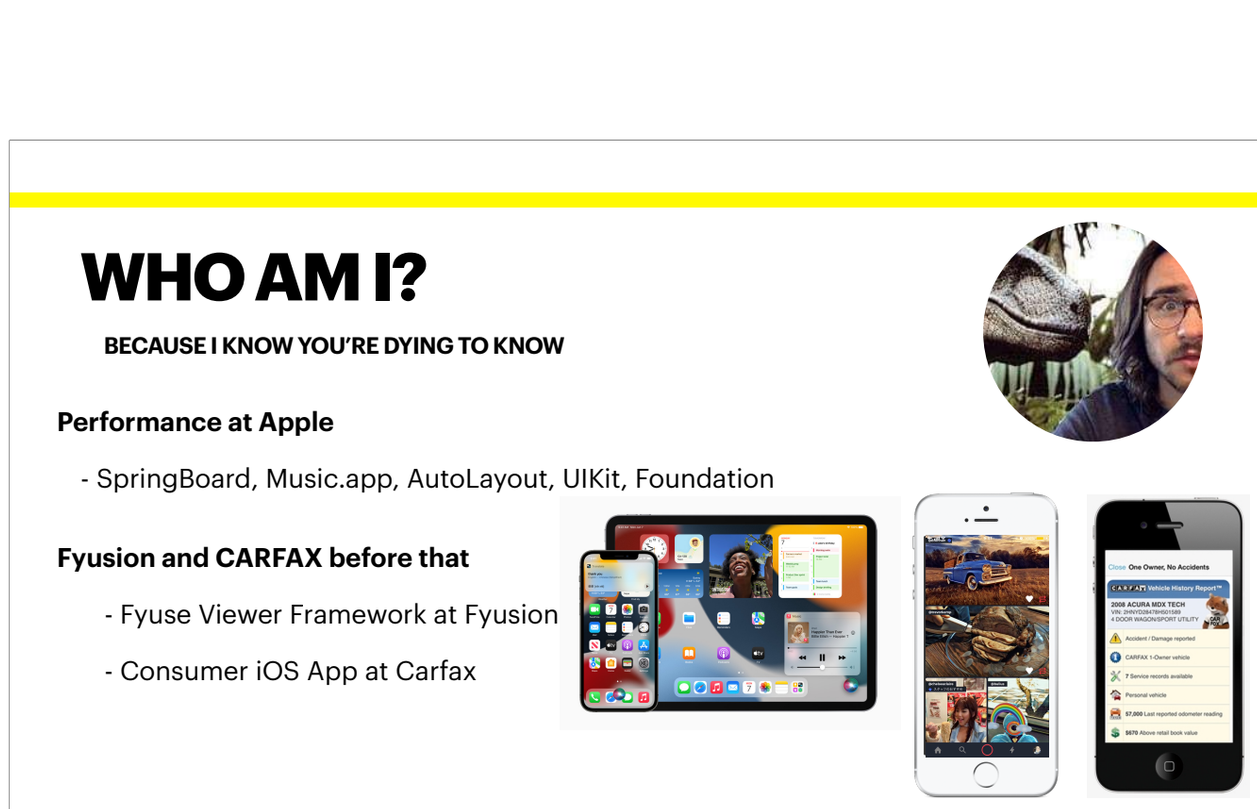

REMEMBER ME
AN ADVENTURE IN MEMORY DEBUGGING



The really convenient thing is, any time I don't know something, I can just pretend its some big secret internal 😉

Most importantly, I focused on memory debugging for my first two years at Apple.

WHAT WILL WE LEARN TODAY

NO SURPRISES

- How **memory works** on Apple's devices
- Some common **bugs** you may encounter
- The **tools** you have at your disposal to squash these bugs

But first...

A quick **history** lesson

WHY SHOULD YOU CARE?

BECAUSE YOU SHOULD

- Memory usage can impact **everything**, including your CPU performance

- Abusing memory is a frequent **cause of crashes**

- A bad memory citizen not only hurts itself, but **all other processes** on the system

- Swift can push us away from a some of these bugs, *but* that doesn't mean we won't encounter them via **legacy code** and **frameworks** we use!

HISTORY

MEMORY THROUGH THE AGES

a precious resource

Apple II (1977)

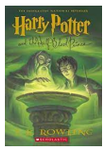
4KB

Macintosh (1984)

128KB

Macintosh Plus (1986)

1 MB



iMac G3 (1998)

32MB

@LUKEPARHAM

https://www.computerhistory.org/timeline/1981/

https://arstechnica.com/gaming/2020/04/war-stories-how-prince-of-persia-slew-the-apple-iis-memory-limitations/

MEMORY THROUGH THE AGES

a precious resource (especially in the mobile era)

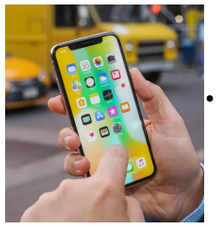
iPhone (2008)

128MB

iPhone 5s (2013)

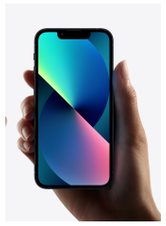
1GB

iPhone X (2017)



3GB

iPhone 13 Pro (2021)



6GB

@LUKEPARHAM

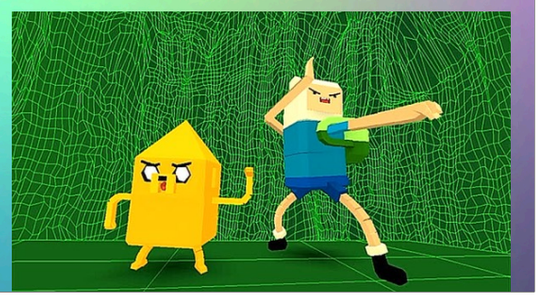
A boon to hand models everywhere

TAKEAWAYS

- Memory capacity is always increasing, but so are the demands of our apps
- RAM is **expensive**, and code can allocate it faster than we can fabricate it
    - video **games**
    - streaming **video**
    - machine learning **models**
    - tons of **text** and **images**

THE SYSTEM

VIRTUAL MEMORY



VIRTUAL MEMORY BASICS

Everything that needs to be acted upon will be loaded into memory

- This means the **__TEXT** for the code you've written
- As well as all data your program **creates** or **loads** from the outside

  ie) mmap, malloc, vm_allocate

VIRTUAL MEMORY BASICS

Virtual memory is an abstraction that allows programs to overcome the physical memory limits of a device

- Each program is given its own **virtual address space**
  - This address space is both **contiguous** and much **larger** than physical memory
- Virtual addresses are translated to physical addresses automatically by **hardware**
- Virtual addresses referencing data not in physical memory result in a **fault** where the data is loaded into physical memory before the program can continue

These faults are known as “**page faults**” and there is some overhead associated with this work. This is one reason memory can affect CPU performance.

VIRTUAL MEMORY BASICS

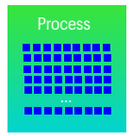
Memory is split up into “pages” of equal sizes by the VM

- Depending on the device, this is either **4KB** or **16KB** per page
  - **malloc** hides this from you and gives you memory in the chunks you request
  - Its goal is to make sure these pages are tightly packed without leaving too many holes (aka avoiding **fragmentation**)

Current devices have 16K virtual memory mapped onto 16K physical pages

CLEAN AND DIRTY MEMORY

Memory is either Clean or Dirty



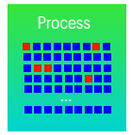
- This is tracked at the **page level** of granularity
- **Clean** pages ***can*** be removed from physical memory and brought back later
- **Dirty** pages ***cannot*** be re-created and therefore contribute to system pressure

*** Pages go from being “clean” to “dirty” after being written to!**

For example, __TEXT is usually clean because it isn’t modified in memory and can be reloaded later when necessary.
But, what do you think happens if the functions you call are spread throughout your binary in an inefficient way?

CLEAN AND DIRTY MEMORY

Memory is either Clean or Dirty

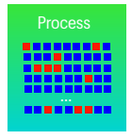


- This is tracked at the **page level** of granularity
- **Clean** pages ***can*** be removed from physical memory and brought back later
- **Dirty** pages ***cannot*** be re-created and therefore contribute to system pressure

*** Pages go from being “clean” to “dirty” after being written to!**

As pages are written to, they change to being “dirty” pages.

CLEAN AND DIRTY MEMORY



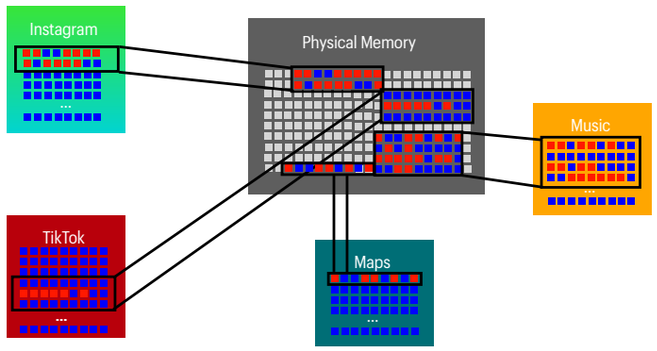
Memory is either Clean or Dirty

- This is tracked at the **page level** of granularity
- **Clean** pages ***can*** be removed from physical memory and brought back later
- **Dirty** pages ***cannot*** be re-created and therefore contribute to system pressure

*** Pages go from being “clean” to “dirty” after being written to!**

VIRTUAL MEMORY BASICS

This abstraction means that physical memory is actually a set of windows into the more “real” virtual footprint of the processes running on the system



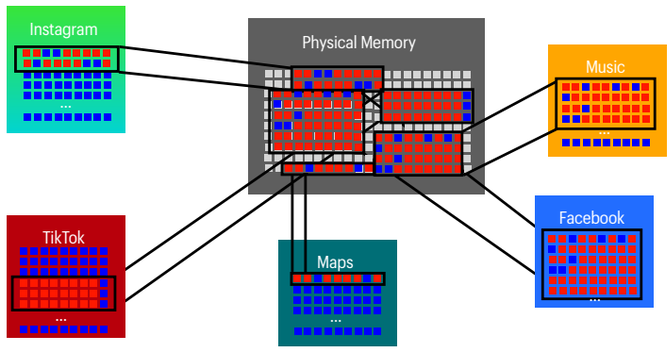
@LUKEPARHAM

The red squares are dirty pages, and the blue are clean.

Dirty memory is memory that cannot be removed from physical memory and thus, contributes to memory pressure.

VIRTUAL MEMORY BASICS

But what about when physical memory starts getting full of dirty pages?



@LUKEPARHAM

As memory pressure increases, page faults become more frequent, as well as churn from the system desperately trying to free up pages.

http://newosxbook.com/articles/MemoryPressure.html

JETSAM - LICENSE TO KILL

iOS and watchOS don't use paging to free up memory

- instead, a system called **Jetsam** constantly monitors memory pressure and just straight up **kills programs** to free up memory
    - this happens in **priority order** starting with idle processes and moving up until pressure is at an acceptable level
- memory pressure can happen at any time, and **may not** even be your app's fault!
- the system also sends out **notifications** to let your app know there is memory pressure

JETSAM - LICENSE TO KILL

PUNISHABLE BY DEATH

Jetsam maintains a list of processes and their priority and kills them to reclaim memory

- When a process is killed, Jetsam will tell you the **reason**
  - **per-process-limit**: the process crossed its memory limit
  - **vm-pageshortage**: the system was under pressure and the foreground app needed memory
  - **vnode-limit**: the system needed more file descriptors
  - **fc-thrashing**: the file system cache is experiencing “thrashing”

@LUKEPARHAM

https://opensource.apple.com/source/xnu/xnu-6153.81.5/bsd/kern/kern_memorystatus.c.auto.html

https://developer.apple.com/documentation/xcode/identifying-high-memory-use-with-jetsam-event-reports

JETSAM - LICENSE TO KILL

```c
/* For logging clarity */
static const char *memorystatus_kill_cause_name[] = {
        "",                                     /* kMemorystatusInvalid                         */
        "jettisoned",                           /* kMemorystatusKilled                          */
        "highwater",                            /* kMemorystatusKilledHiwat                     */
        "vnode-limit",                          /* kMemorystatusKilledVnodes                    */
        "vm-pageshortage",                      /* kMemorystatusKilledVMPageShortage            */
        "proc-thrashing",                       /* kMemorystatusKilledProcThrashing             */
        "fc-thrashing",                         /* kMemorystatusKilledFCThrashing               */
        "per-process-limit",                    /* kMemorystatusKilledPerProcessLimit           */
        "disk-space-shortage",                  /* kMemorystatusKilledDiskSpaceShortage         */
        "idle-exit",                            /* kMemorystatusKilledIdleExit                  */
        "zone-map-exhaustion",                  /* kMemorystatusKilledZoneMapExhaustion         */
        "vm-compressor-thrashing",              /* kMemorystatusKilledVMCompressorThrashing     */
        "vm-compressor-space-shortage",         /* kMemorystatusKilledVMCompressorSpaceShortage */
};

static const char *
memorystatus_priority_band_name(int32_t priority)
{
        switch (priority) {
        case JETSAM_PRIORITY_FOREGROUND:
                return "FOREGROUND";
        case JETSAM_PRIORITY_AUDIO_AND_ACCESSORY:
                return "AUDIO_AND_ACCESSORY";
        case JETSAM_PRIORITY_CONDUCTOR:
                return "CONDUCTOR";
        case JETSAM_PRIORITY_DRIVER_APPLE:
                return "DRIVER_APPLE";
        case JETSAM_PRIORITY_HOME:
                return "HOME";
        case JETSAM_PRIORITY_EXECUTIVE:
                return "EXECUTIVE";
        case JETSAM_PRIORITY_IMPORTANT:
                return "IMPORTANT";
        case JETSAM_PRIORITY_CRITICAL:
                return "CRITICAL";
        }

        return "?";
}
```

@LUKEPARHAM

https://opensource.apple.com/source/xnu/xnu-6153.81.5/bsd/kern/kern_memorystatus.c.auto.html

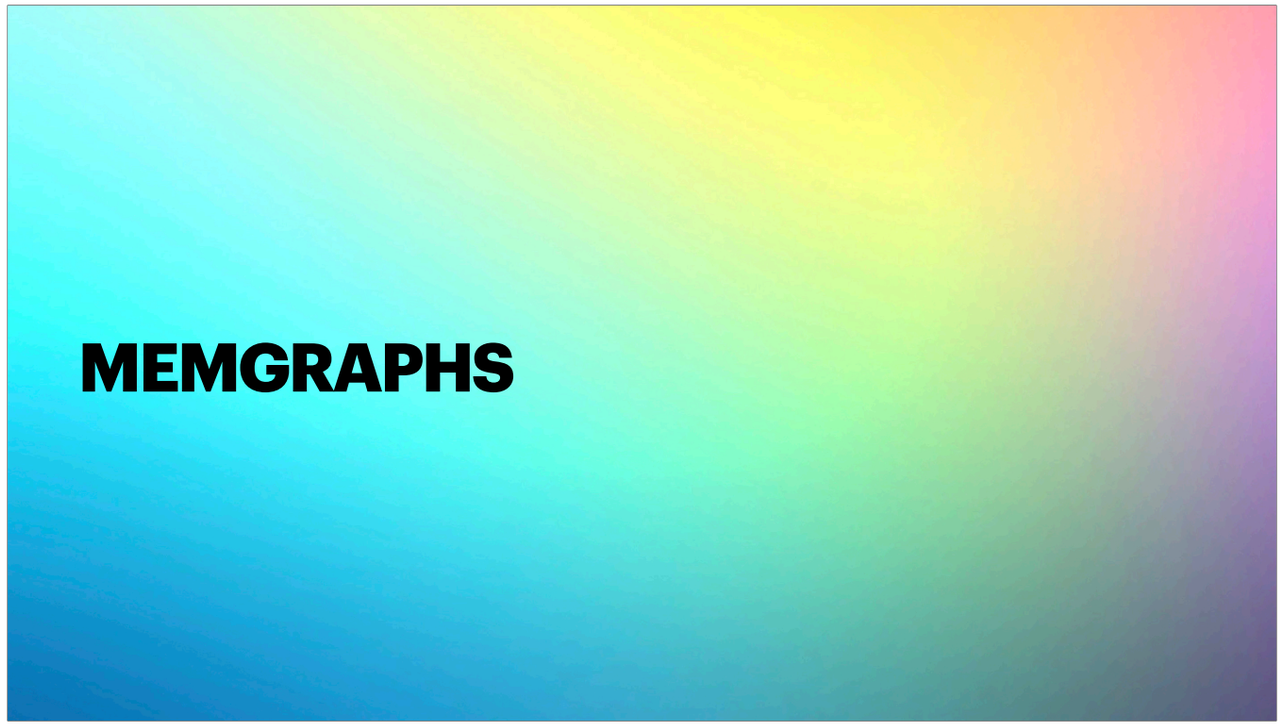
ONLY YOU CAN PREVENT LOW MEMORY CONDITIONS

Every developer should make it their goal to have an app that behaves correctly

- Your app shouldn’t be getting killed for crossing it’s memory limit

- Let’s see what some common issues are

- As well as what tools we have at our disposal to fix them



Next we’re gonna talk about memgraphs. The currency of analyzing an app’s memory.

MEMGRAPHS

A SNAPSHOT OF A PROCESS'S FOOTPRINT

memgraphs are a convenient file format containing a snapshot of your app's memory

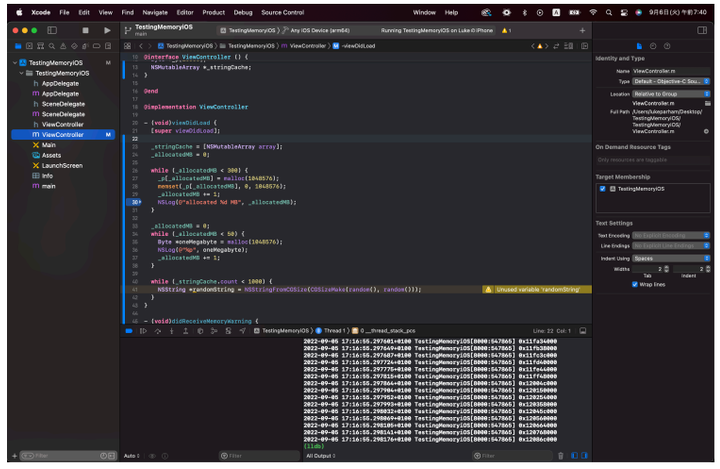
- can be opened in **Xcode** and explored graphically
- can also be explored with a number of **command line tools**
  - **footprint**, **vmmap**, **leaks**, **heap**, **malloc_history**
  - analyzing output with tools like **awk** and **grep** can be a powerful debugging workflow
  - collect them via **Xcode** or the **leaks** command on command line

@LUKEPARHAM

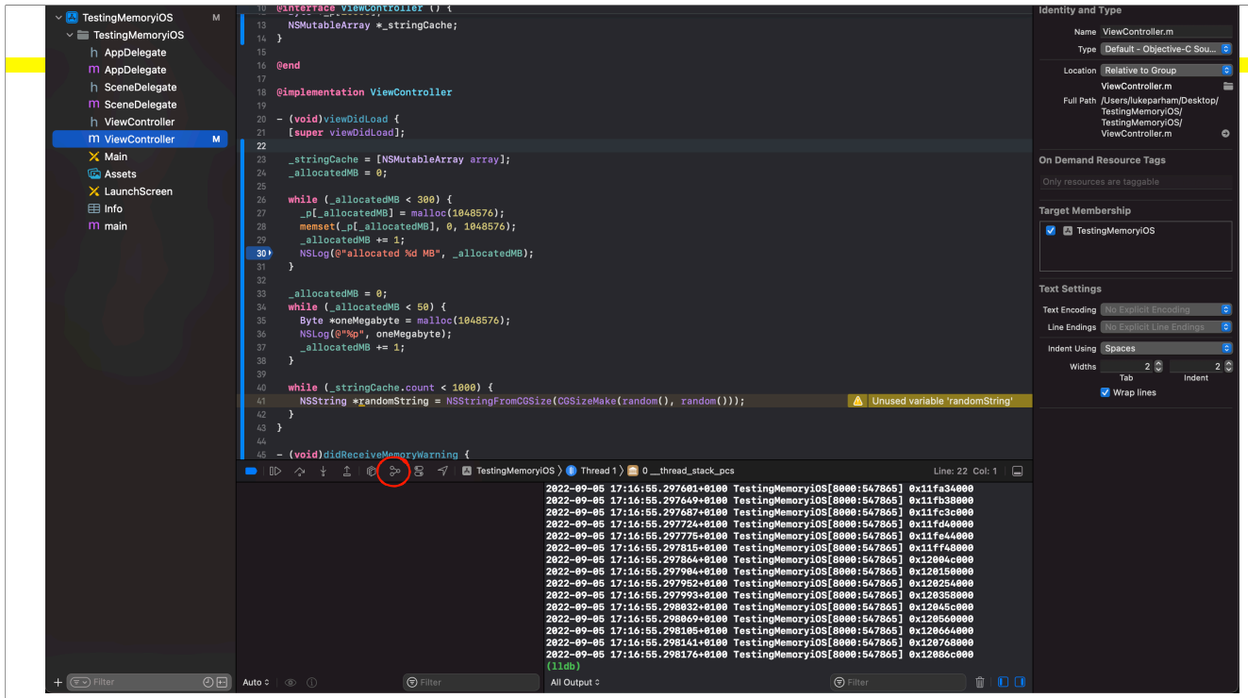
They're super useful because they're very easy to use as a historical reference for how your program looked at any given time.

You could take one after launch and then after automation every time you submit your app and have a good idea of what your footprint looks like over time and then be able to easily compare against what used to be there when you see that your memory has increased.

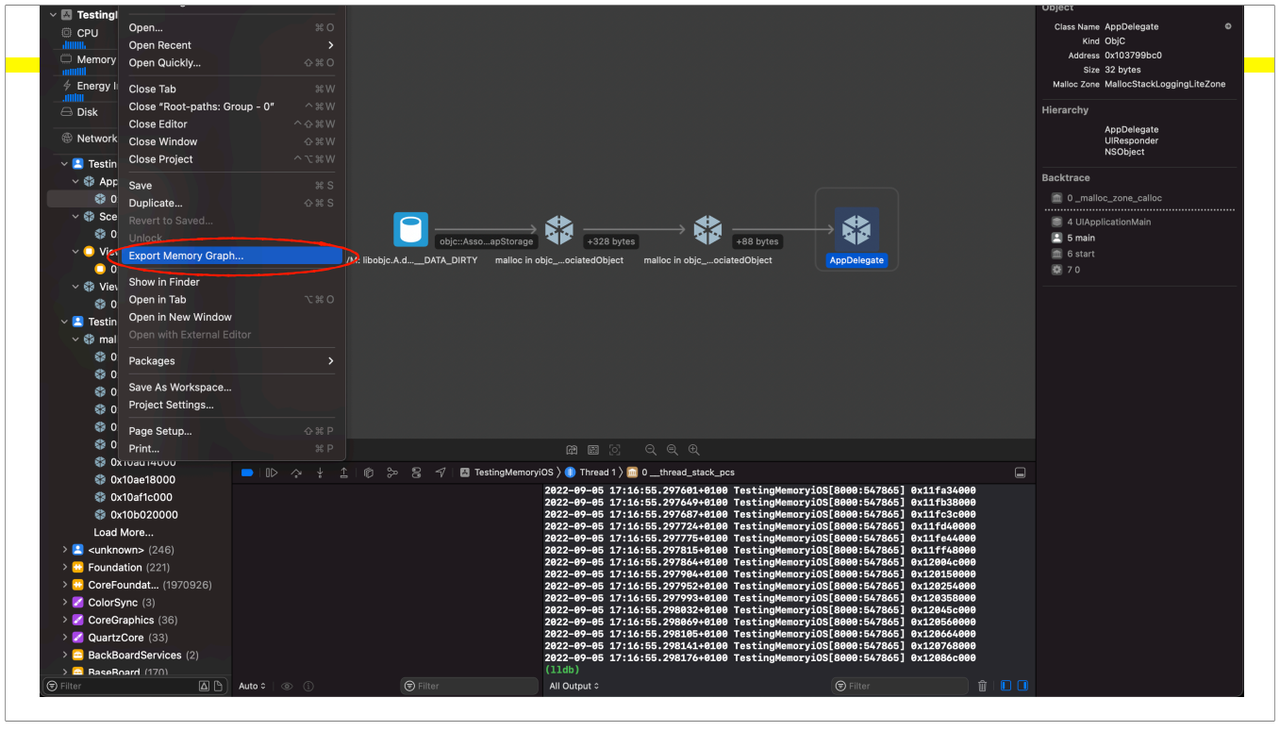
MEMGRAPHS



leaks --outputGraph=BigMalloc.memgraph MyProcess




```
leaks --outputGraph=BigMalloc.memgraph  MyProcess
```




```
leaks --outputGraph=BigMalloc.memgraph  MyProcess
```

MEMGRAPHS ON MAC

```
~/Desktop lukeparham@LukenoMacBook-Pro
[> leaks --outputGraph=FinderMemgraph.memgraph Finder
Output graph successfully written to 'FinderMemgraph.memgraph' [3.15 MB]
```

leaks --outputGraph=BigMalloc.memgraph MyProcess

THE BUGS

Everybody knows what a leak is, but we’re still gonna talk about it because not everybody knows what a leak is.

TURNING OFF THE WATERWORKS

A "*leak*" is when a program loses track of a reference, and forgets to free that memory

- The simplest example is calling **malloc()** and forgetting to call **free()**
- Next, a leak can happen in ObjC/Swift via a **retain cycle**
  - Two objects with **strong references** to each other
  - Commonly happens when **owned blocks** hold strong references to **self**
- Leaks can also happen at **boundaries** between ARC and MRR code
- Usually need **Malloc Stack Logging** to figure out where a leak is coming from

MALLOC STACK LOGGING

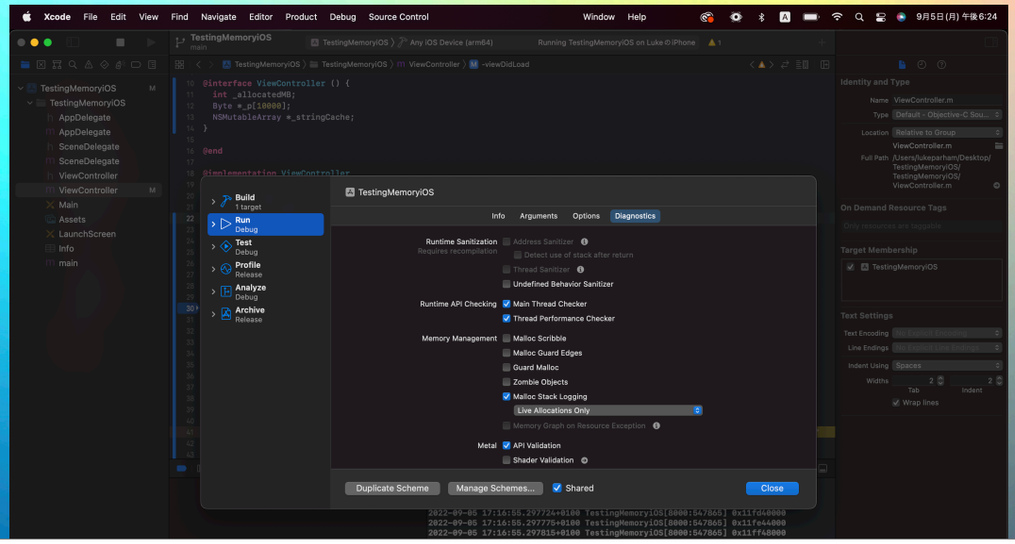
- Sometimes you need to know exactly where the allocation of an object happened.

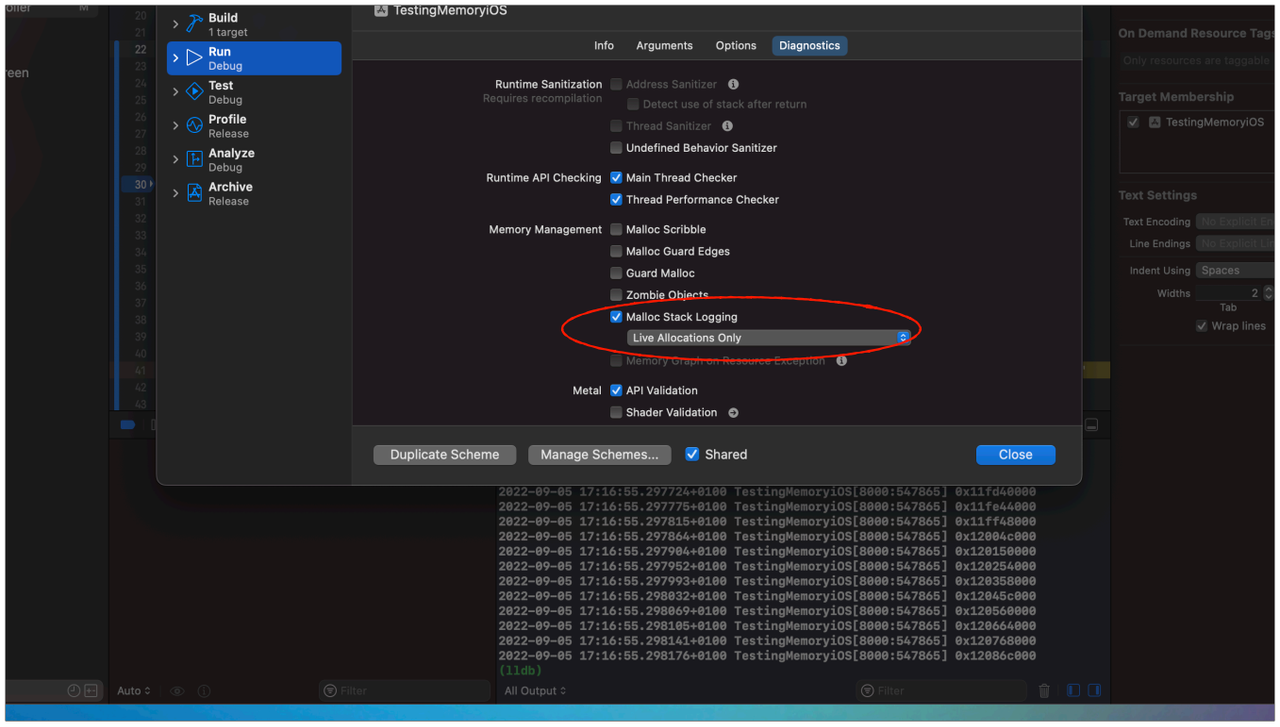
- Enabling malloc stack logging means that for any given allocation, you can see what the **call-stack** of your program was when it happened.

- There are different modes

  - **Lite**: allows inspection of live objects
  - **Full**: allows inspection of all objects in the program's history (much more overhead)

TURNING ON MALLOC STACK LOGGING




Build
1 target
Run
Debug
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
TestingMemoryiOS
Info
Arguments
Options
Diagnostics
Runtime Sanitization
Requires recompilation
Address Sanitizer
Detect use of stack after return
Thread Sanitizer
Undefined Behavior Sanitizer
Runtime API Checking
Main Thread Checker
Thread Performance Checker
Memory Management
Malloc Scribble
Malloc Guard Edges
Guard Malloc
Zombie Objects
Malloc Stack Logging
Live Allocations Only
Memory Graph on Resource Exception
Metal
API Validation
Shader Validation
Duplicate Scheme
Manage Schemes...
Shared
Close
2022-09-05 17:16:55.297724+0100 TestingMemoryiOS[8000:547865] 0x11fd40000
2022-09-05 17:16:55.297775+0100 TestingMemoryiOS[8000:547865] 0x11fe44000
2022-09-05 17:16:55.297815+0100 TestingMemoryiOS[8000:547865] 0x11ff48000
2022-09-05 17:16:55.297864+0100 TestingMemoryiOS[8000:547865] 0x12004c000
2022-09-05 17:16:55.297904+0100 TestingMemoryiOS[8000:547865] 0x120150000
2022-09-05 17:16:55.297952+0100 TestingMemoryiOS[8000:547865] 0x120254000
2022-09-05 17:16:55.297993+0100 TestingMemoryiOS[8000:547865] 0x120358000
2022-09-05 17:16:55.298032+0100 TestingMemoryiOS[8000:547865] 0x12045c000
2022-09-05 17:16:55.298069+0100 TestingMemoryiOS[8000:547865] 0x120560000
2022-09-05 17:16:55.298105+0100 TestingMemoryiOS[8000:547865] 0x120664000
2022-09-05 17:16:55.298141+0100 TestingMemoryiOS[8000:547865] 0x120768000
2022-09-05 17:16:55.298176+0100 TestingMemoryiOS[8000:547865] 0x12086c000
(lldb)
Auto
Filter
All Output
Filter
On Demand Resource Tags
Only resources are taggable
Target Membership
TestingMemoryiOS
Text Settings
Text Encoding
Line Endings
Indent Using
Spaces
Widths
2
Tab
Wrap lines

LEAKS

```
leaks BigMallocLeaksMSL.memgraph
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [7750]
Path:            /private/var/containers/Bundle/Application/32F4CB36-4F18-4E5B-998E-8658AF245C13/TestingMemoryiOS.app/TestingMemoryiOS
Load Address:    0x104f74000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7749]

Date/Time:       2022-09-05 16:44:55.901 +0100
Launch Time:     2022-09-05 16:44:46.912 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         321.5M
Physical footprint (peak):  321.7M
====

leaks Report Version: 4.0, multi-line stacks
Process 7750: 22341 nodes malloced for 366419 KB
Process 7750: 50 leaks for 53248000 total leaked bytes.

STACK OF 50 INSTANCES OF 'ROOT LEAK: <malloc in -[ViewController viewDidLoad]>':
32  dyld                                0x104fc5ce4 start + 520
31  TestingMemoryiOS                    0x104f7a14c main + 120
30  UIKitCore                           0x1b8d57d90 UIApplicationMain + 364
29  UIKitCore                           0x1b8fd66a8 -[UIApplication _run] + 1100
28  com.apple.GraphicsServices          0x1d279a374 GSEventRunModal + 144
27  com.apple.CoreFoundation            0x1b6666bc8 CFRunLoopRunSpecific + 600
26  com.apple.CoreFoundation            0x1b665385c __CFRunLoopRun + 828
25  com.apple.CoreFoundation            0x1b664d6f8 __CFRunLoopDoSources0 + 368
24  com.apple.CoreFoundation            0x1b67141a8 __CFRunLoopDoSource0 + 208
23  com.apple.CoreFoundation            0x1b6703414 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 28
22  com.apple.FrontBoardServices        0x1c88d1708 -[FBSSerialQueue _performNextFromRunLoopSource] + 28
21  com.apple.FrontBoardServices        0x1c88cd040 -[FBSSerialQueue _targetQueue_performNextIfPossible] + 220
20  com.apple.FrontBoardServices        0x1c88cdc70 __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__ + 48
19  libdispatch.dylib                   0x1052ae264 _dispatch_block_invoke_direct + 368
18  libdispatch.dylib                   0x1052aa7c0 _dispatch_client_callout + 20
17  com.apple.FrontBoardServices        0x1c88cd958 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke + 372
16  com.apple.FrontBoardServices        0x1c88cc388 -[FBSWorkspace _calloutQueue_executeCalloutFromSource:withBlock:] + 240
15  com.apple.FrontBoardServices        0x1c8911090 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke.215 + 128
14  com.apple.FrontBoardServices        0x1c886eb294 -[FBSScene _calloutQueue_agent_didCreateWithTransitionContext:completion:] + 440
13  UIKitCore                           0x1b8d5a588 -[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:] + 388
12  UIKitCore                           0x1b8e19a48 -[UIApplication workspace:didCreateScene:withTransitionContext:completion:] + 344
11  UIKitCore                           0x1b9116c64 -[UIApplication _connectUISceneFromFBSScene:transitionContext:] + 1256
10  UIKitCore                           0x1b8e17474 +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:] + 1576
9   UIKitCore                           0x1b8b14e7c -[UIWindowScene _makeKeyAndVisibleIfNeeded] + 204
8   UIKitCore                           0x1b8d8c66c -[UIWindow _mainQueue_makeKeyAndVisible] + 56
7   UIKitCore                           0x1b8d71f08 -[UIWindow _setHidden:forced:] + 280
6   UIKitCore                           0x1b8cdb5ac -[UIWindow _updateLayerOrderingAndSetLayerHidden:actionBlock:] + 232
5   UIKitCore                           0x1b8d85bf4 -[UIWindow addRootViewControllerViewIfPossible] + 180
4   UIKitCore                           0x1b8c439e4 -[UIViewController view] + 32
3   UIKitCore                           0x1b8c70984 -[UIViewController loadViewIfRequired] + 1048
2   UIKitCore                           0x1b8c6da34 -[UIViewController _sendViewDidLoadWithAppearanceProxyObjectTaggingEnabled] + 108
1   TestingMemoryiOS                    0x104f79e78 -[ViewController viewDidLoad] + 332
0   libsystem_malloc.dylib              0x1c8975024 _malloc_zone_malloc + 156
====
    50 (50.8M) << TOTAL >>
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f7b0000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f0b4000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f9b8000> [1064960]
```

The leaks tool gives you a list of all objects recognized as leaked memory by the system.

```
> leaks BigMallocLeaksMSL.memgraph
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [7750]
Path:            /private/var/containers/Bundle/Application/32F4CB36-4F1B-4E5B-998E-B658AF245C13/TestingMemoryiOS.app/Te
Load Address:    0x104f74000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7749]

Date/Time:       2022-09-05 16:44:55.901 +0100
Launch Time:     2022-09-05 16:44:46.912 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         321.5M
Physical footprint (peak):  321.7M
----

leaks Report Version: 4.0, multi-line stacks
Process 7750: 22341 nodes malloced for 366419 KB
Process 7750: 50 leaks for 53248000 total leaked bytes.

STACK OF 50 INSTANCES OF 'ROOT LEAK: <malloc in -[ViewController viewDidLoad]>':
32  dyld                                0x104fc5ce4 start + 520
31  TestingMemoryiOS                    0x104f7a14c main + 120
30  UIKitCore                           0x1b8d57d90 UIApplicationMain + 364
29  UIKitCore                           0x1b8fd6648 -[UIApplication _run] + 1100
28  com.apple.GraphicsServices          0x1d279a374 GSEventRunModal + 164
```

```
leaks Report Version: 4.0, multi-line stacks
Process 7750: 22341 nodes malloced for 366419 KB
Process 7750: 50 leaks for 53248000 total leaked bytes.

STACK OF 50 INSTANCES OF 'ROOT LEAK: <malloc in -[ViewController viewDidLoad]>':
32  dyld                                0x104fc5ce4 start + 520
31  TestingMemoryiOS                    0x104f7a14c main + 120
30  UIKitCore                           0x1b8d57d90 UIApplicationMain + 364
29  UIKitCore                           0x1b8fd6648 -[UIApplication _run] + 1100
28  com.apple.GraphicsServices          0x1d279a374 GSEventRunModal + 164
27  com.apple.CoreFoundation            0x1b6666bc8 CFRunLoopRunSpecific + 600
26  com.apple.CoreFoundation            0x1b665305c __CFRunLoopRun + 828
25  com.apple.CoreFoundation            0x1b664d6f8 __CFRunLoopDoSources0 + 368
24  com.apple.CoreFoundation            0x1b67141a0 __CFRunLoopDoSource0 + 208
23  com.apple.CoreFoundation            0x1b6703414 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 28
22  com.apple.FrontBoardServices        0x1c88d1700 -[FBSSerialQueue _performNextFromRunLoopSource] + 28
21  com.apple.FrontBoardServices        0x1c88cd040 -[FBSSerialQueue _targetQueue_performNextIfPossible] + 220
20  com.apple.FrontBoardServices        0x1c88cdc70 __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__ + 48
19  libdispatch.dylib                   0x1052ae264 _dispatch_block_invoke_direct + 368
18  libdispatch.dylib                   0x1052aa7c0 _dispatch_client_callout + 20
17  com.apple.FrontBoardServices        0x1c88cd958 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_blo
16  com.apple.FrontBoardServices        0x1c88cc308 -[FBSWorkspace _calloutQueue_executeCalloutFromSource:withBlock:] + 240
15  com.apple.FrontBoardServices        0x1c8911090 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_blo
14  com.apple.FrontBoardServices        0x1c88eb294 -[FBSScene _callOutQueue_agent_didCreateWithTransitionContext:completion:] + 440
13  UIKitCore                           0x1b8d5a508 -[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:] + 388
12  UIKitCore                           0x1b8e19a48 -[UIApplication workspace:didCreateScene:withTransitionContext:completion:] + 344
11  UIKitCore                           0x1b9116c64 -[UIApplication _connectUISceneFromFBSScene:transitionContext:] + 1256
10  UIKitCore                           0x1b8e17474 +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:] + 1576
9   UIKitCore                           0x1b8b14e7c -[UIWindowScene _makeKeyAndVisibleIfNeeded] + 204
8   UIKitCore                           0x1b8d8c66c -[UIWindow _mainQueue_makeKeyAndVisible] + 56
7   UIKitCore                           0x1b8d71fd0 -[UIWindow _setHidden:forced:] + 280
6   UIKitCore                           0x1b8cdb5ac -[UIWindow _updateLayerOrderingAndSetLayerHidden:actionBlock:] + 232
5   UIKitCore                           0x1b8d85bf4 -[UIWindow addRootViewControllerViewIfPossible] + 180
4   UIKitCore                           0x1b8c439e4 -[UIViewController view] + 32
3   UIKitCore                           0x1b8c70984 -[UIViewController loadViewIfRequired] + 1048
2   UIKitCore                           0x1b8c6de34 -[UIViewController _sendViewDidLoadWithAppearanceProxyObjectTaggingEnabled] + 108
1   TestingMemoryiOS                    0x104f79e78 -[ViewController viewDidLoad] + 332
0   libsystem_malloc.dylib              0x1c8975024 _malloc_zone_malloc + 156
====
   50 (50.8M) << TOTAL >>
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f7b0000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f8b4000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f9b8000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fabc000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fbc0000> [1064960]
     1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fcc4000> [1064960]
```

```
leaks Report Version: 4.0, multi-line stacks
Process 7750: 22341 nodes malloced for 366419 KB
Process 7750: 50 leaks for 53248000 total leaked bytes.

STACK OF 50 INSTANCES OF 'ROOT LEAK: <malloc in -[ViewController viewDidLoad]>':
32  dyld                                0x104fc5ce4 start + 520
31  TestingMemoryiOS                    0x104f7a14c main + 120
30  UIKitCore                           0x1b8d57d90 UIApplicationMain + 364
29  UIKitCore                           0x1b8fd6648 -[UIApplication _run] + 1100
28  com.apple.GraphicsServices          0x1d279a374 GSEventRunModal + 164
27  com.apple.CoreFoundation            0x1b6666bc8 CFRunLoopRunSpecific + 600
26  com.apple.CoreFoundation            0x1b665305c __CFRunLoopRun + 828
25  com.apple.CoreFoundation            0x1b664d6f8 __CFRunLoopDoSources0 + 368
24  com.apple.CoreFoundation            0x1b67141a0 __CFRunLoopDoSource0 + 208
23  com.apple.CoreFoundation            0x1b6703414 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 28
22  com.apple.FrontBoardServices        0x1c88d1700 -[FBSSerialQueue _performNextFromRunLoopSource] + 28
21  com.apple.FrontBoardServices        0x1c88cd040 -[FBSSerialQueue _targetQueue_performNextIfPossible] + 220
20  com.apple.FrontBoardServices        0x1c88cdc70 __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__ + 48
19  libdispatch.dylib                   0x1052ae264 _dispatch_block_invoke_direct + 368
18  libdispatch.dylib                   0x1052aa7c0 _dispatch_client_callout + 20
17  com.apple.FrontBoardServices        0x1c88cd958 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_blo
16  com.apple.FrontBoardServices        0x1c88cc308 -[FBSWorkspace _calloutQueue_executeCalloutFromSource:withBlock:] + 240
15  com.apple.FrontBoardServices        0x1c8911090 __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_blo
14  com.apple.FrontBoardServices        0x1c88eb294 -[FBSScene _callOutQueue_agent_didCreateWithTransitionContext:completion:] + 440
13  UIKitCore                           0x1b8d5a508 -[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:] + 388
12  UIKitCore                           0x1b8e19a48 -[UIApplication workspace:didCreateScene:withTransitionContext:completion:] + 344
11  UIKitCore                           0x1b9116c64 -[UIApplication _connectUISceneFromFBSScene:transitionContext:] + 1256
10  UIKitCore                           0x1b8e17474 +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:] + 1576
9   UIKitCore                           0x1b8b14e7c -[UIWindowScene _makeKeyAndVisibleIfNeeded] + 204
8   UIKitCore                           0x1b8d8c66c -[UIWindow _mainQueue_makeKeyAndVisible] + 56
7   UIKitCore                           0x1b8d71fd0 -[UIWindow _setHidden:forced:] + 280
6   UIKitCore                           0x1b8cdb5ac -[UIWindow _updateLayerOrderingAndSetLayerHidden:actionBlock:] + 232
5   UIKitCore                           0x1b8d85bf4 -[UIWindow addRootViewControllerViewIfPossible] + 180
4   UIKitCore                           0x1b8c439e4 -[UIViewController view] + 32
3   UIKitCore                           0x1b8c70984 -[UIViewController loadViewIfRequired] + 1048
2   UIKitCore                           0x1b8c6de34 -[UIViewController _sendViewDidLoadWithAppearanceProxyObjectTaggingEnabled] + 108
1   TestingMemoryiOS                    0x104f79e78 -[ViewController viewDidLoad] + 332
0   libsystem_malloc.dylib              0x1c8975024 _malloc_zone_malloc + 156
====
    50 (50.8M) << TOTAL >>
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f7b0000> [1064960]
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f8b4000> [1064960]
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11f9b8000> [1064960]
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fabc000> [1064960]
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fbc0000> [1064960]
      1 (1.02M) ROOT LEAK: <malloc in -[ViewController viewDidLoad] 0x11fcc4000> [1064960]
```

MALLOC_HISTORY

```
> malloc_history --callTree BigMallocMSL.memgraph
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [7674]
Path:            /private/var/containers/Bundle/Application/6E568940-1386-45A1-9848-DE8BD8A6502D/TestingMemoryiOS.app/TestingMemoryiOS
Load Address:    0x102208000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7673]

Date/Time:       2022-09-05 16:34:08.621 +0100
Launch Time:     2022-09-05 16:33:58.982 +0100
OS Version:      iPhone OS 15.6 (19F77)
Report Version:  104

Physical footprint:         314.9M
Physical footprint (peak):  321.1M
----

VM region sizes shown as dirty + swapped/compressed – purgableVolatile.

Stack logging was dynamically enabled in target process, after it was launched,
so no backtraces are available for earlier allocations.

Malloc blocks with no stack:    531   total size: 76K
  VM regions with no stack:       4   total size: 64K

Call graph:
    21960 (307M) << TOTAL >>
      4347 (305M) start  (in dyld) + 520  [0x102601ce4]
      + 4339 (305M) main  (in TestingMemoryiOS) + 120  [0x10220e14c]
      +   1142 (305M) UIApplicationMain  (in UIKitCore) + 344  [0x1b8d57d90]
      +   2954 (305M) -[UIApplication _run]  (in UIKitCore) + 1100  [0x1b8fd6648]
      +     2954 (305M) GSEventRunModal  (in GraphicsServices) + 164  [0x1d279a374]
      +       2953 (305M) CFRunLoopRunSpecific  (in CoreFoundation) + 600  [0x1b6666bc8]
      +         2873 (305M) __CFRunLoopRun  (in CoreFoundation) + 828  [0x1b665385c]
      +         2265 (305M) __CFRunLoopDoSources0  (in CoreFoundation) + 368  [0x1b664d6f8]
      +         2265 (305M) __CFRunLoopDoSource0  (in CoreFoundation) + 208  [0x1b67141a0]
      +           2265 (305M) __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__  (in CoreFoundation) + 28  [0x1b6703414]
      +             2229 (305M) -[FBSSerialQueue _performNextFromRunLoopSource]  (in FrontBoardServices) + 28  [0x1c88d1700]
      +               2229 (305M) -[FBSSerialQueue _targetQueue_performNextIfPossible]  (in FrontBoardServices) + 220  [0x1c88cd040]
      +                 2229 (305M) __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__  (in FrontBoardServices) + 48  [0x1c88cdc70]
      +                   2229 (305M) _dispatch_block_invoke_direct  (in libdispatch.dylib) + 368  [0x10247e264]
      +                     2229 (305M) _dispatch_client_callout  (in libdispatch.dylib) + 20  [0x10247a7c0]
      +                       2046 (305M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke  (in FrontBoardServices) + 372  [0x1c88cd958]
      +                         + 2043 (305M) -[FBSWorkspace _calloutQueue_executeCalloutFromSource:withBlock:]  (in FrontBoardServices) + 240  [0x1c88cc308]
      +                         + 2043 (305M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke.215  (in FrontBoardServices) + 128  [0x1c8911090]
      +                         +   2036 (305M) -[FBSScene _calloutQueue_agent_didCreateWithTransitionContext:completion:]  (in FrontBoardServices) + 440  [0x1c88eb294]
      +                         +   : 2036 (305M) -[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:]  (in UIKitCore) + 388  [0x1b8d5a508]
      +                         +   :   1895 (305M) -[UIApplication workspace:didCreateScene:withTransitionContext:completion:]  (in UIKitCore) + 344  [0x1b8e19a48]
      +                         +   :   | 1819 (305M) -[UIApplication _connectUISceneFromFBSScene:transitionContext:]  (in UIKitCore) + 1256  [0x1b9116c64]
      +                         +   :   | + 1322 (305M) +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:]  (in UIKitCore) + 1576  [0x1b8e17474]
      +                         +   :   | + 1292 (305M) -[UIWindowScene _makeKeyAndVisibleIfNeeded]  (in UIKitCore) + 284  [0x1b8b147c]
```

Malloc history will show you an allocation tree for the address or addresses you've provided as input. The addresses can be any set of addresses in your program.

```
> malloc_history --callTree BigMallocMSL.memgraph
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [7674]
Path:            /private/var/containers/Bundle/Application/6E568940-1386-45A1-9848-DE8BD8A6502D/TestingMemoryiO
Load Address:    0x102208000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7673]

Date/Time:       2022-09-05 16:34:08.621 +0100
Launch Time:     2022-09-05 16:33:58.982 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         314.9M
Physical footprint (peak):  321.1M
----

VM region sizes shown as dirty + swapped/compressed - purgableVolatile.

Stack logging was dynamically enabled in target process, after it was launched,
so no backtraces are available for earlier allocations.

Malloc blocks with no stack:    531   total size: 76K
   VM regions with no stack:      4   total size: 64K

Call graph:
```

```
Physical footprint (peak):  321.1M
----

VM region sizes shown as dirty + swapped/compressed – purgableVolatile.

Stack logging was dynamically enabled in target process, after it was launched,
so no backtraces are available for earlier allocations.

Malloc blocks with no stack:    531   total size: 76K
   VM regions with no stack:      4   total size: 64K

Call graph:
    21960 (307M) << TOTAL >>
      4347 (305M) start  (in dyld) + 520  [0x102601ce4]
      + 4339 (305M) main  (in TestingMemoryiOS) + 120  [0x10220e14c]
      + ! 3162 (305M) UIApplicationMain  (in UIKitCore) + 364  [0x1b8d57d90]
      + ! : 2954 (305M) -[UIApplication _run]  (in UIKitCore) + 1100  [0x1b8fd6648]
      + ! : | 2954 (305M) GSEventRunModal  (in GraphicsServices) + 164  [0x1d279a374]
      + ! : |   2953 (305M) CFRunLoopRunSpecific  (in CoreFoundation) + 600  [0x1b6666bc8]
      + ! : |   + 2871 (305M) __CFRunLoopRun  (in CoreFoundation) + 828  [0x1b665305c]
      + ! : |   + ! 2265 (305M) __CFRunLoopDoSources0  (in CoreFoundation) + 368  [0x1b664d6f8]
      + ! : |   + ! : 2265 (305M) __CFRunLoopDoSource0  (in CoreFoundation) + 208  [0x1b67141a0]
      + ! : |   + ! :   2265 (305M) __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__  (in CoreFoundation) + 28  [0x1b6703414]
      + ! : |   + ! :     2229 (305M) -[FBSSerialQueue _performNextFromRunLoopSource]  (in FrontBoardServices) + 28  [0x1c88d1700]
      + ! : |   + ! :     | 2229 (305M) -[FBSSerialQueue _targetQueue_performNextIfPossible]  (in FrontBoardServices) + 220  [0x1c88cd040]
      + ! : |   + ! :     |   2229 (305M) __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__  (in FrontBoardServices) + 48  [0x1c88cdc70]
      + ! : |   + ! :     |     2229 (305M) _dispatch_block_invoke_direct  (in libdispatch.dylib) + 368  [0x10247e264]
      + ! : |   + ! :     |       2229 (305M) _dispatch_client_callout  (in libdispatch.dylib) + 20  [0x10247a7c0]
      + ! : |   + ! :     |         2046 (305M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke  (
      + ! : |   + ! :     |         + 2043 (305M) -[FBSWorkspace _calloutQueue_executeCalloutFromSource:withBlock:]  (in FrontBoardServices) + 240  [0x1c88cc308]
      + ! : |   + ! :     |         + ! 2043 (305M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invok
      + ! : |   + ! :     |         + !   2036 (305M) -[FBSScene _callOutQueue_agent_didCreateWithTransitionContext:completion:]  (in FrontBoardServices) + 440  [
      + ! : |   + ! :     |         + !   : 2036 (305M) -[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:]  (in UIKitCore) + 388  [0x1b8d5a
      + ! : |   + ! :     |         + !   :   1895 (305M) -[UIApplication workspace:didCreateScene:withTransitionContext:completion:]  (in UIKitCore) + 344  [0x1b
      + ! : |   + ! :     |         + !   :   | 1819 (305M) -[UIApplication _connectUISceneFromFBSScene:transitionContext:]  (in UIKitCore) + 1256  [0x1b9116c64]
      + ! : |   + ! :     |         + !   :   | + 1322 (305M) +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:]  (in UIKitCore) + 1576  [0x1b8e17
      + ! : |   + ! :     |         + !   :   | + ! 1292 (305M) -[UIWindowScene _makeKeyAndVisibleIfNeeded]  (in UIKitCore) + 204  [0x1b8b14e7c]
      + ! : |   + ! :     |         + !   :   | + ! : 1179 (305M) -[UIWindow _mainQueue_makeKeyAndVisible]  (in UIKitCore) + 56  [0x1b8d8c66c]
      + ! : |   + ! :     |         + !   :   | + ! : | 1179 (305M) -[UIWindow _setHidden:forced:]  (in UIKitCore) + 280  [0x1b8d71fd0]
      + ! : |   + ! :     |         + !   :   | + ! : |   1162 (305M) -[UIWindow _updateLayerOrderingAndSetLayerHidden:actionBlock:]  (in UIKitCore) + 232  [0x1b8
      + ! : |   + ! :     |         + !   :   | + ! : |   + 325 (305M) -[UIWindow addRootViewControllerViewIfPossible]  (in UIKitCore) + 180  [0x1b8d85bf4]
      + ! : |   + ! :     |         + !   :   | + ! : |   + ! 325 (305M) -[UIViewController view]  (in UIKitCore) + 32  [0x1b8c439e4]
      + ! : |   + ! :     |         + !   :   | + ! : |   + !   300 (305M) -[UIViewController loadViewIfRequired]  (in UIKitCore) + 1048  [0x1b8c70984]
      + ! : |   + ! :     |         + !   :   | + ! : |   + !   : 300 (305M) -[UIViewController _sendViewDidLoadWithAppearanceProxyObjectTaggingEnabled]  (in UIKi
      + ! : |   + ! :     |         + !   :   | + ! : |   + !   :   300 (305M) -[ViewController viewDidLoad]  (in TestingMemoryiOS) + 120  [0x10220de1c]
      + ! : |   + ! :     |         + !   :   | + ! : |   + !   :     300 (305M) _malloc_zone_malloc  (in libsystem_malloc.dylib) + 156  [0x1c8975024]
```

ABANDONMENTS

DON'T LEAVE ME

An "*abandonment*" is when a program keeps data around that it doesn't actually need.

- From a system/code perspective, *there is no bug*
  - Therefore, there aren't really tools to catch this explicitly
- This can happen as a result of **caching** too much
- Leaving **big images** hanging around
- Or forgetting to **invalidate** something you don't need like an NSTimer

COMMAND LINE WORKFLOW

A great way to search for abandoned memory is to use the command line workflow

- grab a **memgraph** of your process while it's using excess memory
- start with **footprint**, to get the gist of where your memory is
  - from here, the two big places to look are the heap via **MALLOC** zones and image allocations from frameworks like **ImageIO**, **IOSurface**, and **CoreAnimation**
  - feel free to explore *all* the zones and try to figure out what they are with these tools

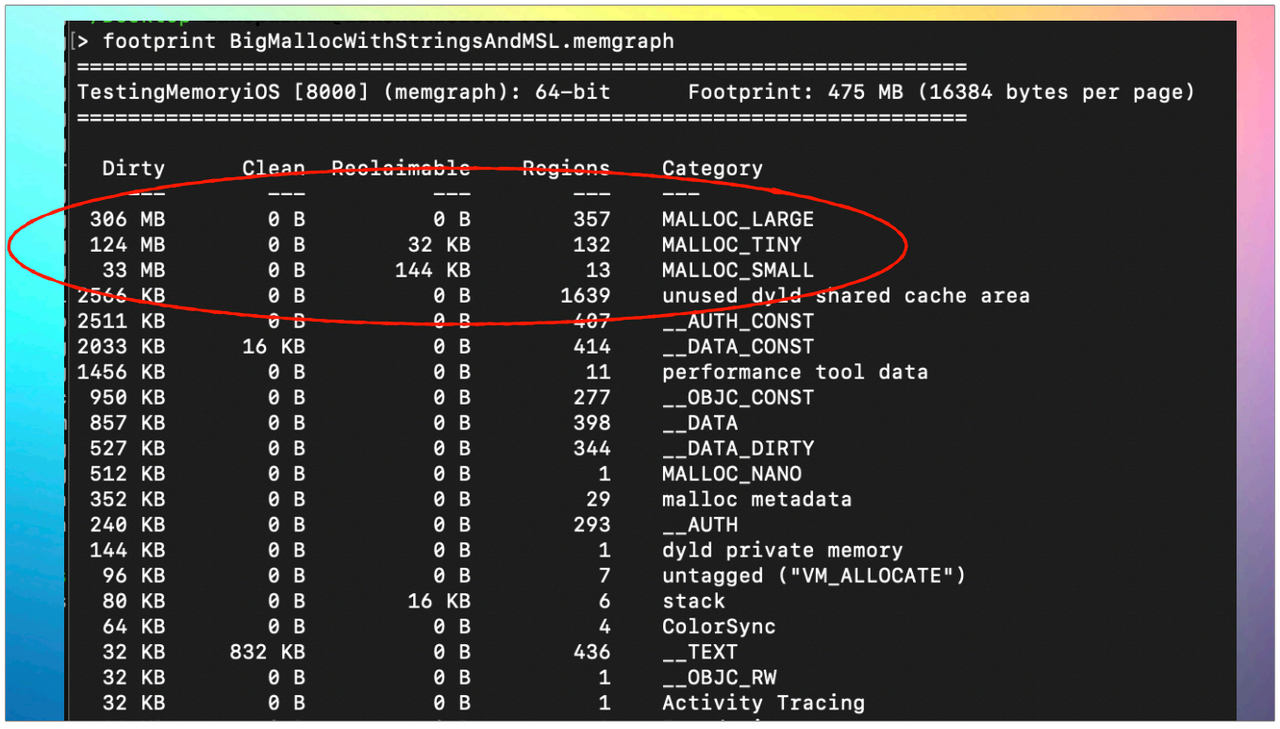
FOOTPRINT

```
> footprint BigMallocWithStringsAndMSL.memgraph
================================================================
TestingMemoryiOS [8000] (memgraph): 64-bit      Footprint: 475 MB (16384 bytes per page)
================================================================

  Dirty      Clean  Reclaimable    Regions    Category
    ---        ---          ---        ---    ---
 306 MB        0 B          0 B        357    MALLOC_LARGE
 124 MB        0 B        32 KB        132    MALLOC_TINY
  33 MB        0 B       144 KB         13    MALLOC_SMALL
2566 KB        0 B          0 B       1639    unused dyld shared cache area
2511 KB        0 B          0 B        407    __AUTH_CONST
2033 KB      16 KB          0 B        414    __DATA_CONST
1456 KB        0 B          0 B         11    performance tool data
 950 KB        0 B          0 B        277    __OBJC_CONST
 857 KB        0 B          0 B        398    __DATA
 527 KB        0 B          0 B        344    __DATA_DIRTY
 512 KB        0 B          0 B          1    MALLOC_NANO
 352 KB        0 B          0 B         29    malloc metadata
 240 KB        0 B          0 B        293    __AUTH
 144 KB        0 B          0 B          1    dyld private memory
  96 KB        0 B          0 B          7    untagged ("VM_ALLOCATE")
  80 KB        0 B        16 KB          6    stack
  64 KB        0 B          0 B          4    ColorSync
  32 KB     832 KB          0 B        436    __TEXT
  32 KB        0 B          0 B          1    __OBJC_RW
  32 KB        0 B          0 B          1    Activity Tracing
  16 KB        0 B          0 B          1    Foundation
  16 KB        0 B          0 B          1    CoreAnimation
  16 KB        0 B          0 B          1    os_alloc_once
   0 B    2080 KB          0 B          3    mapped file
   0 B     448 KB          0 B          7    __LINKEDIT
   0 B        0 B          0 B          1    __CTF
   0 B        0 B          0 B          1    __FONT_DATA
   0 B        0 B          0 B          1    __OBJC_RO
   0 B        0 B          0 B          1    __UNICODE
   ---        ---          ---        ---    ---
 475 MB    3376 KB       192 KB       4788    TOTAL

Auxiliary data:
       phys_footprint_peak: 476 MB
       phys_footprint: 476 MB
```

Footprint will give you the high level overview of your program’s memory.



You mainly want to look for megabytes here, but I spent a lot of time tracking down Kilobytes and you can be as picky as you want to be.

COMMAND LINE WORKFLOW

CONTINUED...

if there's a lot of allocations in **MALLOC** zones, then reach for **heap** next

- look for objects that seem problematic and use **malloc_history** to see where they were allocated (make sure **Malloc Stack Logging** is turned on)

- if that doesn't help, use **vmmap --traceTree** to see what's still holding onto them

Malloc stack logging can be enabled in the scheme editor in Xcode

HEAP

```
> heap BigMallocWithStringsAndMSL.memgraph
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C78BD-310B-4236-826D-B2F761362CDE/TestingMemoryiOS.app/TestingMemoryiOS
Load Address:    0x102c2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Process 8000: 4 zones

All zones: 1994162 nodes malloced - Sizes: 1040KB[350] 528KB[1] 80KB[1] 32KB[3] 10.5KB[1] 8.5KB[2] 5KB[2] 4.5KB[3989] 4KB[1] 3KB[14] 2.5KB[9] 2KB[4] 1.5KB[28] 816[1] 808[1] 752[1] 688[1] 672[8] 608[1] 576[2] 544[2]
528[28] 496[3] 480[2] 464[1] 448[2] 432[2] 400[2] 384[6] 368[2] 352[3] 336[5] 304[52] 288[10] 272[55] 256[2] 240[26] 224[25] 208[23] 192[21] 176[39] 160[80] 144[252] 128[201] 112[263] 96[772] 80[2593] 64[1976461] 48
[7589] 32[1148] 16[160]

Found 1021 ObjC classes
Found 83 CFTypes
Derived 113 type names for non-objects from allocation backtraces

----------------------------------------------------------------------
All zones: 1994162 nodes (518998912 bytes)

   COUNT      BYTES       AVG   CLASS_NAME                                    TYPE   BINARY
   =====      =====       ===   ==========                                    ====   ======
 1969980  126081520      64.0   CFString                                      ObjC   CoreFoundation
   14191     884528      62.3   malloc in add_trampoline                      C      libMainThreadChecker.dylib
    3904   17989632    4608.8   @autoreleasepool content                      C      libobjc.A.dylib
    2042      94808      46.5   Class.data (class_rw_t)                       C      libobjc.A.dylib
     349 3716718040 10647960.0   malloc in -[ViewController viewDidLoad]       C      TestingMemoryiOS
     230      14768      64.2   __NSMallocBlock__                             ObjC   libsystem_blocks.dylib
     226      54496     241.1   Class.methodCache._buckets (bucket_t)         C      libobjc.A.dylib
     222      13728      61.8   Class.data.extended (class_rw_ext_t)          C      libobjc.A.dylib
     175      17024      97.3   non-object from libsystem_notify.dylib        C      libsystem_notify.dylib
     172      11728      68.2   Class.data.methods (method_array_t)           C      libobjc.A.dylib
     133      15216     114.4   non-object in zone QuartzCore_0x103680000     <unknown>
     131       6288      48.0   NSMutableDictionary                           ObjC   CoreFoundation
     112       7168      64.0   NSMutableArray                                ObjC   CoreFoundation
     108       5632      52.1   non-object in zone DefaultMallocZone_0x102f7c000 <unknown>
      90       3856      42.8   Class.data.properties (property_array_t)      C      libobjc.A.dylib
      78       7488      96.0   BSObjCArgument                                ObjC   BaseBoard
      73       2224      30.5   NSMutableArray (Storage)                      C      CoreFoundation
      65       6888     106.0   non-object from MetalPerformanceShadersGraph  C      MetalPerformanceShadersGraph
      65       2672      41.1   Class.data.protocols (protocol_array_t)       C      libobjc.A.dylib
      57      16736     293.6   NSMutableDictionary (Storage)                 C      CoreFoundation
      54       8640     160.0   HIDServiceClient                              ObjC   IOKit
      54        928      17.2   calloc in _IOHIDServiceClientCacheProperties  C      IOKit
```

```
Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Process 8000: 4 zones

All zones: 1994162 nodes malloced - Sizes: 1040KB[350] 528KB[1] 80KB[1] 32KB[3] 10.5KB[1] 8.5KB[2] 5KB[2] 4.5KB[3909] 4KB[1] 3KB[14] 2.5KB[9] 2KB[4]
528[28] 496[3] 480[2] 464[1] 448[2] 432[2] 400[2] 384[6] 368[2] 352[3] 336[5] 304[52] 288[10] 272[55] 256[2] 240[26] 224[25] 208[23] 192[21] 176[39]
[7589] 32[1148] 16[160]

Found 1021 ObjC classes
Found 83 CFTypes
Derived 113 type names for non-objects from allocation backtraces

---------------------------------------------------------------------------
All zones: 1994162 nodes (518998912 bytes)

   COUNT      BYTES       AVG   CLASS NAME                                    TYPE    BINARY
   -----      -----       ---   ==========                                    ====    ======
 1969980  126081520      64.0   CFString                                      ObjC    CoreFoundation
   14131     884528      62.3   malloc in add_trampoline                      C       libMainThreadChecker.dylib
    3904   17989632    4608.0   @autoreleasepool content                      C       libobjc.A.dylib
    2042      94880      46.5   Class.data (class_rw_t)                       C       libobjc.A.dylib
     349  371671040 1064960.0    malloc in -[ViewController viewDidLoad]       C        TestingMemoryiOS
     230      14768      64.2   __NSMallocBlock__                             ObjC    libsystem_blocks.dylib
     226      54496     241.1   Class.methodCache._buckets (bucket_t)         C       libobjc.A.dylib
     222      13728      61.8   Class.data.extended (class_rw_ext_t)          C       libobjc.A.dylib
     175      17024      97.3   non-object from libsystem_notify.dylib        C       libsystem_notify.dylib
     172      11728      68.2   Class.data.methods (method_array_t)           C       libobjc.A.dylib
     133      15216     114.4   non-object in zone QuartzCore_0x103680000             <unknown>
     131       6288      48.0   NSMutableDictionary                           ObjC    CoreFoundation
     112       7168      64.0   NSMutableArray                                ObjC    CoreFoundation
     108       5632      52.1   non-object in zone DefaultMallocZone_0x102f7c000      <unknown>
      90       3856      42.8   Class.data.properties (property_array_t)      C       libobjc.A.dylib
      78       7488      96.0   BSObjCArgument                                ObjC    BaseBoard
      73       2224      30.5   NSMutableArray (Storage)                      C       CoreFoundation
      65       5088      78.3   non-object from MetalPerformanceShadersGraph  C       MetalPerformanceShadersGraph
      65       2672      41.1   Class.data.protocols (protocol_array_t)       C       libobjc.A.dylib
      57      16736     293.6   NSMutableDictionary (Storage)                 C       CoreFoundation
      54       8640     160.0   HIDServiceClient                              ObjC    IOKit
      54        928      17.2   calloc in _IOHIDServiceClientCacheProperties  C       IOKit
```

HEAP --ADDRESSES

```
> heap BigMallocWithStringsAndMSL.memgraph --addresses=CFString
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7B8D-3108-4236-826D-82F761362CDE/TestingMemoryiOS.app/TestingMemoryiOS
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  184

Physical footprint:        475.7M
Physical footprint (peak):  475.7M
~~~~

Active blocks in all zones that match pattern 'CFString':
0x103505b90: CFString (144 bytes)
0x1035842f0: CFString (64 bytes)
0x103584720: CFString (64 bytes)
0x103585970: CFString (64 bytes)
0x1035859b0: CFString (64 bytes)
0x1035859f0: CFString (64 bytes)
0x103585a30: CFString (64 bytes)
0x103585b10: CFString (64 bytes)
0x103585d40: CFString (64 bytes)
0x103585d80: CFString (64 bytes)
0x103585dc0: CFString (64 bytes)
0x103585e00: CFString (64 bytes)
0x1035862e0: CFString (64 bytes)
0x103586320: CFString (64 bytes)
0x1035863c0: CFString (64 bytes)
0x103586440: CFString (64 bytes)
0x103586710: CFString (64 bytes)
0x103586750: CFString (64 bytes)
0x103586790: CFString (64 bytes)
0x1035867d0: CFString (64 bytes)
0x103586810: CFString (64 bytes)
0x103586850: CFString (64 bytes)
0x103586890: CFString (64 bytes)
0x1035868d0: CFString (64 bytes)
0x103586a00: CFString (64 bytes)
0x103586a80: CFString (64 bytes)
0x103586b10: CFString (64 bytes)
0x103586b60: CFString (64 bytes)
0x103586ba0: CFString (64 bytes)
0x103586c30: CFString (80 bytes)
0x103586cb0: CFString (64 bytes)
0x103586cf0: CFString (64 bytes)
0x103586d30: CFString (64 bytes)
0x103586d70: CFString (64 bytes)
0x103586db0: CFString (64 bytes)
0x103587000: CFString (48 bytes)
0x103587030: CFString (80 bytes)
```

```
> heap BigMallocWithStringsAndMSL.memgraph --addresses=CFString
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7BBD-3108-4236-826D-82F761362CDE/TestingMem
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Active blocks in all zones that match pattern 'CFString':
0x103505b90: CFString (144 bytes)
0x1035842f0: CFString (64 bytes)
0x103584720: CFString (64 bytes)
0x103585970: CFString (64 bytes)
0x1035859b0: CFString (64 bytes)
0x1035859f0: CFString (64 bytes)
0x103585a30: CFString (64 bytes)
0x103585b10: CFString (64 bytes)
0x103585d40: CFString (64 bytes)
```

```
> heap BigMallocWithStringsAndMSL.memgraph --addresses=CFString
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7BBD-3108-4236-826D-82F761362CDE/TestingMem
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Active blocks in all zones that match pattern 'CFString':
0x103505b90: CFString (144 bytes)
0x1035842f0: CFString (64 bytes)
0x103584720: CFString (64 bytes)
0x103585970: CFString (64 bytes)
0x1035859b0: CFString (64 bytes)
0x1035859f0: CFString (64 bytes)
0x103585a30: CFString (64 bytes)
0x103585b10: CFString (64 bytes)
0x103585d40: CFString (64 bytes)
```

MALLOC_HISTORY

```
> malloc_history -callTree BigMallocWithStringsAndMSL.memgraph 0x103505b90
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C78BD-3108-4236-826D-82F761362CDE/TestingMemoryiOS.app/TestingMemoryiOS
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
-----

Stack backtrace for allocation 0x103505b90

Call graph:
    1 (144 bytes) start  (in dyld) + 488  [0x1030ddcc4]
      1 (144 bytes) dyld4::prepare(dyld4::APIs&, dyld3::MachOAnalyzer const*)  (in dyld) + 3860  [0x10304f76c]
        1 (144 bytes) dyld4::APIs::runAllInitializersForMain()  (in dyld) + 312  [0x1030644a8]
          1 (144 bytes) dyld4::Loader::runInitializersBottomUpPlusUpwardLinks(dyld4::RuntimeState&) const  (in dyld) + 124  [0x10304479c]
            1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const  (in dyld) + 180  [0x10303e560]
              1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const  (in dyld) + 180  [0x10303e560]
                1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const  (in dyld) + 216  [0x10303e584]
                  1 (144 bytes) dyld4::Loader::findAndRunAllInitializers(dyld4::RuntimeState&) const  (in dyld) + 172  [0x1030422c28]
                    1 (144 bytes) dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&, dyld3::MachOAnalyzer::VMAddrConverter const&, void (unsigned int) block_pointer, void const*) const  (in dyld) + 516  [0x10304
6108]
                      1 (144 bytes) dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::SectionInfo const&, bool, bool&) block_pointer) const  (in dyld) + 192  [0x10303ba04]
                        1 (144 bytes) dyld3::MachOFile::forEachLoadCommand(Diagnostics&, void (load_command const*, bool&) block_pointer) const  (in dyld) + 168  [0x1030396b8]
                          1 (144 bytes) invocation function for block in dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::SectionInfo const&, bool, bool&) block_pointer) const  (in dyld) + 528  [0x10303a4d8]
                            1 (144 bytes) invocation function for block in dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&, dyld3::MachOAnalyzer::VMAddrConverter const&, void (unsigned int) block_pointer, void
 const*) const  (in dyld) + 340  [0x1030709b4]
                              1 (144 bytes) invocation function for block in dyld4::Loader::findAndRunAllInitializers(dyld4::RuntimeState&) const  (in dyld) + 164  [0x10303c910]
                                1 (144 bytes) __CFInitialize  (in CoreFoundation) + 988  [0x1b66cc9c0]
                                  1 (144 bytes) CFStringCreateWithCString  (in CoreFoundation) + 96  [0x1b667cf04]
                                    1 (144 bytes) __CFStringCreateImmutableFunnel3  (in CoreFoundation) + 1848  [0x1b666f838]
                                      1 (144 bytes) _CFRuntimeCreateInstance  (in CoreFoundation) + 332  [0x1b6673628]
                                        1 (144 bytes) _malloc_zone_calloc  (in libsystem_malloc.dylib) + 88  [0x1c8973e5c]
```

```
> malloc_history -callTree BigMallocWithStringsAndMSL.memgraph 0x103505b90
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7BBD-3108-4236-826D-82F761362CDE/TestingMemoryiOS.app/TestingMemor
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Stack backtrace for allocation 0x103505b90

Call graph:
    1 (144 bytes) start  (in dyld) + 488  [0x10304dcc4]
      1 (144 bytes) dyld4::prepare(dyld4::APIs&, dyld3::MachOAnalyzer const*)  (in dyld) + 3060  [0x10304f76c]
        1 (144 bytes) dyld4::APIs::runAllInitializersForMain()  (in dyld) + 312  [0x1030644a0]
          1 (144 bytes) dyld4::Loader::runInitializersBottomUpPlusUpwardLinks(dyld4::RuntimeState&) const  (in dyld) + 124  [0x103
            1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const
              1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) cons
                1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) co
                  1 (144 bytes) dyld4::Loader::findAndRunAllInitializers(dyld4::RuntimeState&) const  (in dyld) + 172  [0x103042c2
                    1 (144 bytes) dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&, dyld3::MachOAnalyzer::VMAddrConverter con
6108]
                      1 (144 bytes) dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::SectionInfo const&, bool, bool&) bloc
```

```
> malloc_history -callTree BigMallocWithStringsAndMSL.memgrapl  0x103505b90
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7BBD-3108-4236-826D-82F761362CDE/TestingMemoryiOS.app/TestingMemor
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Stack backtrace for allocation 0x103505b90

Call graph:
    1 (144 bytes) start  (in dyld) + 488  [0x10304dcc4]
      1 (144 bytes) dyld4::prepare(dyld4::APIs&, dyld3::MachOAnalyzer const*)  (in dyld) + 3060  [0x10304f76c]
        1 (144 bytes) dyld4::APIs::runAllInitializersForMain()  (in dyld) + 312  [0x1030644a0]
          1 (144 bytes) dyld4::Loader::runInitializersBottomUpPlusUpwardLinks(dyld4::RuntimeState&) const  (in dyld) + 124  [0x103
            1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const
              1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) cons
                1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) co
                  1 (144 bytes) dyld4::Loader::findAndRunAllInitializers(dyld4::RuntimeState&) const  (in dyld) + 172  [0x103042c2
                    1 (144 bytes) dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&, dyld3::MachOAnalyzer::VMAddrConverter con
6108]
                      1 (144 bytes) dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::SectionInfo const&, bool, bool&) bloc
```

```
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Stack backtrace for allocation 0x103505b90

Call graph:
    1 (144 bytes) start  (in dyld) + 488  [0x10304dcc4]
      1 (144 bytes) dyld4::prepare(dyld4::APIs&, dyld3::MachOAnalyzer const*)  (in dyld) + 3060  [0x10304f76c]
        1 (144 bytes) dyld4::APIs::runAllInitializersForMain()  (in dyld) + 312  [0x1030644a0]
          1 (144 bytes) dyld4::Loader::runInitializersBottomUpPlusUpwardLinks(dyld4::RuntimeState&) const  (in dyld) + 124  [0x103
            1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) const
              1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) con
                1 (144 bytes) dyld4::Loader::runInitializersBottomUp(dyld4::RuntimeState&, dyld3::Array<dyld4::Loader const*>&) c
                  1 (144 bytes) dyld4::Loader::findAndRunAllInitializers(dyld4::RuntimeState&) const  (in dyld) + 172  [0x103042c
                    1 (144 bytes) dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&, dyld3::MachOAnalyzer::VMAddrConverter co
6108]
                      1 (144 bytes) dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::SectionInfo const&, bool, bool&) blo
                        1 (144 bytes) dyld3::MachOFile::forEachLoadCommand(Diagnostics&, void (load_command const*, bool&) block_
                          1 (144 bytes) invocation function for block in dyld3::MachOFile::forEachSection(void (dyld3::MachOFile::
                            1 (144 bytes) invocation function for block in dyld3::MachOAnalyzer::forEachInitializer(Diagnostics&,
 const*) const  (in dyld) + 340  [0x1030709b4]
                              1 (144 bytes) invocation function for block in dyld4::Loader::findAndRunAllInitializers(dyld4::Runti
                                1 (144 bytes) __CFInitialize  (in CoreFoundation) + 988  [0x1b66cc9c0]
                                  1 (144 bytes) CFStringCreateWithCString  (in CoreFoundation) + 96  [0x1b667cf04]
                                    1 (144 bytes) __CFStringCreateImmutableFunnel3  (in CoreFoundation) + 1848  [0x1b666f838]
                                      1 (144 bytes) _CFRuntimeCreateInstance  (in CoreFoundation) + 332  [0x1b6673628]
                                        1 (144 bytes) _malloc_zone_calloc  (in libsystem_malloc.dylib) + 88  [0x1c8973e5c]
```

COMMAND LINE WORKFLOW

CONTINUED...

If you want to see where **all of a certain type** of memory in a program is coming from, you'll need to combine some tools

- malloc_history
- heap --addresses
- awk
- grep
-head

Malloc stack logging can be enabled in the scheme editor in Xcode

COMMAND LINE WORKFLOW

1) heap BigMallocWithStringsAndMSL.memgraph --addresses

2) awk -F “:” ‘{print $1}’

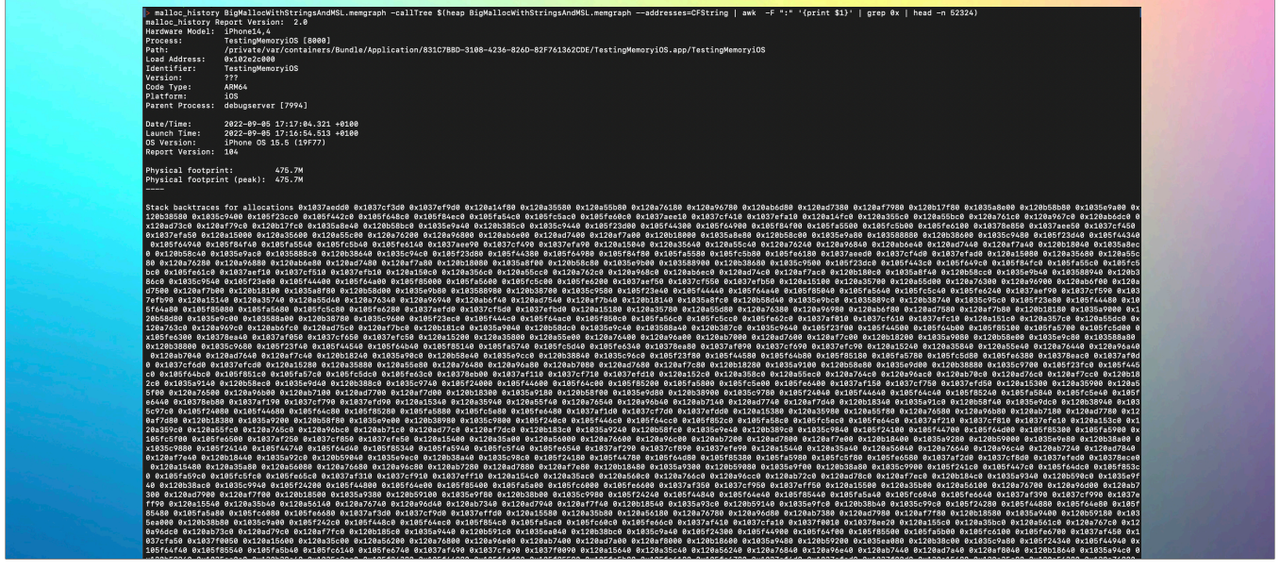
3) grep 0x

4) head -n

5) malloc_history -callTree

malloc_history BigMallocWithStringsAndMSL.memgraph -callTree $(heap BigMallocWithStringsAndMSL.memgraph --addresses=CFString | awk -F ":" '{print $1}' | grep 0x | head -n 52324)

COMMAND LINE WORKFLOW



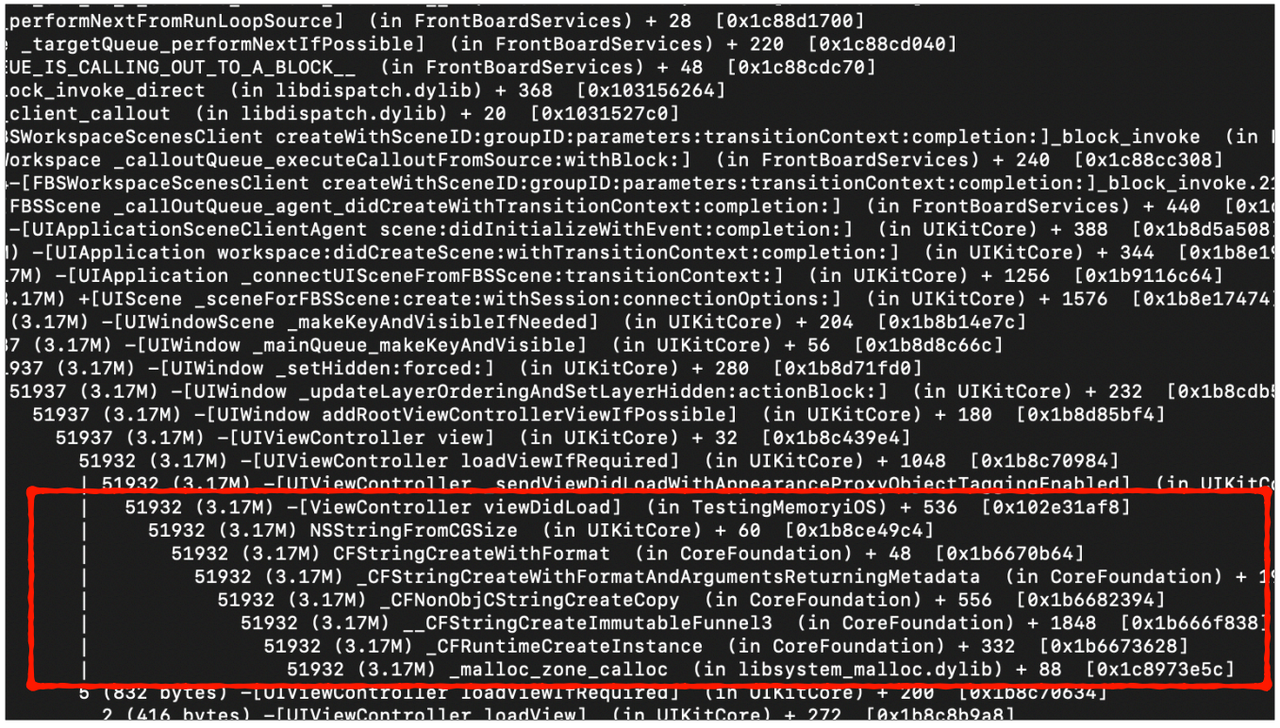
```
> malloc_history BigMallocWithStringsAndMSL.memgraph -callTree $(heap BigMallocWithStringsAndMSL.memgrap
malloc_history Report Version:  2.0
Hardware Model:  iPhone14,4
Process:         TestingMemoryiOS [8000]
Path:            /private/var/containers/Bundle/Application/831C7BBD-3108-4236-826D-82F761362CDE/Testing
Load Address:    0x102e2c000
Identifier:      TestingMemoryiOS
Version:         ???
Code Type:       ARM64
Platform:        iOS
Parent Process:  debugserver [7994]

Date/Time:       2022-09-05 17:17:04.321 +0100
Launch Time:     2022-09-05 17:16:54.513 +0100
OS Version:      iPhone OS 15.5 (19F77)
Report Version:  104

Physical footprint:         475.7M
Physical footprint (peak):  475.7M
----

Stack backtraces for allocations 0x1037aedd0 0x1037cf3d0 0x1037ef9d0 0x120a14f80 0x120a35580 0x120a55b80
120b38580 0x1035c9400 0x105f23cc0 0x105f442c0 0x105f648c0 0x105f84ec0 0x105fa54c0 0x105fc5ac0 0x105fe60c
x120ad73c0 0x120af79c0 0x120b17fc0 0x1035a8e40 0x120b58bc0 0x1035e9a40 0x120b385c0 0x1035c9440 0x105f23d
0x1037efa50 0x120a15000 0x120a35600 0x120a55c00 0x120a76200 0x120a96800 0x120ab6e00 0x120ad7400 0x120af7
 0x105f64940 0x105f84f40 0x105fa5540 0x105fc5b40 0x105fe6140 0x1037aee90 0x1037cf490 0x1037efa90 0x120a1
0 0x120b58c40 0x1035e9ac0 0x1035888c0 0x120b38640 0x1035c94c0 0x105f23d80 0x105f44380 0x105f64980 0x105f
```

```
Call graph:
    52324 (3.20M) << TOTAL >>
      52119 (3.18M) start  (in dyld) + 520  [0x10304dce4]
      + 52118 (3.18M) main  (in TestingMemoryiOS) + 120  [0x102e31e14]
      + ! 51971 (3.17M) UIApplicationMain  (in UIKitCore) + 364  [0x1b8d57d90]
      + ! : 51970 (3.17M) -[UIApplication _run]  (in UIKitCore) + 1100  [0x1b8fd6648]
      + ! : | 51970 (3.17M) GSEventRunModal  (in GraphicsServices) + 164  [0x1d279a374]
      + ! : |   51970 (3.17M) CFRunLoopRunSpecific  (in CoreFoundation) + 600  [0x1b6666bc8]
      + ! : |     51970 (3.17M) __CFRunLoopRun  (in CoreFoundation) + 828  [0x1b665305c]
      + ! : |       51970 (3.17M) __CFRunLoopDoSources0  (in CoreFoundation) + 368  [0x1b664d6f8]
      + ! : |         51970 (3.17M) __CFRunLoopDoSource0  (in CoreFoundation) + 208  [0x1b67141a0]
      + ! : |           51970 (3.17M) __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__  (in CoreFoun
      + ! : |             51970 (3.17M) -[FBSSerialQueue _performNextFromRunLoopSource]  (in FrontBoardService
      + ! : |               51970 (3.17M) -[FBSSerialQueue _targetQueue_performNextIfPossible]  (in FrontBoard
      + ! : |                 51970 (3.17M) __FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__  (in FrontBoardServic
      + ! : |                   51970 (3.17M) _dispatch_block_invoke_direct  (in libdispatch.dylib) + 368  [0x
      + ! : |                     51970 (3.17M) _dispatch_client_callout  (in libdispatch.dylib) + 20  [0x1031
      + ! : |                       51970 (3.17M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID:par
      + ! : |                         51970 (3.17M) -[FBSWorkspace _calloutQueue_executeCalloutFromSource:with
      + ! : |                           51970 (3.17M) __94-[FBSWorkspaceScenesClient createWithSceneID:groupID
      + ! : |                             51969 (3.17M) -[FBSScene _callOutQueue_agent_didCreateWithTransition
      + ! : |                             + 51969 (3.17M) -[UIApplicationSceneClientAgent scene:didInitializeW
      + ! : |                             +   51969 (3.17M) -[UIApplication workspace:didCreateScene:withTrans
      + ! : |                             +     51959 (3.17M) -[UIApplication _connectUISceneFromFBSScene:tran
      + ! : |                             +     ! 51937 (3.17M) +[UIScene _sceneForFBSScene:create:withSession
      + ! : |                             +     ! : 51937 (3.17M) -[UIWindowScene _makeKeyAndVisibleIfNeeded]
      + ! : |                             +     ! :   51937 (3.17M) -[UIWindow _mainQueue_makeKeyAndVisible]
      + ! : |                             +     ! :     51937 (3.17M) -[UIWindow _setHidden:forced:]  (in UIKi
      + ! : |                             +     ! :       51937 (3.17M) -[UIWindow _updateLayerOrderingAndSetL
      + ! : |                             +     ! :         51937 (3.17M) -[UIWindow addRootViewControllerView
```


performNextFromRunLoopSource] (in FrontBoardServices) + 28 [0x1c88d1700]
_targetQueue_performNextIfPossible] (in FrontBoardServices) + 220 [0x1c88cd040]
UE_IS_CALLING_OUT_TO_A_BLOCK__ (in FrontBoardServices) + 48 [0x1c88cdc70]
lock_invoke_direct (in libdispatch.dylib) + 368 [0x103156264]
client_callout (in libdispatch.dylib) + 20 [0x1031527c0]
SWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke (in
Workspace _calloutQueue_executeCalloutFromSource:withBlock:] (in FrontBoardServices) + 240 [0x1c88cc308]
-[FBSWorkspaceScenesClient createWithSceneID:groupID:parameters:transitionContext:completion:]_block_invoke.2
FBSScene _callOutQueue_agent_didCreateWithTransitionContext:completion:] (in FrontBoardServices) + 440 [0x1
-[UIApplicationSceneClientAgent scene:didInitializeWithEvent:completion:] (in UIKitCore) + 388 [0x1b8d5a508]
) -[UIApplication workspace:didCreateScene:withTransitionContext:completion:] (in UIKitCore) + 344 [0x1b8e1
7M) -[UIApplication _connectUISceneFromFBSScene:transitionContext:] (in UIKitCore) + 1256 [0x1b9116c64]
.17M) +[UIScene _sceneForFBSScene:create:withSession:connectionOptions:] (in UIKitCore) + 1576 [0x1b8e17474]
(3.17M) -[UIWindowScene _makeKeyAndVisibleIfNeeded] (in UIKitCore) + 204 [0x1b8b14e7c]
7 (3.17M) -[UIWindow _mainQueue_makeKeyAndVisible] (in UIKitCore) + 56 [0x1b8d8c66c]
937 (3.17M) -[UIWindow _setHidden:forced:] (in UIKitCore) + 280 [0x1b8d71fd0]
51937 (3.17M) -[UIWindow _updateLayerOrderingAndSetLayerHidden:actionBlock:] (in UIKitCore) + 232 [0x1b8cdb
51937 (3.17M) -[UIWindow addRootViewControllerViewIfPossible] (in UIKitCore) + 180 [0x1b8d85bf4]
51937 (3.17M) -[UIViewController view] (in UIKitCore) + 32 [0x1b8c439e4]
51932 (3.17M) -[UIViewController loadViewIfRequired] (in UIKitCore) + 1048 [0x1b8c70984]
51932 (3.17M) -[UIViewController _sendViewDidLoadWithAppearanceProxyObjectTaggingEnabled] (in UIKitC
51932 (3.17M) -[ViewController viewDidLoad] (in TestingMemoryiOS) + 536 [0x102e31af8]
51932 (3.17M) NSStringFromCGSize (in UIKitCore) + 60 [0x1b8ce49c4]
51932 (3.17M) CFStringCreateWithFormat (in CoreFoundation) + 48 [0x1b6670b64]
51932 (3.17M) _CFStringCreateWithFormatAndArgumentsReturningMetadata (in CoreFoundation) + 1
51932 (3.17M) _CFNonObjCStringCreateCopy (in CoreFoundation) + 556 [0x1b6682394]
51932 (3.17M) __CFStringCreateImmutableFunnel3 (in CoreFoundation) + 1848 [0x1b666f838]
51932 (3.17M) _CFRuntimeCreateInstance (in CoreFoundation) + 332 [0x1b6673628]
51932 (3.17M) _malloc_zone_calloc (in libsystem_malloc.dylib) + 88 [0x1c8973e5c]
5 (832 bytes) -[UIViewController loadViewIfRequired] (in UIKitCore) + 200 [0x1b8c70634]
2 (416 bytes) -[UIViewController loadView] (in UIKitCore) + 272 [0x1b8c8b9a8]

MEMORY SMASHERS

These are a class of bugs that’s just really hard to diagnose.
I don’t know about you, but I still remember hitting bugs earlier in my career where, thinking back, I’m pretty sure happened as a result of a memory smasher.

They are often the bugs where you’re sitting there thinking “this should not be possible..."

https://andrewhoos.com/blog/what-is-a-memory-smasher

HULK SMASH

A “memory smasher” is an errant piece of code that writes garbage into a random location

- This garbage may not cause a crash right away, but instead lays in wait...
  - ie) partially **overwriting a pointer value**
- Then, at some point, an unrelated line of code sees an obscure crash!
  - ie) a -**methodNotFound** Exception when there shouldn’t be one...

@LUKEPARHAM

ASAN


BATH TIME
Well come on man! PB's not gonna want you if you smell like rancid old socks!
jackie-lyn.deviantart.com

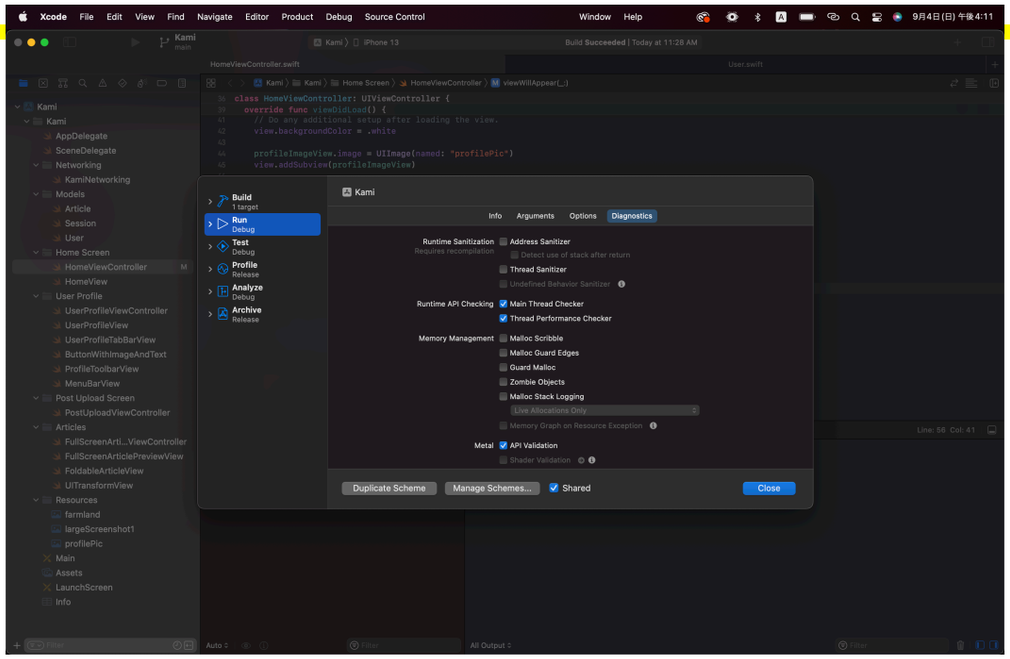
ENABLING ASAN

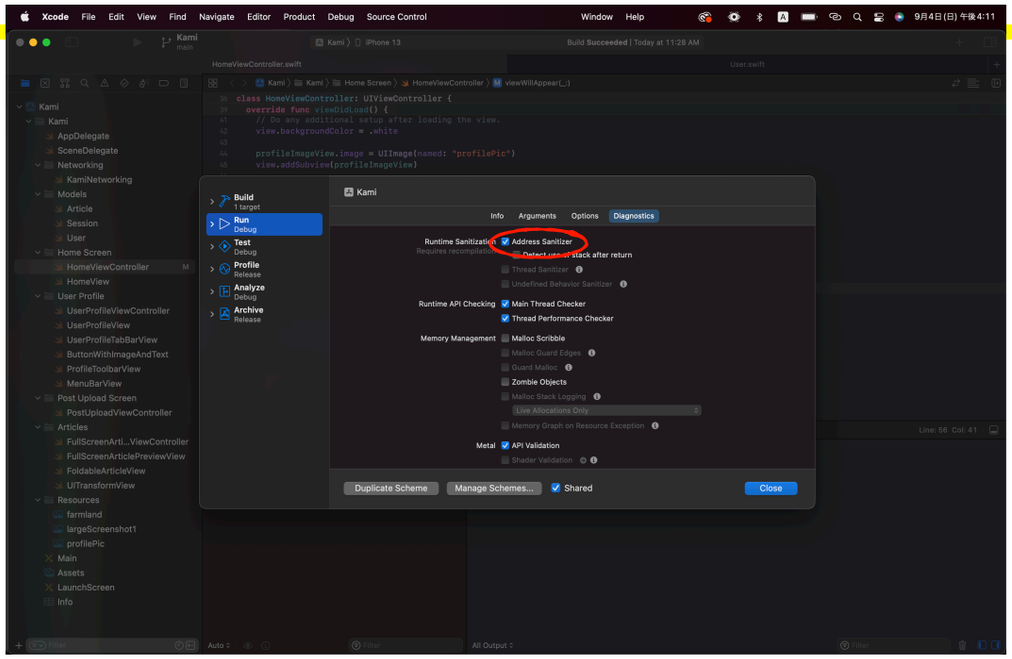
THE ADDRESS SANITIZER

- ASAN compiles your app so that all memory writes and reads are tracked to ensure they’re doing what they’re supposed to

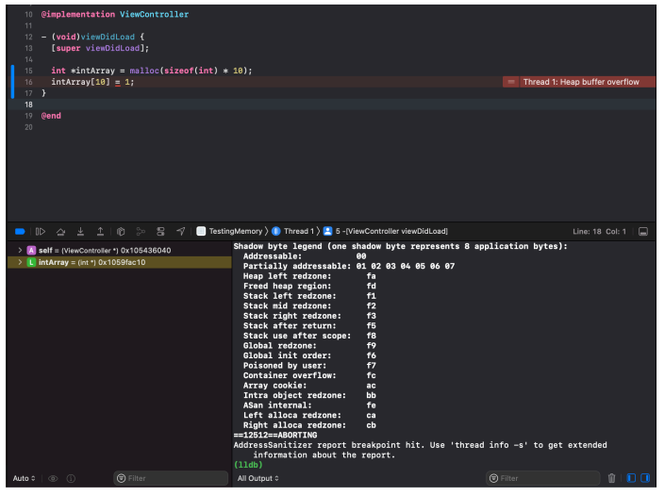
- If an errant write is detected, the program will crash at that line and expose a previously hard to diagnose bug

https://developer.apple.com/library/archive/documentation/Performance/Conceptual/ManagingMemory/Articles/MallocDebug.html#//apple_ref/doc/uid/20001884-CJBJFIDD

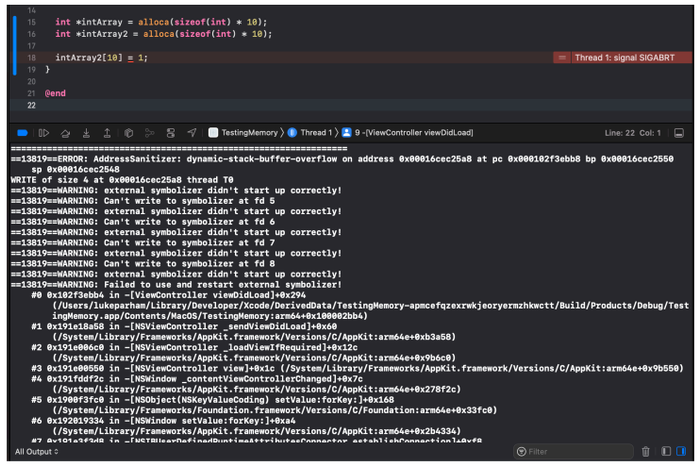

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Kami
Build
1 target
Run
Debug
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Info
Arguments
Options
Diagnostics
Runtime Sanitization
Requires recompilation
Address Sanitizer
Detect use of stack after return
Thread Sanitizer
Undefined Behavior Sanitizer
Runtime API Checking
Main Thread Checker
Thread Performance Checker
Memory Management
Malloc Scribble
Malloc Guard Edges
Guard Malloc
Zombie Objects
Malloc Stack Logging
Live Allocations Only
Memory Graph on Resource Exception
Metal
API Validation
Shader Validation
Duplicate Scheme
Manage Schemes...
Shared
Close


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Build Succeeded | Today at 11:28 AM
Kami
Build
1 target
Run
Debug
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Info
Arguments
Options
Diagnostics
Runtime Sanitization
Address Sanitizer
Thread Sanitizer
Undefined Behavior Sanitizer
Runtime API Checking
Main Thread Checker
Thread Performance Checker
Memory Management
Malloc Scribble
Malloc Guard Edges
Guard Malloc
Zombie Objects
Malloc Stack Logging
Live Allocations Only
Memory Graph on Resource Exception
Metal
API Validation
Shader Validation
Duplicate Scheme
Manage Schemes...
Shared
Close

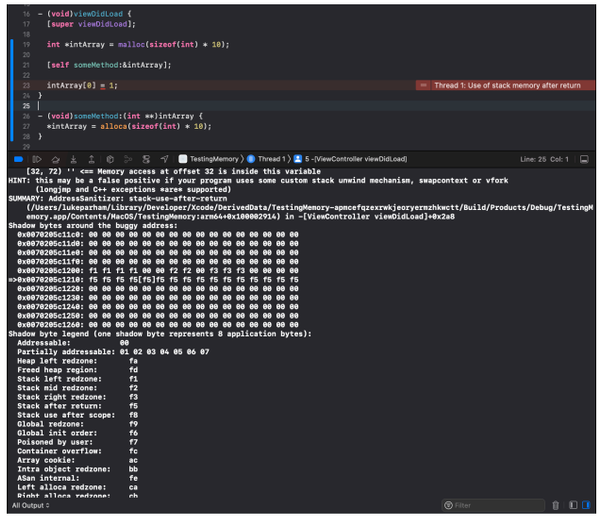
HEAP BUFFER OVERFLOW



STACK BUFFER OVERFLOW



STACK USE AFTER RETURN



WHERE TO GO FROM HERE?

A BUNCH OF USEFUL LINKS

GOODBYE FOR NOW

- Real-World Apple Performance Optimization
- Memgraph Capture the Flag
- What every programmer should know about memory
- Data Oriented Design - Mike Acton
- *OS Internals Books
- Various WWDC Talks

@LUKEPARHAM

Diagnosing Memory, Thread and Crash Issues Early
https://developer.apple.com/documentation/xcode/diagnosing-memory-thread-and-crash-issues-early

What every programmer should know about memory
https://lwn.net/Articles/250967/

*OS Internals
https://www.amazon.com/MacOS-iOS-Internals-User-Mode/dp/099105556X/ref=sr_1_2?crid=3MGBOZ3HFO1PP&keywords=ios+internals+volume+1&qid=1662306761&sprefix=%2Caps%2C109&sr=8-2

Memgraph Capture the Flag
https://developer.apple.com/news/?id=keebiiyl

CTF Solution
https://developer.apple.com/news/?id=3st92oup

Apple Docs

https://developer.apple.com/library/archive/documentation/Performance/Conceptual/ManagingMemory/Articles/AboutMemory.html

WWDC
https://developer.apple.com/videos/play/wwdc2018/416/
https://developer.apple.com/videos/play/wwdc2015/413/