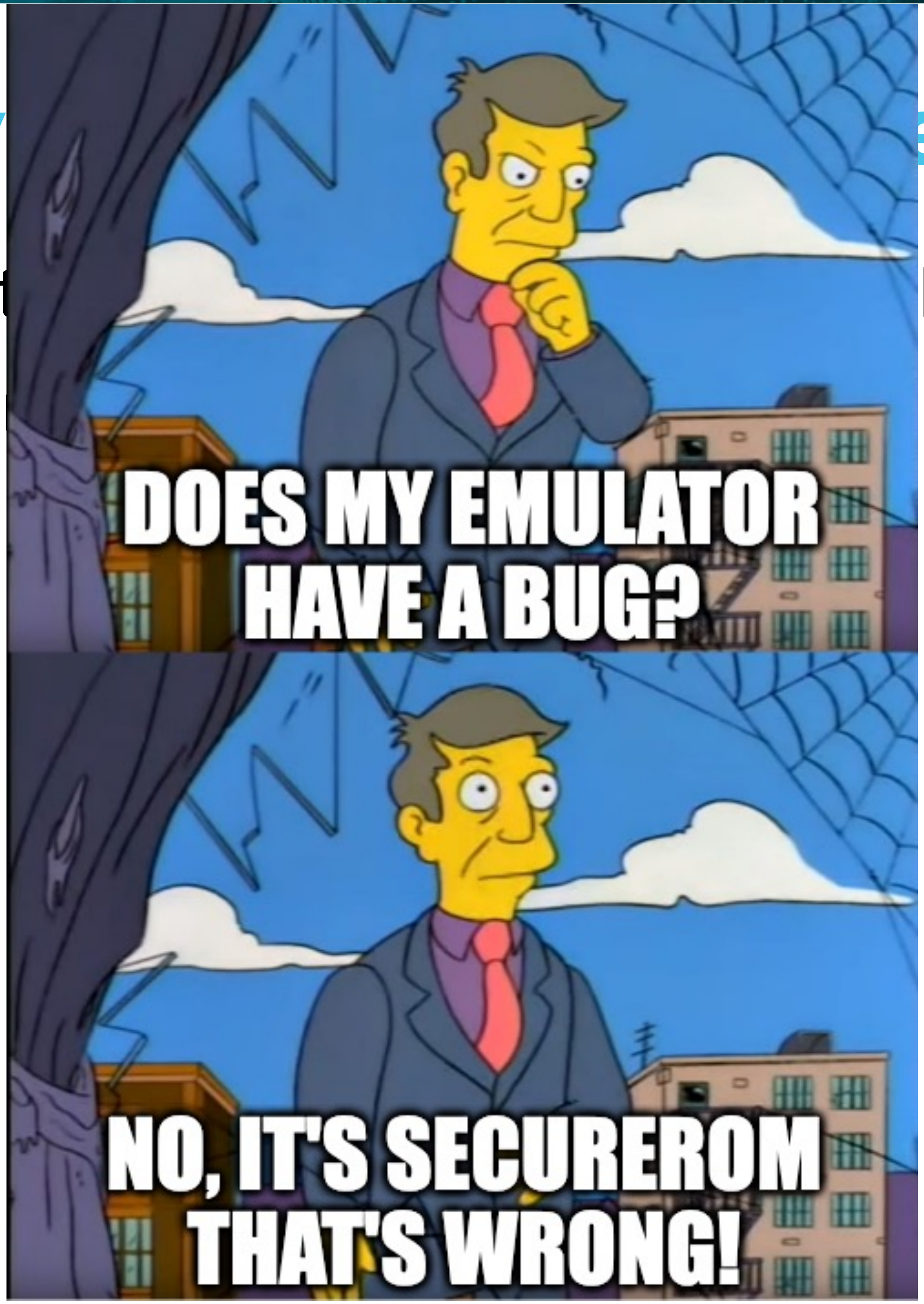
Reverse Engineer - Demo

- We set breakpoints, stepping, and exploring SecureROM memory
- We also found a bug in SecureROM that prevents it from resetting on panic

Using TruEmu for research

Rev ... emo

- We set breakpoints, st ... ROM memory
- We also found a bug i ... from resetting on panic



Using TruEmu for greybox fuzzing

Snapshot

- Our iOS boot time is great (5s), but still not good enough for fuzzing
- Using VM snapshots to start at the fuzzable state immediately (0.5s / cycle)

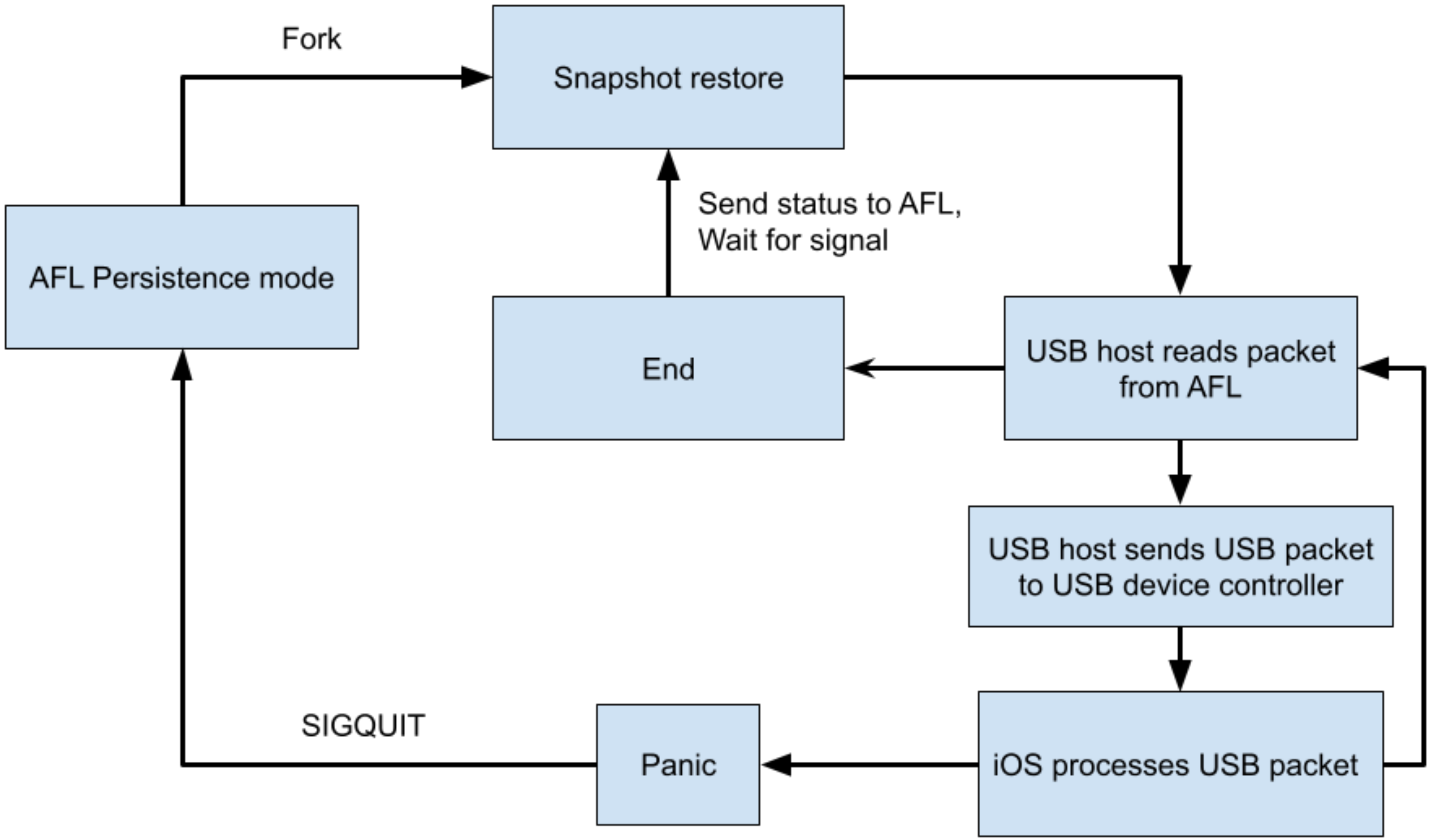
Using TruEmu for greybox fuzzing

Code coverage

- AFL uses code coverage to maximize the number of paths reached
- We are running emulation using TCG, which is a JIT compiler
- TCG compiles emulated code into basic blocks
- → Records coverage when a block is being executed

Using TruEmu for greybox fuzzing

USB fuzzing



Using TruEmu for greybox fuzzing

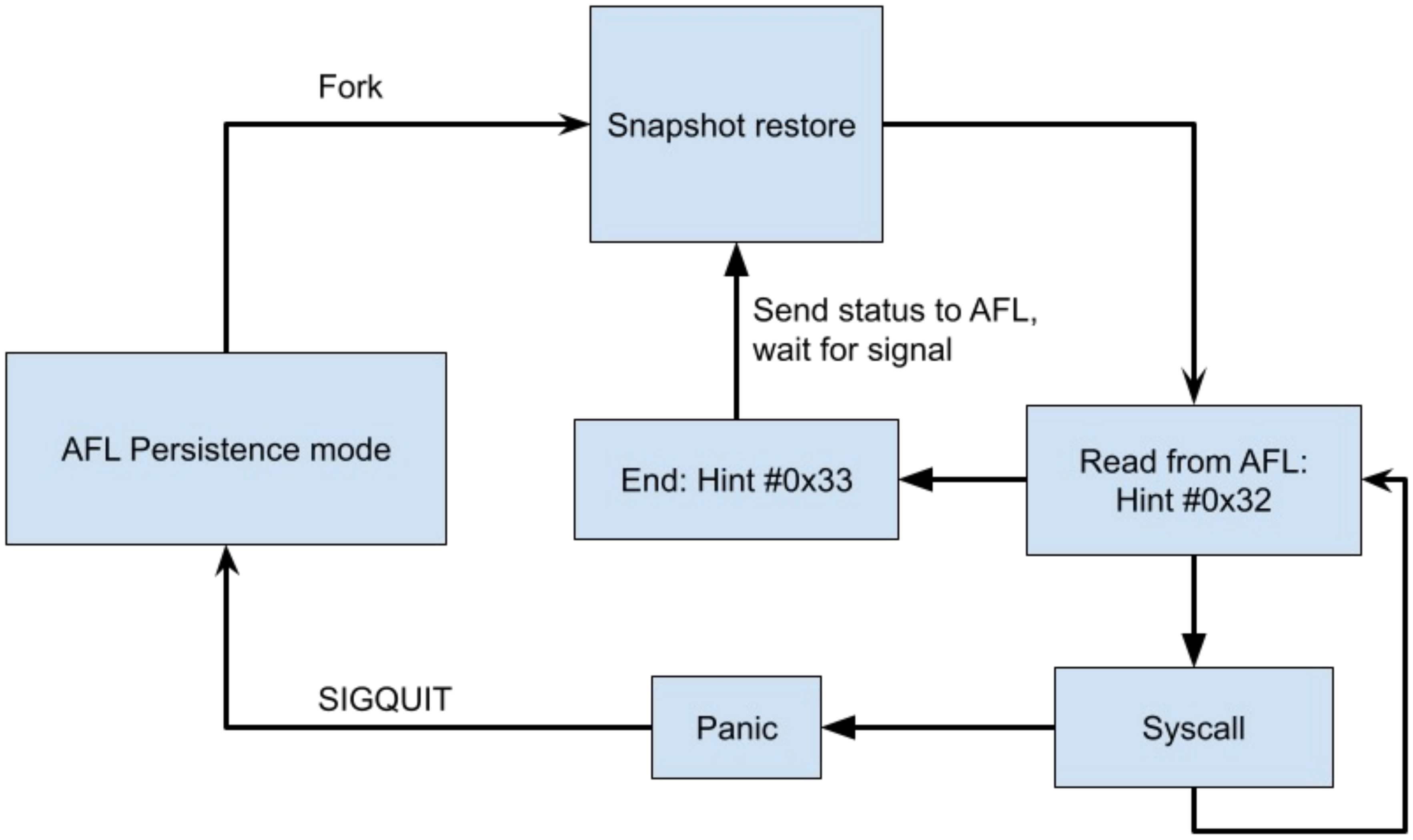
USB fuzzing

```
               american fuzzy lop 2.57b (qemu-system-aarch64)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 12 hrs, 32 min, 24 sec     │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 10 min, 52 sec      │  total paths : 276   │
│ last uniq crash : none seen yet                      │ uniq crashes : 0     │
│  last uniq hang : 0 days, 4 hrs, 25 min, 7 sec       │   uniq hangs : 26    │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0 (0.00%)         │    map density : 2.27% / 3.45%       │
│ paths timed out : 0 (0.00%)         │ count coverage : 3.62 bits/tuple     │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : bitflip 4/1           │ favored paths : 1 (0.36%)            │
│ stage execs : 5802/10.6k (54.75%)   │  new edges on : 85 (30.80%)          │
│ total execs : 30.1k                 │ total crashes : 0 (0 unique)         │
│  exec speed : 0.32/sec (zzzz...)    │  total tmouts : 253 (26 unique)      │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 241/10.6k, 30/10.6k, 0/0              │    levels : 2          │
│  byte flips : 0/0, 0/0, 0/0                         │   pending : 276        │
│ arithmetics : 0/0, 0/0, 0/0                         │  pend fav : 1          │
│  known ints : 0/0, 0/0, 0/0                         │ own finds : 275        │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 0/0, 0/0                              │ stability : 99.34%     │
│        trim : 0.00%/649, n/a                        ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000:  5%]
```

Using TruEmu for greybox fuzzing

Syscall fuzzing



Using TruEmu for greybox fuzzing

Syscall fuzzing

```
                american fuzzy lop 2.57b (qemu-system-aarch64)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 1 days, 22 hrs, 12 min, 15 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 20 min, 46 sec       │  total paths : 250   │
│ last uniq crash : none seen yet                       │ uniq crashes : 0     │
│  last uniq hang : 0 days, 4 hrs, 50 min, 25 sec       │   uniq hangs : 86    │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 39 (15.60%)       │    map density : 2.91% / 11.21%       │
│ paths timed out : 10 (4.00%)        │ count coverage : 1.91 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : arith 8/8             │ favored paths : 111 (44.40%)          │
│ stage execs : 1446/5727 (25.25%)    │  new edges on : 146 (58.40%)          │
│ total execs : 211k                  │ total crashes : 0 (0 unique)          │
│  exec speed : 2.79/sec (zzzz...)    │  total tmouts : 18.9k (86 unique)     │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 84/9544, 19/9531, 18/9505             │    levels : 3          │
│  byte flips : 5/1193, 3/1180, 2/1154                │   pending : 238        │
│ arithmetics : 66/61.9k, 7/67.2k, 0/8165             │  pend fav : 102        │
│  known ints : 2/416, 2/1725, 2/3205                 │ own finds : 249        │
│  dictionary : 0/0, 0/0, 2/599                       │  imported : n/a        │
│       havoc : 36/3251, 0/0                          │ stability : 94.80%     │
│        trim : 3.40%/436, 0.00%                      ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000:  5%]
```

Using TruEmu for greybox fuzzing

Current challenges

- Problem 1: Timer interrupts interfere with coverage result
  - Partial Solution: Mask all interrupts
  - However, our thread is the only one running, so only simple bugs can be found
- Problem 2: Apple does not provide KASAN builds for iOS
  - Potential solution: Hooks allocator’s functions?

TruEmu’s future and roadmap

TruEmu's future and roadmap

Future features

- Framebuffer
- Touch screen
- Working GUI
- SEP
- GPU?
- Fuzzer

TruEmu's future and roadmap

We need you!

- Our code is open-sourced at:
  - http://github.com/TrungNguyen1909/qemu-t8030
- Aid our reverse engineering process through direct/indirect ways
- Contribute to our repo
- Support Linux on ARM Macs efforts

Shoutout

Projects that were helpful for us

- Asahi Linux – Linux on Apple Silicon: https://asahilinux.org
- Corellium – Linux Sandcastle, Linux M1 (abandoned): http://github.com/corellium
- Aleph Security – xnu-qemu-arm64 (abandoned):
  - http://github.com/alephsecurity/xnu-qemu-arm64
- National Science Foundation (NSF) under Award Number CNS-2145744

Takeaways

- iOS full emulation is hard, but it is possible!
- iOS devices' hardware internals and their emulation in a QEMU-based system.
- How TruEMU can be used to enable multiple security applications
- We hope to lower the entry barrier to iOS security research!