

# Crash Course on Crashes

Liviu Romascanu



# ABOUT ME

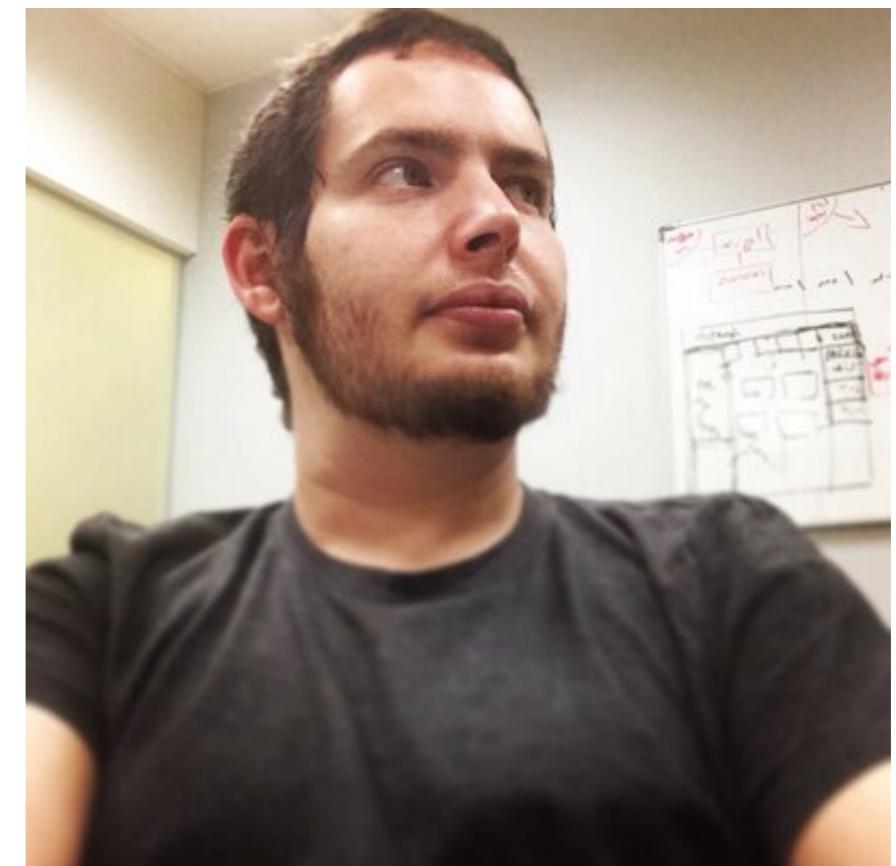
---

HAVE BEEN IN THE IOS WORLD  
FOR THE PAST 9 YEARS

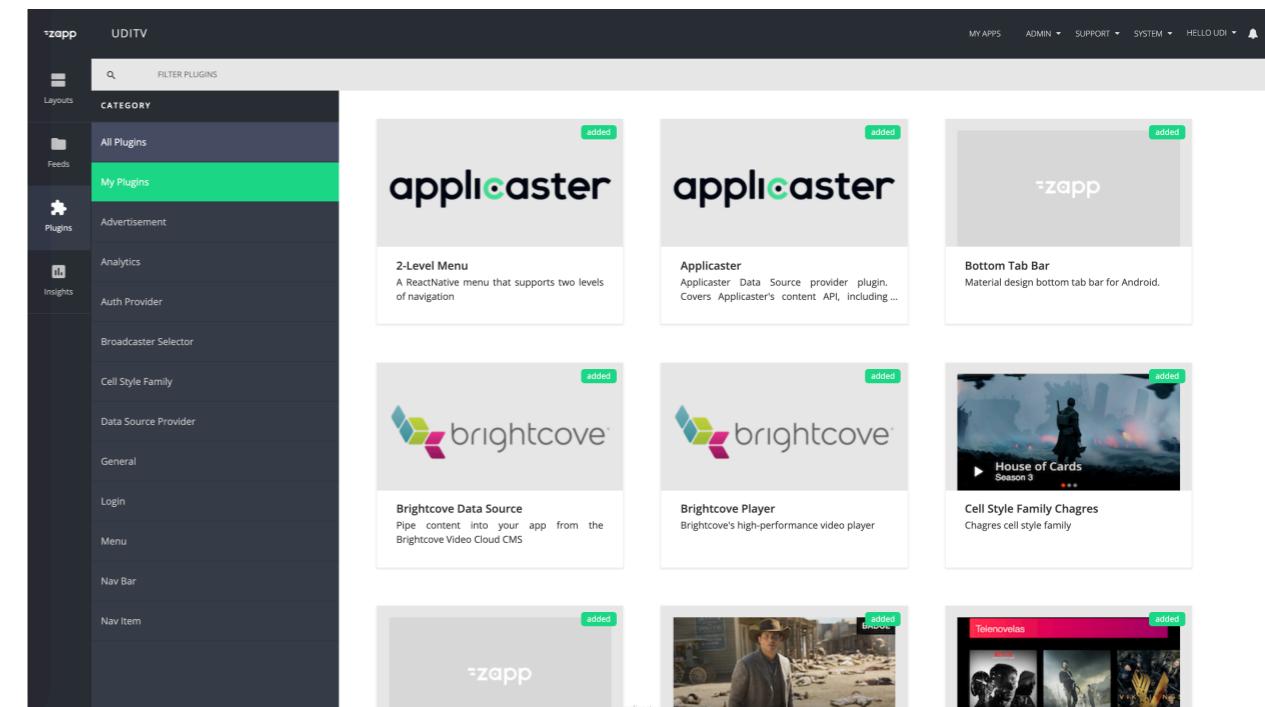
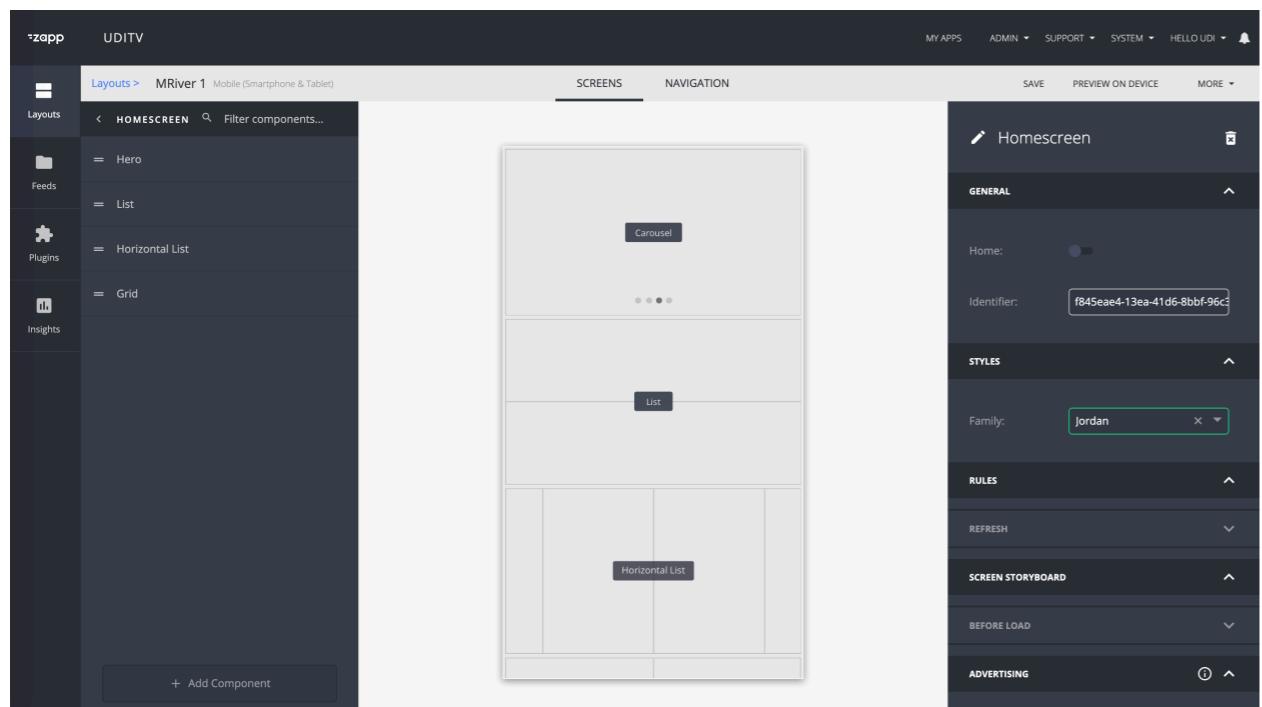
- Doing iOS development, React Native, Rails and NodeJS.
- Been with Applicaster for the past 8 years.
- Lead developer and head of Dev Relations team at Applicaster

Email me at [l.romasca@applicaster.com](mailto:l.romasca@applicaster.com)

Twitter @livius16



# APPLICASTER



# WHAT WE ARE GOING TO DISCUSS TODAY

---

- COMMON CRASHES

A SHORT LIST OF COMMON CRASHES WE ENCOUNTERED AND HANDLED INCLUDING DEBUG TRICKS TO HELP CATCH THOSE COMMON CRASHES

- CRASH LOGS ANATOMY

HOW DOES A CRASH LOG LOOK LIKE, HOW TO READ THROUGH IT QUICKLY AND HOW TO ADD SYMBOLS IN CASE THEY AREN'T SYMBOLICATED AUTOMATICALLY

- COLLECTING CRASH LOGS

HOW TO COLLECT CRASH LOGS LOCALLY, FROM ITUNES CONNECT AND FROM CRASH LOG SERVICES

- ALL THE CODE EXAMPLES AND SLIDES ARE AVAILABLE

[HTTP://WWW.GITHUB.COM/APPLICASTER/ALTCNF2019](http://www.github.com/applicaster/altconf2019)

# GOALS

---

- IDENTIFY THE LINE WHERE THE CODE CRASHES

BEST CASE SCENARIO - WE ARE ABLE TO PINPOINT THE EXACT LINE OF CODE IN OUR CODE THAT CAUSES THE ISSUE.

- UNDERSTAND THE REASON

WHY DID THE CRASH HAPPEN? UNDERSTANDING WHY ALWAYS LEADS TO UNDERSTANDING HOW TO FIX

- SHORTEN UNREADABLE STACK / IMPROVE DEBUGGING

WHEN CRASH DOESN'T HAPPEN DIRECTLY IN OUR CODE, WE WANT TO BE ABLE TO SHORTEN THE UNREADABLE PORTION OF THE STACK TRACE AND TO GET AS MUCH INFO AS POSSIBLE ON THE OBJECT

# ABOUT THE EXAMPLES

---

- **SIMPLE**

SIMPLE EXAMPLES OUT OF ONE SINGLE VIEW APPLICATION CONTAINING ALL THE CODE

- **NO NETWORKING CODE / ASYNC CODE**

IN REAL LIFE MOST ISSUES WOULD HAPPEN IN MORE COMPLEX, NON-SYNCHRONOUS CODE AND/OR INVOLVE OTHER LIBRARIES

- **NO DEPENDANCIES - OPEN SOURCE OR PRECOMPILED LIBRARIES**

IN REAL LIFE, MANY OF US USE COCOAPODS AND IMPORT PRE-COMPILED LIBRARIES IN ORDER TO SHORTEN DEVELOPMENT TIME AND REDUCE BOILERPLATE CODE AND AVOID “RE-INVENTING THE WHEEL” WHICH IS A GOOD IDEA - HOWEVER THIS COMPLICATES DEBUGGING.

# COMMON CRASHES

---

- UNRECOGNIZED SELECTOR

WHEN YOU EXPECT ONE OBJECT TYPE BUT YOU ACTUALLY GET ANOTHER.

- BAD MEMORY ACCESS

MOST COMMONLY DUE TO TRYING TO ACCESS RELEASED OBJECTS AND GETTING GARBAGE FROM THE MEMORY

- WATCHDOGS CRASHES

MOSTLY DUE TO FAST ALLOCATION OF MEMORY IN A SHORT TIME OR SIGNIFICANT MEMORY LEAKS

- THERE ARE MANY MORE CRASHES

I'M SURE YOU'E ENCOUNTERED THEM: OUT OF BOUNDS, FORCE UNWRAPPING WRONG CLASSES ETC.

- THE ONES ABOVE ARE COMMON EXAMPLES WHERE IN MY EXPERIENCE XCODE DOESN'T DO A GREAT JOB OF FLUSHING OUT THE CULPRIT.

# COMMON CRASHES

---

GO TO [GITHUB.COM/APPLICASTER/ALTCNF2019](https://github.com/applicaster/altconf2019)

THE CODE CONTAINS A SAMPLE THAT WILL HELP YOU REPRODUCE EACH SCENARIO.

# UNRECOGNIZED SELECTOR

---

HAPPENS WHEN A FUNCTION IS CALLED ON A CLASS /  
OBJECT THAT DOESN'T IMPLEMENT THEM  
YOU THINK IT'S ONE TYPE OF OBJECT BUT ACTUALLY ITS NOT

WAY MORE COMMON IN OBJECTIVE C THAN SWIFT  
BUT YOU CAN DEFINITELY FIND THEM IN SWIFT - ESPECIALLY IF YOUR FOND OF “!”  
WHEN UNWRAPPING OBJECTS.

```
NSObject *object =[NSString stringWithFormat:@"[Foo, Bar"];  
  
for (NSString *string in ((NSArray *)object))  
{  
    NSLog(@"%@", string);  
}
```

# UNRECOGNIZED SELECTOR

```
2019-05-25 19:57:45.183542-0400 AltConf1[30077:740623] -[__NSCFString countByEnumeratingWithState:objects:count]: unrecognized
selector sent to instance 0x600000cc7760
2019-05-25 19:57:45.227413-0400 AltConf1[30077:740623] *** Terminating app due to uncaught exception 'NSInvalidArgumentException',
reason: '-[__NSCFString countByEnumeratingWithState:objects:count]: unrecognized selector sent to instance 0x600000cc7760'
*** First throw call stack:
(
    0   CoreFoundation                      0x0000000105bf01bb __exceptionPreprocess + 331
    1   libobjc.A.dylib                     0x000000010518e735 objc_exception_throw + 48
    2   CoreFoundation                      0x0000000105c0ef44 -[NSObject(NSObject) doesNotRecognizeSelector:] + 132
    3   CoreFoundation                      0x0000000105bf4ed6 ___forwarding___ + 1446
    4   CoreFoundation                      0x0000000105bf6da8 _CF_forwarding_prep_0 + 120
    5   AltConf1                            0x000000010487152d -[ViewController unrecognizedSelectorClicked:] + 205
    6   UIKitCore                           0x0000000108a58ecb -[UIApplication sendAction:to:from:forEvent:] + 83
    7   UIKitCore                           0x00000001084940bd -[UIControl sendAction:to:forEvent:] + 67
    8   UIKitCore                           0x00000001084943da -[UIControl _sendActionsForEvents:withEvent:] + 450
    9   UIKitCore                           0x000000010849331e -[UIControl touchesEnded:withEvent:] + 583
    10  UIKitCore                           0x0000000108a940a4 -[UIWindow _sendTouchesForEvent:] + 2729
    11  UIKitCore                           0x0000000108a957a0 -[UIWindow sendEvent:] + 4080
    12  UIKitCore                           0x0000000108a73394 -[UIApplication sendEvent:] + 352
    13  UIKitCore                           0x0000000108b485a9 __dispatchPreprocessedEventFromEventQueue + 3054
    14  UIKitCore                           0x0000000108b4b1cb __handleEventQueueInternal + 5948
    15  CoreFoundation                      0x0000000105b55721 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 17
    16  CoreFoundation                      0x0000000105b54f93 __CFRunLoopDoSources0 + 243
    17  CoreFoundation                      0x0000000105b4f63f __CFRunLoopRun + 1263
    18  CoreFoundation                      0x0000000105b4ee11 CFRunLoopRunSpecific + 625
    19  GraphicsServices                   0x000000010e1e81dd GSEventRunModal + 62
    20  UIKitCore                           0x0000000108a5781d UIApplicationMain + 140
    21  AltConf1                            0x0000000104871730 main + 112
    22  libdyld.dylib                      0x0000000107564575 start + 1
    23  ???                                0x0000000000000001 0x0 + 1
)
libc++abi.dylib: terminating with uncaught exception of type NSException
(lldb)
```

# UNRECOGNIZED SELECTOR

The screenshot shows the Xcode Instruments interface. The top bar indicates "Running AltConf1 on iPhone XR". The left sidebar shows CPU usage at 6%, Memory at 46 MB, and Disk at Zero KB/s. Thread 1 is highlighted with a yellow warning icon, showing a stack trace with the last frame being `UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]))`. The bottom section shows the assembly stack trace for Thread 1, starting with `argc = (int) 1` and `argv = (char **) 0x7ffeeb38dfa0`, leading through UIKitCore, CoreFoundation, and GraphicsServices libraries to `main` and `libdyld.dylib`.

```
1 //  
2 // main.m  
3 // AltConf1  
4 //  
5 // Created by Liviu Romascanu on 24/05/2019.  
6 // Copyright © 2019 Applicaster. All rights reserved.  
7 //  
8  
9 #import <UIKit/UIKit.h>  
10 #import "AppDelegate.h"  
11  
12 int main(int argc, char * argv[]) {  
13     @autoreleasepool {  
14         return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));  
15     }  
16 }  
17
```

Thread 1: signal SIGABRT

```
[A] argc = (int) 1  
[A] argv = (char **) 0x7ffeeb38dfa0
```

```
0x000000010849331e -[UIControl sendAction:to:forEvent:] + 67  
0x00000001084943da -[UIControl _sendActionsForEvents:withEvent:] + 450  
0x000000010849331e -[UIControl touchesEnded:withEvent:] + 583  
0x0000000108a940a4 -[UIWindow _sendTouchesForEvent:] + 2729  
0x0000000108a957a0 -[UIWindow sendEvent:] + 4080  
0x0000000108a73394 -[UIApplication sendEvent:] + 352  
0x0000000108b485a9 __dispatchPreprocessedEventFromEventQueue + 3054  
0x0000000108b4b1cb __handleEventQueueInternal + 5948  
0x0000000105b55721  
__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 17  
0x0000000105b54f93 __CFRunLoopDoSources0 + 243  
0x0000000105b4f63f __CFRunLoopRun + 1263  
0x0000000105b4ee11 CFRunLoopRunSpecific + 625  
0x000000010e1e81dd GSEventRunModal + 62  
0x0000000108a5781d UIApplicationMain + 140  
0x0000000104871730 main + 112  
0x0000000107564575 start + 1  
0x0000000000000001 0x0 + 1
```

# UNRECOGNIZED SELECTOR

Foundation > Collections > NSFastEnumeration > M countByEnumeratingWithState:objects:count:

Instance Method

## countByEnumeratingWithState:objects:count:

Returns by reference a C array of objects over which the sender should iterate, and as the return value the number of objects in the array.

**Required.**

**Language**  
Swift | Objective-C

**SDKs**  
iOS 2.0+  
macOS 10.5+  
tvOS 9.0+  
watchOS 2.0+

**Declaration**

```
- (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state objects:(i
```

**Parameters**

**state**  
Context information that is used in the enumeration to, in addition to other possibilities, ensure that the collection has not been mutated.

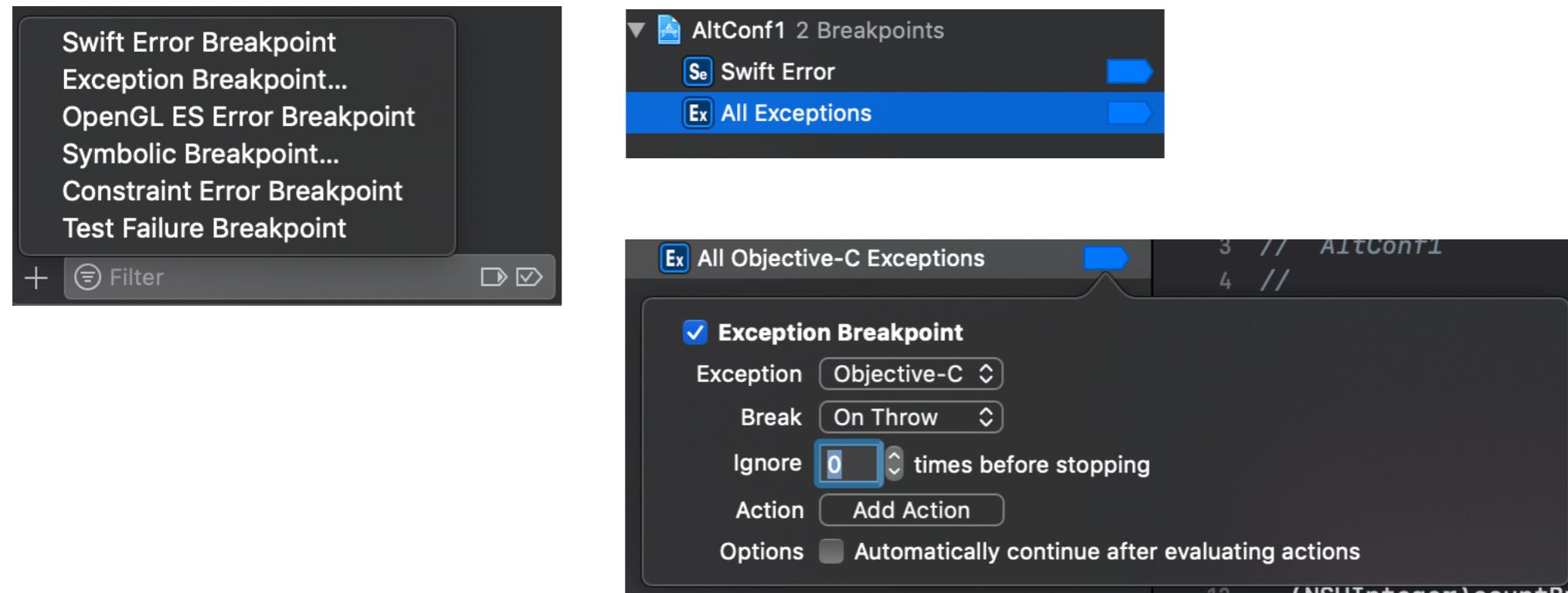
**stackbuf**  
A C array of objects over which the sender is to iterate.

**len**  
The maximum number of objects to return in stackbuf.

**Framework**  
Foundation

**On This Page**  
[Declaration](#)  
[Parameters](#)  
[Return Value](#)  
[Discussion](#)

# UNRECOGNIZED SELECTOR



# UNRECOGNIZED SELECTOR

The screenshot shows the Xcode interface during a debugging session. The top status bar indicates the project is AltConf1, the file is ViewController.m, and the method is -unrecognizedSelectorClicked:. The left sidebar shows the current process (AltConf1 PID 30439) and its resource usage (CPU 0%, Memory 46 MB, Disk Zero KB/s). The Threads tab shows Thread 1 (Queue: com....thread (serial)) with a breakpoint at line 25 of the code. The assembly and stack frames sections are visible below the threads list.

```
14
15 @implementation ViewController
16
17 - (void)viewDidLoad {
18     [super viewDidLoad];
19     // Do any additional setup after loading the view, typically from a nib.
20 }
21
22 - (IBAction)unrecognizedSelectorClicked:(id)sender {
23     NSObject *object =[NSString stringWithFormat:@"[Foo, Bar"];
24
25     for (NSString *string in ((NSArray *)object))
26     {
27         NSLog(@"%@", string);
28     }
29 }
30
31 @end
32
```

Thread 1: breakpoint 2.1

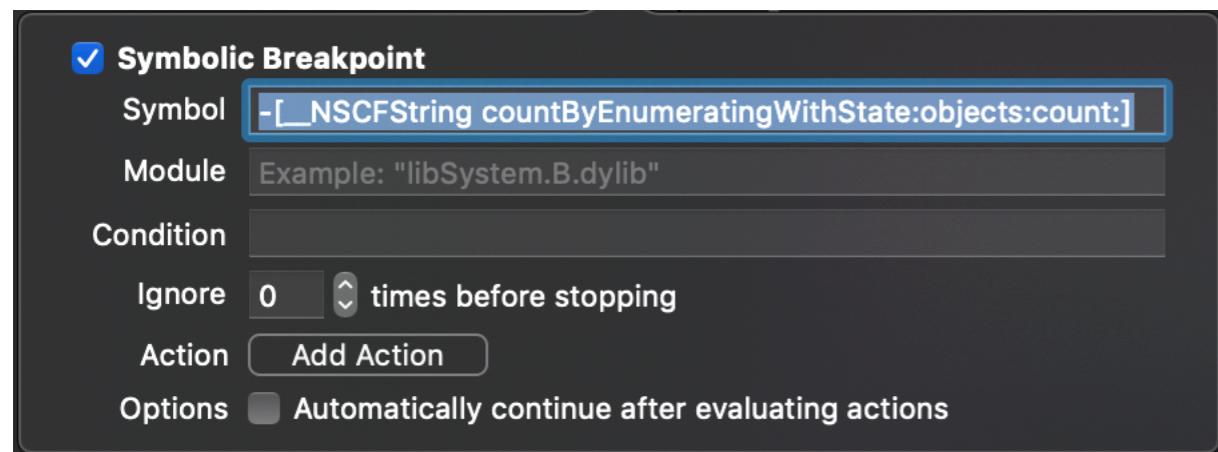
2019-05-25 20:15:52.494727-0400 AltConf1[30439:759136] -[\_\_NSCFString countByEnumeratingWithState:objects:count]: unrecognized selector sent to instance 0x600001ff4640  
(lldb)

# UNRECOGNIZED SELECTOR

CAN WE DO ANY BETTER?  
WELL , ACTUALLY - YES WE CAN!

WE CAN UTILIZE ONE OF 2 TOOLS TO ACHIEVE THE SAME RESULT:

1. SYMBOLIC BREAK POINT
2. EXTENSIONS



# UNRECOGNIZED SELECTOR

## EXTENSIONS!

```
9 #import <Foundation/Foundation.h>
10
11 NS_ASSUME_NONNULL_BEGIN
12
13 @interface NSString (Debug)
14
15 - (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state objects:(id
16   _Nullable *)buffer count:(NSUInteger)len;
17
18 @end
19 NS_ASSUME_NONNULL_END
20
```

```
9 #import "NSString+debug.h"
10
11 @implementation NSString (Debug)
12
13 - (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state objects:(id
14   _Nullable *)buffer count:(NSUInteger)len {
15   NSLog(@"We shouldn't have reached this - string value :%@", self);
16   return 0;
17
18
19 @end
```

# UNRECOGNIZED SELECTOR

The screenshot shows the Xcode interface during a debug session. On the left, the Debug Navigator displays the current process (AltConf1 PID 30858) and its resource usage (CPU 0%, Memory 46 MB, Disk Zero KB/s, Network Zero KB/s). It also lists several threads, with Thread 1 being the active one (Queue: com.apple.main-thread (serial)). The stack trace for Thread 1 shows the following call chain:

- 0 -[NSString(Debug) countByEnumeratingWithState:objects:co...]
- 1 -[ViewController unrecognizedSelectorClicked:]
- 2 -[UIApplication sendAction:to:from:forEvent:]
- 3 -[UIControl sendAction:to:forEvent:]
- 4 -[UIControl \_sendActionsForEvents:withEvent:]
- 5 -[UIControl touchesEnded:withEvent:]
- 6 -[UIWindow \_sendTouchesForEvent:]
- 7 -[UIWindow sendEvent:]
- 8 -[UIApplication sendEvent:]
- 9 \_dispatchPreprocessedEventFromEventQueue
- 10 \_handleEventQueueInternal
- 11 \_\_CFRUNLOOP\_IS\_CALLING\_OUT\_TO\_A\_SOURCE0\_PERFOR...
- 12 \_\_CFRunLoopDoSources0
- 13 \_\_CFRunLoopRun
- 14 CFRUNLoopRunSpecific
- 15 GSEventRunModal
- 16 UIApplicationMain
- 17 main
- 18 start

Following this, the stack trace continues with other threads and a UI thread.

The right side of the interface shows the source code for `NSString+debug.m`. The code is as follows:

```
1 //  
2 //  NSString+debug.m  
3 //  AltConf1  
4 //  
5 //  Created by Liviu Romascanu on 24/05/2019.  
6 //  Copyright © 2019 Applicaster. All rights reserved.  
7 //  
8  
9 #import "NSString+debug.h"  
10  
11 @implementation NSString (Debug)  
12  
13 - (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state objects:(id  
    _Nullable *)buffer count:(NSUInteger)len {  
    NSLog(@"%@", @"We shouldn't have reached this - string value :%@", self); Thread 1: breakpoint 1....  
    14    return 0;  
    15}  
16 }  
17  
18  
19 @end  
20
```

A breakpoint is set at line 14, which is highlighted in blue. The debugger status bar at the bottom indicates the current frame is at line 14, with variables `self`, `state`, `buffer`, and `len` listed.

# UNRECOGNIZED SELECTOR

---

## CONCLUSIONS

1. WE FOUND OUT HOW TO GO ABOUT UNRECOGNIZED SELECTORS
2. WE LEARNED HOW TO IMPROVE THE STACK TRACE
3. WE HAVE A TRICK HOW TO AVOID CRASHING, AND HOW TO DEBUG THE OBJECT AND GET THE FULL CONTEXT, EVEN WHEN THE CRASH IS NOT HAPPENING IN OUR CODE

# BAD MEMORY ACCESS

---

IS MY OBJECT STILL IN MEMORY?

THE NEXT FAMILY OF CRASHES WE ARE GONNA TACKLE IS BAD MEMORY ACCESS

```
NSObject *primaryObject = [[NSObject alloc] init];
NSObject __unsafe_unretained *secondaryObject = primaryObject;
primaryObject = nil;
NSLog(@"%@", secondaryObject);
```

# BAD MEMORY ACCESS

IS MY OBJECT STILL IN MEMORY?

THE NEXT FAMILY OF CRASHES WE ARE GONNA TACKLE IS BAD MEMORY ACCESS

```
Incident Identifier: E8588C36-113C-4DF6-B6CF-4FED47F39F86
CrashReporter Key: 7d7c2ea701941dad131f18a1eddf6d6d63e69615
Hardware Model: iPad6,3
Process: AltConf1 [904]
Path: /private/var/containers/Bundle/Application/69A122D1-AE84-41C6-9107-D983DE8774CF/AltConf1.app/AltConf1
Identifier: com.applicaster.AltConf1
Version: 1 (1.0)
Code Type: ARM-64 (Native)
Role: Non UI
Parent Process: launchd [1]
Coalition: com.applicaster.AltConf1 [640]

Date/Time: 2019-05-28 12:52:40.5213 -0400
Launch Time: 2019-05-28 12:52:38.4932 -0400
OS Version: iPhone OS 12.3 (16F156)
Baseband Version: n/a
Report Version: 104

Exception Type: EXC_BAD_ACCESS (SIGSEGV)
Exception Subtype: KERN_INVALID_ADDRESS at 0x0000000b99e044f0
VM Region Info: 0xb99e044f0 is not in any region. Bytes after previous region: 38551962865
REGION TYPE                      START - END              [ VSIZE] PRT/MAX SHRMOD  REGION DETAIL
MALLOC_NANO           0000000280000000-00000002a0000000  [512.0M] rw-/rwx SM=PRV
--->          UNUSED SPACE AT END

Termination Signal: Segmentation fault: 11
Termination Reason: Namespace SIGNAL, Code 0xb
Terminating Process: exc handler [904]
Triggered by Thread: 0
```

# BAD MEMORY ACCESS

LET'S GO TO THE DEBUGGER

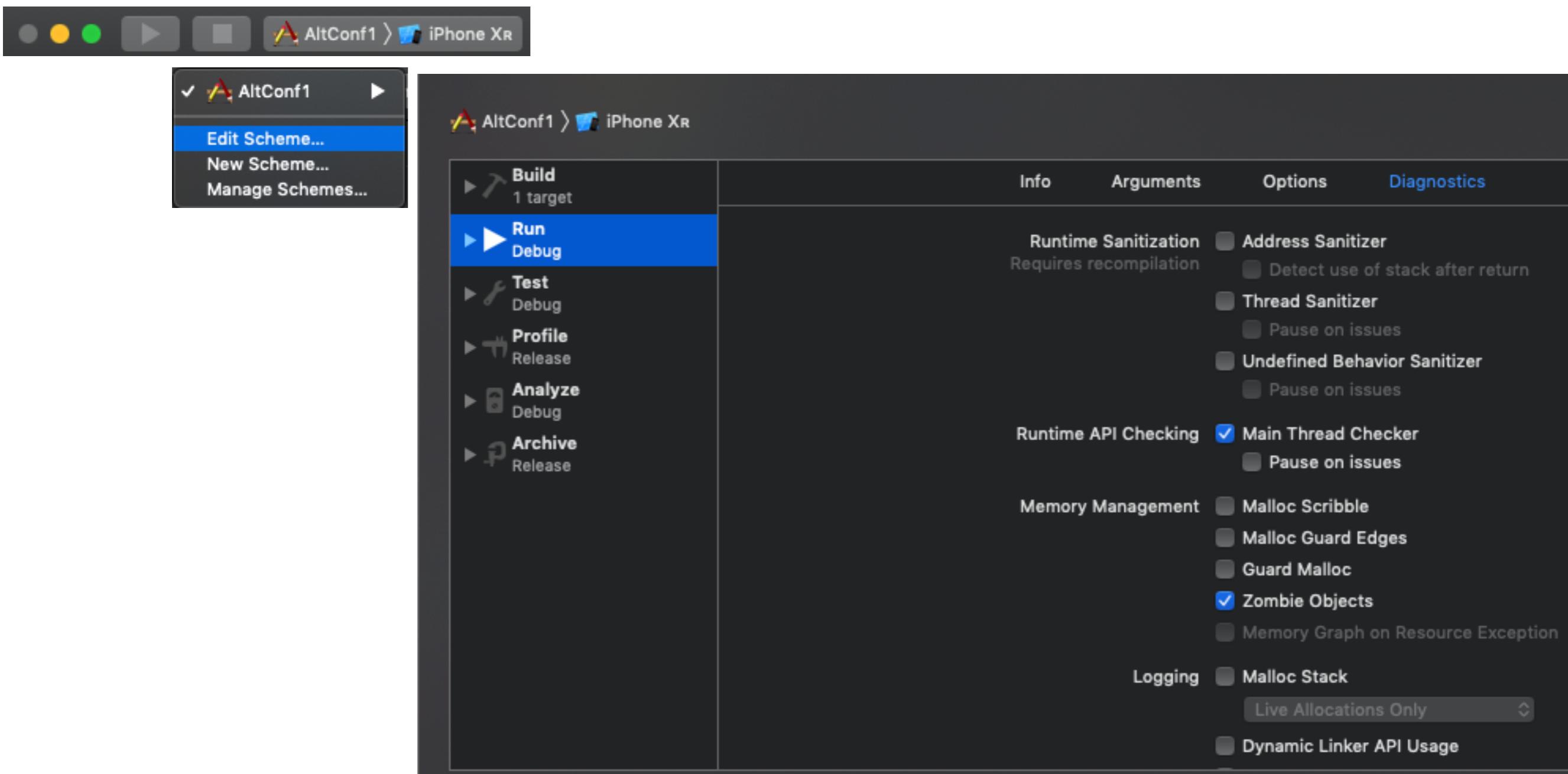
The screenshot shows the Xcode interface during a debug session. The top bar indicates "Running AltConf1 on iPhone XR". The left sidebar shows the target "AltConf1 PID 53167" and various system metrics: CPU (0%), Memory (48.1 MB), Disk (Zero KB/s), and Network (Zero KB/s). Below that is the "Thread 1" section, which is expanded to show the stack trace: 0 objc\_msgSend, 1 \_os\_log\_fmt\_flatten\_data, 2 \_os\_log\_impl\_flatten\_and\_send, 3 \_os\_log\_with\_args\_impl, 4 \_CFLLog\_os\_log\_shim, 5 \_CFLLogvEx3, 6 \_NSLogv, 7 NSLog, 8 -[ViewController badMemAccessClicked:], 9-[UIApplication sendAction:to:from:forEvent:], and 10-[UIControl sendAction:to:forEvent:]. The main editor area displays the code for the `badMemAccessClicked:` method:

```
29 }
30
31 - (IBAction)badMemAccessClicked:(id)sender {
32     NSObject *primaryObject = [[NSObject alloc] init];
33     NSObject __unsafe_unretained *secondaryObject = primaryObject;
34     primaryObject = nil;
35     NSLog(@"%@", secondaryObject); = Thread 1: EXC_BAD_ACCESS (code=1, address=0x66a2d32ac578)
36 }
37
38 @end
39
```

The line `NSLog(@"%@", secondaryObject);` is highlighted in red, indicating a runtime error. A tooltip at the bottom right of the code area says "Thread 1: EXC\_BAD\_ACCESS (code=1, address=0x66a2d32ac578)". The bottom right corner of the code area has a red border. The bottom of the screen shows the debugger's command line interface with the prompt "(lldb)" and some variable values.

# BAD MEMORY ACCESS

ZOMBIES? WHAT DO YOU MEAN ZOMBIES?!?



# BAD MEMORY ACCESS

---

ZOMBIES? WHAT DO YOU MEAN ZOMBIES?!?

A 101 ON MEMORY MANAGEMENT

- OBJECT HAS A REFERENCE COUNT PROPERTY
- WHEN REFERENCE COUNT BECOMES 0 (NO ONE POINTS TO IT) - OBJECT IS RELEASED
- WHEN TRYING TO ACCESS A RELEASED OBJECT - YOU MIGHT CRASH THE APPLICATION

YOU CAN ENCOUNTER THIS AROUND DELEGATES - EVEN IF THE PUBLIC HEADER FILE SAYS “WEAK” WHICH IS SAFE.

# BAD MEMORY ACCESS

---

ZOMBIES? WHAT DO YOU MEAN ZOMBIES?!?

THE ZOMBIE OBJECTS FLAG KEEPS THE OBJECT ALIVE IN MEMORY BUT SENDS AN ASSERTION IF A RELEASED OBJECT GETS A MESSAGE.

ENABLING THIS FLAG WILL CAUSE HIGHER MEMORY CONSUMPTION - SO DON'T ALWAYS TURN IT ON.

# BAD MEMORY ACCESS

## HOW IT LOOKS WITH ZOMBIES ENABLED

The screenshot shows the Xcode interface during a debug session. The top bar indicates "Running AltConf1 on iPhone XR". The left sidebar shows the target "AltConf1 PID 54318" and various monitoring tabs like CPU, Memory, Disk, and Network. The main area displays the code for `-badMemAccessClicked:` in `ViewController.m`. Line 35 contains the problematic code: `NSLog(@"%@", secondaryObject);`. A red highlight covers the entire line, and a tooltip below it reads "Thread 1: EXC\_BAD\_INSTRUCTION (code=EXC\_I386\_INVOP)". The bottom right corner of the code editor shows the output window with the following log entry:

```
2019-05-28 14:47:13.477359-0400
AltConf1[54318:1838992] *** -[NSObject
isProxy]: message sent to deallocated
instance 0x6000009ac6e0
(lldb)
```

# BAD MEMORY ACCESS

## OPTION #2 - ZOMBIES PROFILER

The screenshot shows the Instruments application running on an iPhone XR (12.2) with a session named "AltConf1". The timeline shows a single run from 00:00:00.000 to 00:10:00.000. A modal dialog titled "Zombie Message" is displayed, stating: "An Objective-C message was sent to a deallocated 'NSObject' object (zombie) at address: 0x6000002b0470." Below the dialog, the Allocations tab shows an allocation history for the zombie object. The table has columns: #, Event Type, Δ RefCt, RefCt, Timestamp, Responsible, and Responsible Caller. It lists two events: a Malloc/Release (2) from AltConf1 at 00:10.555.698 and a Zombie from Foundation at 00:10.555.767. The Responsible Caller for the zombie is \_NSLogv. The right panel provides detailed information about the zombie object, including its category as NSObject, type as Unknown, pointer as 0x6000002b0470, and retain delta as +0. The Stack Trace section lists the call stack: malloc\_zone\_malloc, calloc, class\_createInstance, +[NSObject allocWithZone:], -[ViewController badMemAccessClicked:], and -[UIApplication sendAction:to:from:forEvent:].

#	Event Type	Δ RefCt	RefCt	Timestamp	Responsible	Responsible Caller
▶	Malloc/Release (2)			00:10.555.698	AltConf1	-[ViewController badMemAccessClick...]
2	Zombie	-1		00:10.555.767	Foundation	_NSLogv

Retain & Release Count: +0      Unpair (2)

Description  
Category: NSObject  
Type: Unknown  
Pointer: 0x6000002b0470  
Retain Δ: +0

Stack Trace  
malloc\_zone\_malloc  
calloc  
class\_createInstance  
+[NSObject allocWithZone:]  
-[ViewController badMemAccessClicked:]  
-[UIApplication sendAction:to:from:forEvent:]

# BAD MEMORY ACCESS

---

## CONCLUSIONS

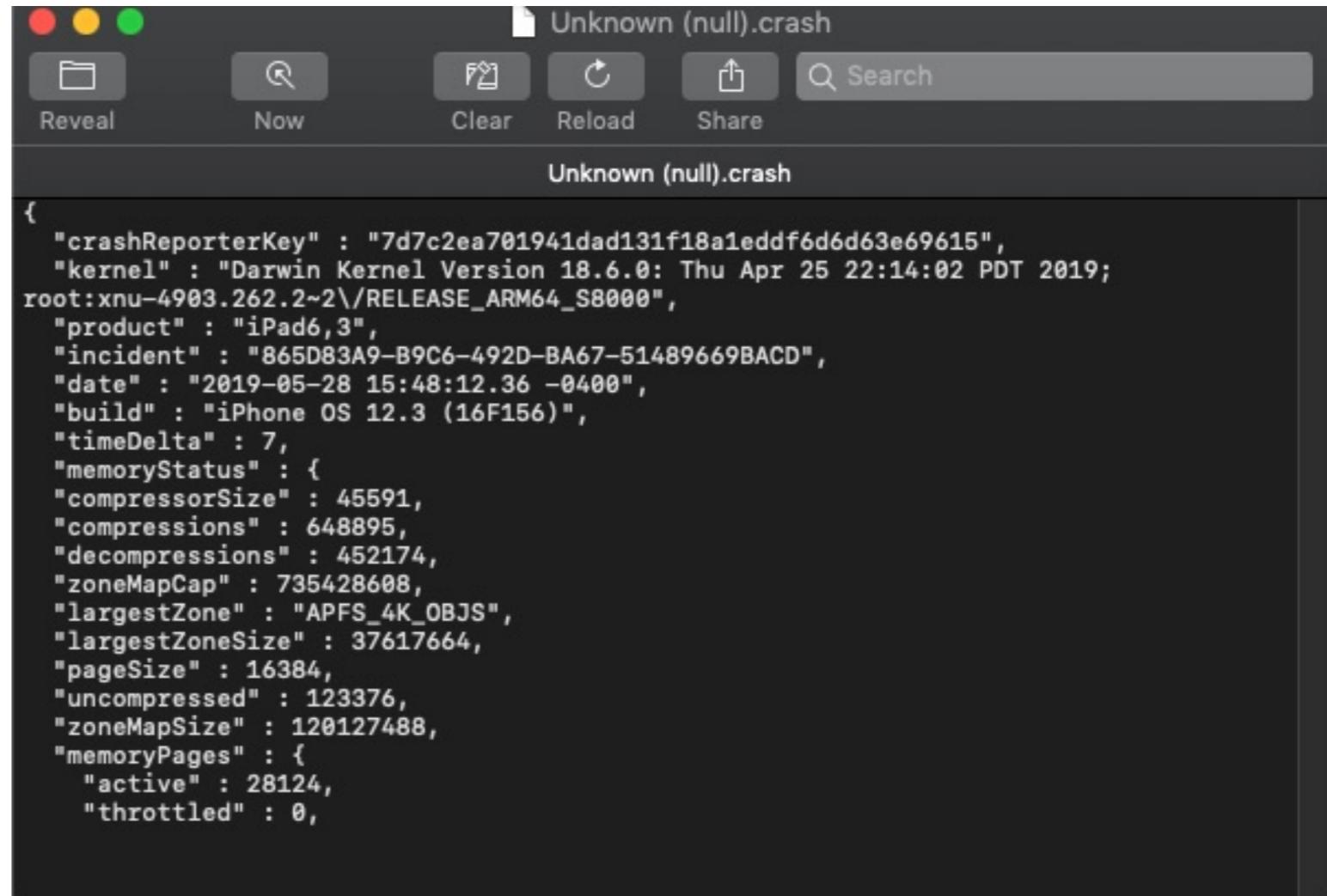
1. WE IDENTIFIED ONE VERY COMMON TYPE OF BAD MEMORY ACCESS
2. WE KNOW HOW TO VERIFY IT'S INDEED THE ISSUE WE ARE ENCOUNTERING
3. WE CAN SEE THE OBJECT TYPE, CALL, CATCH IT DETERMINISTICALLY, AND GET THE MEMORY ADDRESS

# WATCHDOG CRASHES

## WAIT - WHERE IS MY CRASH LOG?

THE LAST FAMILY OF CRASHES WE WILL DISCUSS ARE CRASHES WHICH DON'T CREATE THE CLASSIC CRASH LOG

```
NSMutableArray *imagesArray = [NSMutableArray new];
for (NSInteger i = 0; i >= 0; i++) {
    [imagesArray addObject:[UIImage imageNamed:@"msm"]];
}
```



# WATCHDOG CRASHES

## THE HARD PART

IS UNDERSTANDING WHAT HAPPENED - ESPECIALLY WHEN THE CRASH DOESN'T REPRODUCE IN THE SIMULATOR IN MANY CASES.

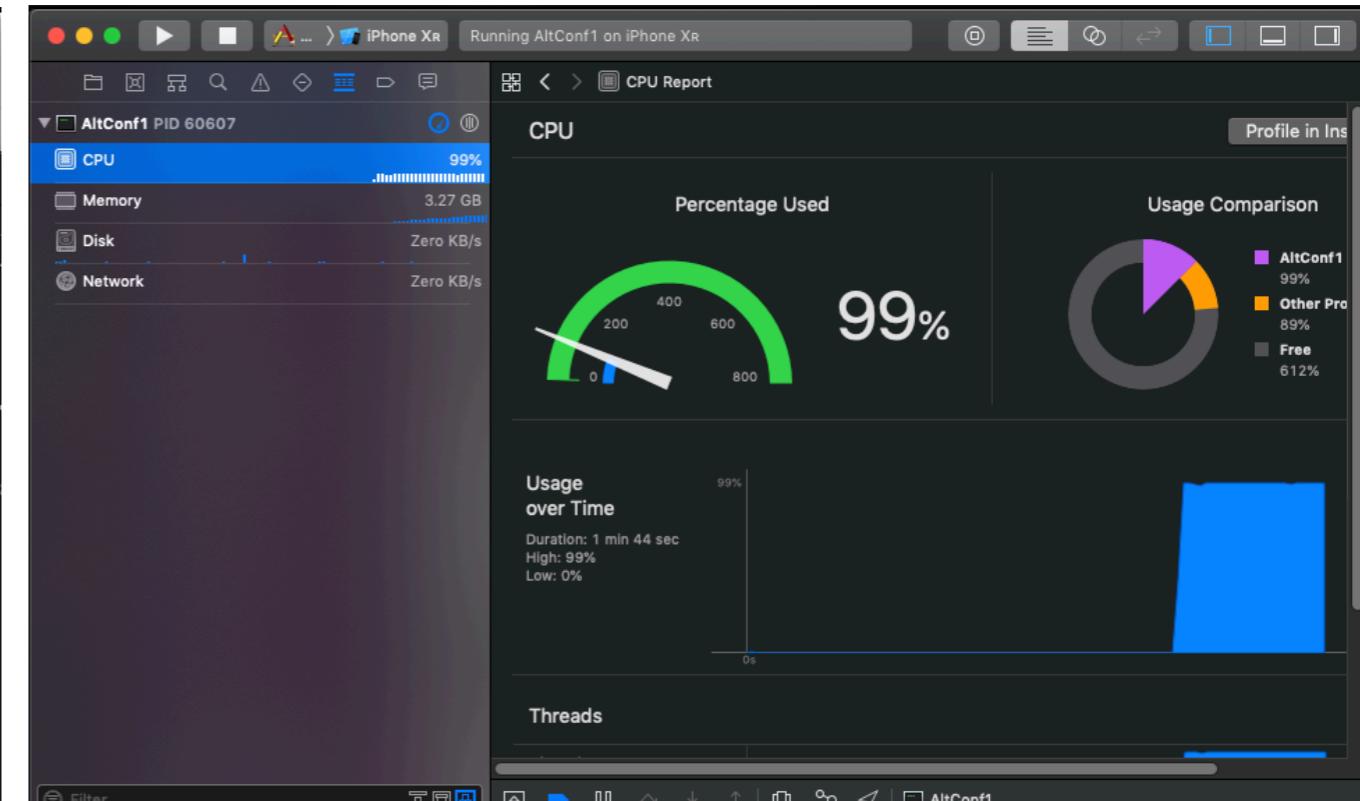
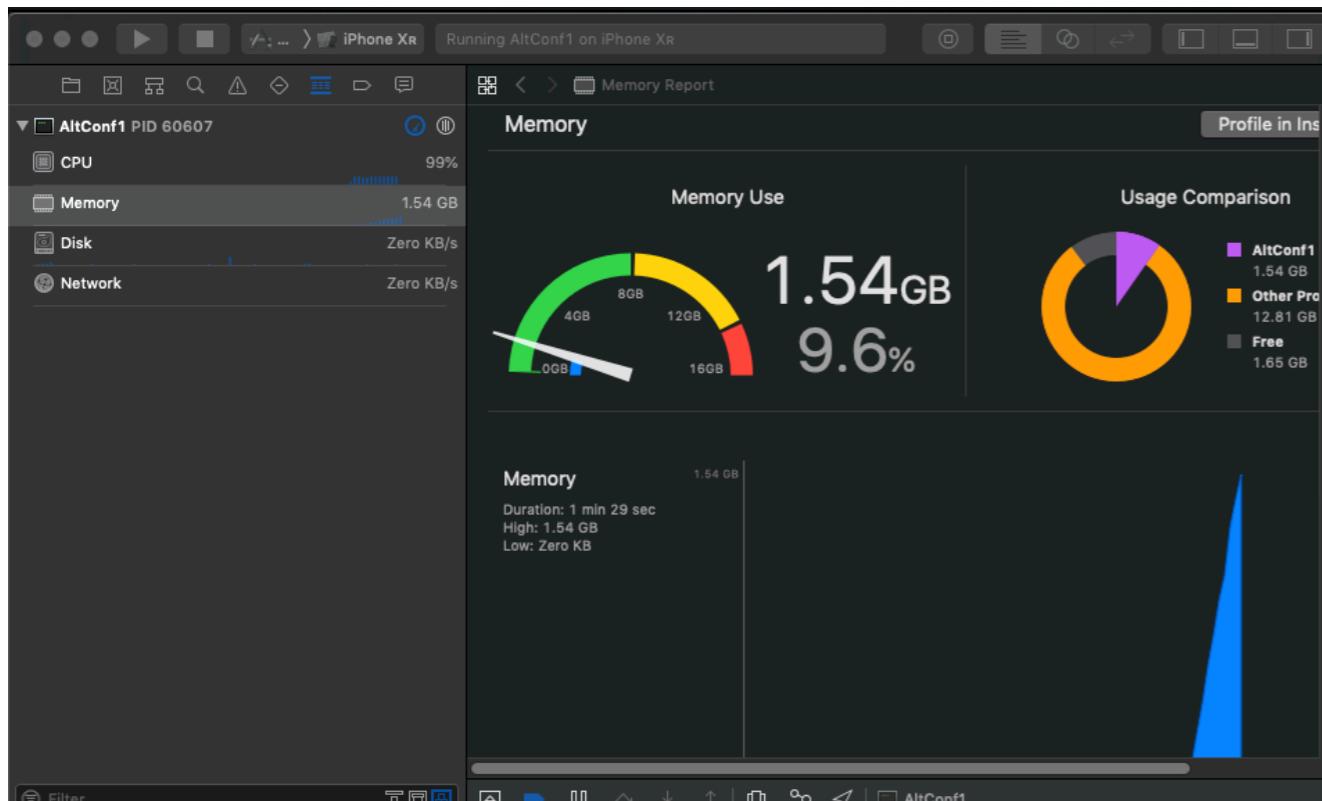
YOUR MAC HAS GREATER RESOURCES AND DIFFERENT OS WATCHDOG BEHAVIOR.

## XCODE TO THE RESCUE

XCODE ITSELF OFFERS A REALLY NICE VIEW

LET'S REVIEW TWO SCENARIOS:

1. MEMORY ALLOCATIONS
2. CPU USAGE FOR EXAMPLE

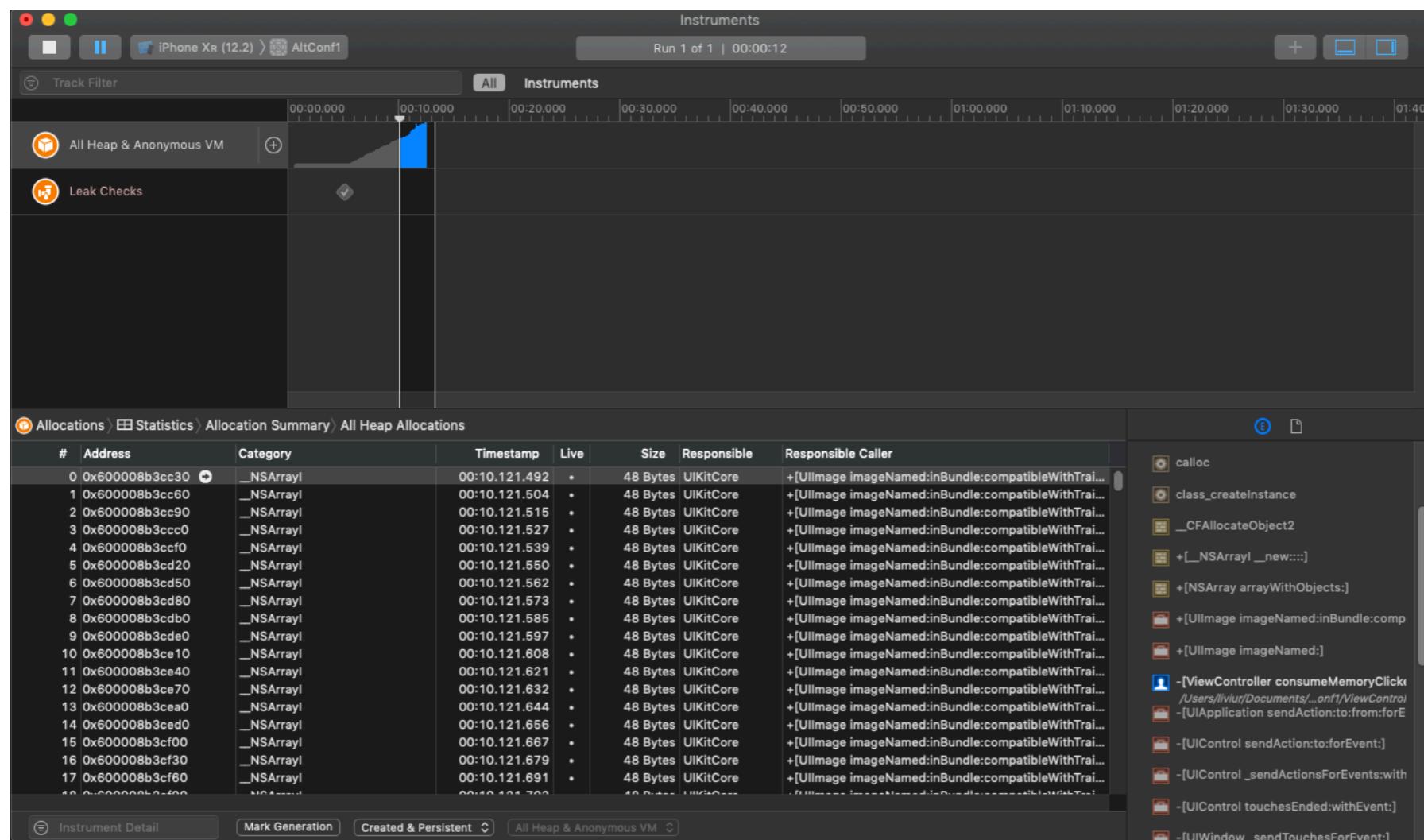


# WATCHDOG CRASHES

TIME TO 1 UP IT

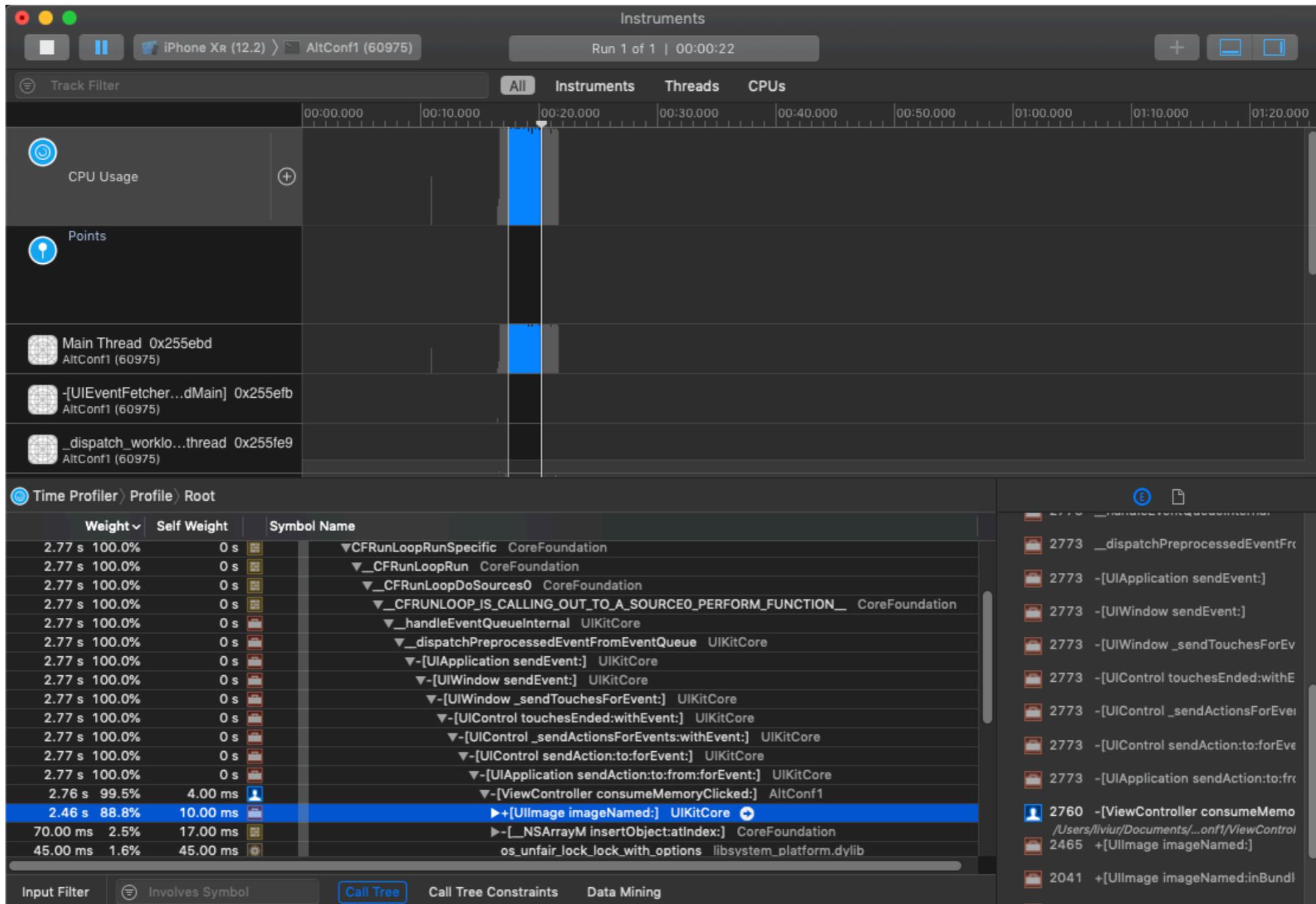
CLICKING PROFILE IN INSTRUMENTS BUTTON ALLOWS YOU TO JUMP DIRECTLY INTO THE RESPECTIVE INSTRUMENTS AND EVEN TRANSFER THE CURRENT DEBUG SESSION IN MOST CASES.

LET'S START WITH THE LEAKS AND ALLOCATIONS INSTRUMENT.



# WATCHDOG CRASHES

## CPU INSTRUMENT



# WATCHDOG CRASHES

---

## CONCLUSIONS

- THESE KIND OF CRASHES ARE A BIT TRICKIER - THEY DON'T ALWAYS GENERATE CRASH LOGS ATTRIBUTED TO THE APPLICATION - EVEN IF THE APPLICATION ITSELF CAUSES THE CRASH
- USE A DEVICE TO REPRODUCE WATCHDOG CRASHES
- FOLLOW THE XCODE DEBUG SESSION PARAMETERS, LIKE CPU AND MEMORY, AND GET COMFORTABLE WITH INSTRUMENTS

# BEST PRACTICES

---

1. USE THE GENERAL BREAK POINTS TO CATCH THINGS EARLY
2. USE SWIFT - WITH A PROPER USE OF OPTIONALS. RARELY DO YOU NEED TO USE FORCE UNWRAP.
3. GET FAMILIAR WITH INSTRUMENTS - THE WWDC INTRO SESSIONS AND WHAT'S NEW ON INSTRUMENTS AND LLDB ARE GOLD MINES - WATCH AT LEAST THE CURRENT / LAST YEAR'S SESSIONS.

UNDERSTANDING CRASH LOGS WWDC 2014 TALK:

[HTTPS://DEVELOPER.APPLE.COM/VIDEOS/PLAY/WWDC2018/414/](https://developer.apple.com/videos/play/wwdc2018/414/)

UNDERSTANDING CRASHES AND CRASH LOGS WWDC 2018 TALK:

[HTTPS://DEVELOPER.APPLE.COM/VIDEOS/PLAY/WWDC2018/414/](https://developer.apple.com/videos/play/wwdc2018/414/)

APPLE TECH NOTE (VERY DETAILED - YET NO LONGER OFFICIALLY UPDATED):

[HTTPS://DEVELOPER.APPLE.COM/LIBRARY/ARCHIVE/TECHNOTES/TN2151/\\_INDEX.HTML](https://developer.apple.com/library/archive/technotes/tn2151/_index.html)

SAMPLE APP AND SLIDES:

# CRASH LOG ANATOMY

1. CONTAINS INFO SUCH AS HARDWARE, PROCESS THAT CAUSED THE CRASH , BUNDLE IDENTIFIER, VERSION.
2. CONTAINS THE TIME (LAUNCH AND CRASH) AND THE OS VERSION
3. CONTAINS THE EXCEPTION TYPE, MEMORY ADDRESS RELATED TO THE CRASH, AND THE TERMINATION REASONS. LAST LINE OF THIS SEGMENT CONTAINS THE THREAD - 0 REPRESENTS MAIN THREAD.
4. CONTAINS THE STACK TRACE ON ACTIVE THREADS (LAST COMMANDS CAPTURED AROUND THE CRASH TIME).

```
Incident Identifier: FF19AED4-2677-44D1-A992-91795BC3980A
CrashReporter Key: 7d7c2ea701941dad131f18a1eddf6d6d63e69615
Hardware Model: iPad6,3
Process: AltConf1 [901]
Path: /private/var/containers/Bundle/Application/69A122D1-AE84-41C6-9107-D983DE8774CF/
AltConf1.app/AltConf1
Identifier: com.applicaster.AltConf1
Version: 1 (1.0)
Code Type: ARM-64 (Native)
Role: Non UI
Parent Process: launchd [1]
Coalition: com.applicaster.AltConf1 [638]

Date/Time: 2019-05-28 12:51:11.1237 -0400
Launch Time: 2019-05-28 12:51:09.4061 -0400
OS Version: iPhone OS 12.3 (16F156)
Baseband Version: n/a
Report Version: 104

Exception Type: EXC_BAD_ACCESS (SIGSEGV)
Exception Subtype: KERN_INVALID_ADDRESS at 0x0000000447db77f0
VM Region Info: 0x447db77f0 is not in any region. Bytes after previous region: 7111145457
    REGION TYPE            START - END          [ VSIZE] PRT/MAX SHRMOD  REGION DETAIL
    MALLOC_NANO           0000000280000000-00000002a0000000 [512.0M] rw-/rwx SM=PRV
-->      UNUSED SPACE AT END

Termination Signal: Segmentation fault: 11
Termination Reason: Namespace SIGNAL, Code 0xb
Terminating Process: exc handler [901]
Triggered by Thread: 0

Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0   libobjc.A.dylib                  0x0000000197682530 objc_msgSend + 16
1   libsystem_trace.dylib            0x00000001980c7fff _os_log_fmt_flatten_data + 136
2   libsystem_trace.dylib            0x00000001980ce7c8 _os_log_impl_flatten_and_send + 1632
3   libsystem_trace.dylib            0x00000001980d0d7c _os_log_with_args_impl + 396
4   CoreFoundation                  0x000000019848087c _CFLLog_os_log_shim + 68
5   CoreFoundation                  0x00000001984807b8 _CFLLogvEx3 + 128
6   Foundation                      0x0000000198efc63c _NSLogv + 124
7   Foundation                      0x0000000198efc670 NSLog + 32
8   AltConf1                         0x00000001005a6684-[ViewController badMemAccessClicked:] + 26244
(ViewController.m:36)
9   UIKitCore                        0x00000001c4795040 -[UIApplication sendAction:to:from:forEvent:] + 96
10  UIKitCore                        0x00000001c423e1c8 -[UIControl sendAction:to:forEvent:] + 80
11  UIKitCore                        0x00000001c423e4e8 -[UIControl _sendActionsForEvents:withEvent:] + 440
12  UIKitCore                        0x00000001c423d554 -[UIControl touchesEnded:withEvent:] + 568
13  UIKitCore                        0x00000001c47cc304 -[UIWindow _sendTouchesForEvent:] + 2108
14  UIKitCore                        0x00000001c47cd52c -[UIWindow sendEvent:] + 3140
15  UIKitCore                        0x00000001c47ad59c -[UIApplication sendEvent:] + 340
16  UIKitCore                        0x00000001c4873714 __dispatchPreprocessedEventFromEventQueue + 1768
17  UIKitCore                        0x00000001c4875e40 __handleEventQueueInternal + 4828
18  UIKitCore                        0x00000001c486f070 __handleHIDEEventFetcherDrain + 152
19  CoreFoundation                   0x0000000198422018
__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 24
20  CoreFoundation                   0x0000000198421f98 __CFRunLoopDoSource0 + 88
21  CoreFoundation                   0x0000000198421880 __CFRunLoopDoSources0 + 176
22  CoreFoundation                   0x000000019841c7bc __CFRunLoopRun + 1004
23  CoreFoundation                   0x000000019841c0b0 CFRunLoopRunSpecific + 436
```

# CRASH LOG ANATOMY

## WHAT TO DO IF I DON'T SEE THE SYMBOLS?

SOMETIMES CRASH LOGS COME WITHOUT ALL THE SYMBOLS - BUT THERE'S A SOLUTION ... SOMETIMES.

```
Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0 libobjc.A.dylib          0x0000000197682530 0x197665000 + 120112
1 libsystem_trace.dylib    0x00000001980c7ff8 0x1980c0000 + 32760
2 libsystem_trace.dylib    0x00000001980ce7c8 0x1980c0000 + 59336
3 libsystem_trace.dylib    0x00000001980d0d7c 0x1980c0000 + 68988
4 CoreFoundation           0x000000019848087c 0x198378000 + 1083516
5 CoreFoundation           0x00000001984807b8 0x198378000 + 1083320
6 Foundation               0x0000000198efc63c 0x198de2000 + 1156668
7 Foundation               0x0000000198efc670 0x198de2000 + 1156720
8 AltConf1                 0x00000001045f6684 0x1045f0000 + 26244
9 UIKitCore                0x00000001c4795040 0x1c3ed7000 + 9166912
10 UIKitCore               0x00000001c423e1c8 0x1c3ed7000 + 3568072
11 UIKitCore               0x00000001c423e4e8 0x1c3ed7000 + 3568872
12 UIKitCore               0x00000001c423d554 0x1c3ed7000 + 3564884
13 UIKitCore               0x00000001c47cc304 0x1c3ed7000 + 9392900
14 UIKitCore               0x00000001c47cd52c 0x1c3ed7000 + 9397548
15 UIKitCore               0x00000001c47ad59c 0x1c3ed7000 + 9266588
16 UIKitCore               0x00000001c4873714 0x1c3ed7000 + 10077972
17 UIKitCore               0x00000001c4875e40 0x1c3ed7000 + 10088000
18 UIKitCore               0x00000001c486f070 0x1c3ed7000 + 10059888
19 CoreFoundation           0x0000000198422018 0x198378000 + 696344
20 CoreFoundation           0x0000000198421f98 0x198378000 + 696216
21 CoreFoundation           0x0000000198421880 0x198378000 + 694400
22 CoreFoundation           0x000000019841c7bc 0x198378000 + 673724
23 CoreFoundation           0x000000019841c0b0 0x198378000 + 671920
```

```
Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0 libobjc.A.dylib          0x0000000197682530 objc_msgSend + 16
1 libsystem_trace.dylib    0x00000001980c7ff8 _os_log_fmt_flatten_data + 136
2 libsystem_trace.dylib    0x00000001980ce7c8 _os_log_impl_flatten_and_send + 1632
3 libsystem_trace.dylib    0x00000001980d0d7c _os_log_with_argsImpl + 396
4 CoreFoundation           0x000000019848087c _CFLLog_os_log_shim + 68
5 CoreFoundation           0x00000001984807b8 _CFLLogvEx3 + 128
6 Foundation               0x0000000198efc63c _NSLogv + 124
7 Foundation               0x0000000198efc670 NSLog + 32
8 AltConf1                 0x00000001005a6684-[ViewController badMemAccessClicked:] + 26244
(ViewController.m:36)
9 UIKitCore                0x00000001c4795040-[UIApplication sendAction:to:from:forEvent:] + 96
10 UIKitCore               0x00000001c423e1c8-[UIControl sendAction:to:forEvent:] + 80
11 UIKitCore               0x00000001c423e4e8-[UIControl _sendActionsForEvents:withEvent:] + 440
12 UIKitCore               0x00000001c423d554-[UIControl touchesEnded:withEvent:] + 568
13 UIKitCore               0x00000001c47cc304-[UIWindow _sendTouchesForEvent:] + 2108
14 UIKitCore               0x00000001c47cd52c-[UIWindow sendEvent:] + 3140
15 UIKitCore               0x00000001c47ad59c-[UIApplication sendEvent:] + 340
16 UIKitCore               0x00000001c4873714__dispatchPreprocessedEventFromEventQueue + 1768
17 UIKitCore               0x00000001c4875e40__handleEventQueueInternal + 4828
18 UIKitCore               0x00000001c486f070__handleHIDEEventFetcherDrain + 152
19 CoreFoundation           0x0000000198422018__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 24
20 CoreFoundation           0x0000000198421f98 __CFRunLoopDoSource0 + 88
21 CoreFoundation           0x0000000198421880 __CFRunLoopDoSources0 + 176
22 CoreFoundation           0x000000019841c7bc __CFRunLoopRun + 1004
23 CoreFoundation           0x000000019841c0b0 __CFRunLoopRunSpecific + 436
```

# CRASH LOG ANATOMY

LET'S GET TO IT:

1. EXPORT THE XCODE DEVELOPER DIRECTORY TO “DEVELOPER\_DIR” ENV VARIABLE:

```
export DEVELOPER_DIR=/Applications/Xcode.app/Contents/Developer
```

2. FOR SIMPLICITY'S SAKE - PLACE THE CRASH LOG AND THE BINARY / DSYM IN THE SAME DIRECTORY

3. LOCATE YOUR SYMBOLICATE CRASHLOG UTILITY. ITS USUAL LOCATION IS:

```
/Applications/Xcode.app/Contents/SharedFrameworks/DVTFoundation.framework/Versions/A/Resources/symbolicatecrash
```

4. RUN THE SYMBOLICATE CRASHLOG UTILITY:

```
<Symbolicate crashlog path>/symbolicatecrash <log.crash> <binary.app> -o <symbolicated.crash>
```

PARAMETERS ARE:

1. UNSYMBOLICATED CRASH LOG
2. APP PACKAGE (BINARY.APP) IN CASE DWARF WAS USED AND IT'S A RICH BINARY OR DSYM DIRECTORY (BINARY.APP/DSYM)
3. OUTPUT FILE AFTER “-O” PARAMETER, OR ALTERNATIVELY YOU CAN FUNNEL (>) THE OUTPUT INTO A CRASH FILE.

NOTICE - YOU CAN ADD ADDITIONAL DSYM FILES IF YOU USE DEVELOPMENT BUILT DYNAMIC FRAMEWORKS WITH A “-D” PARAMETER

# COLLECTING CRASH LOGS

---

## HOW TO GRAB CRASH REPORTS

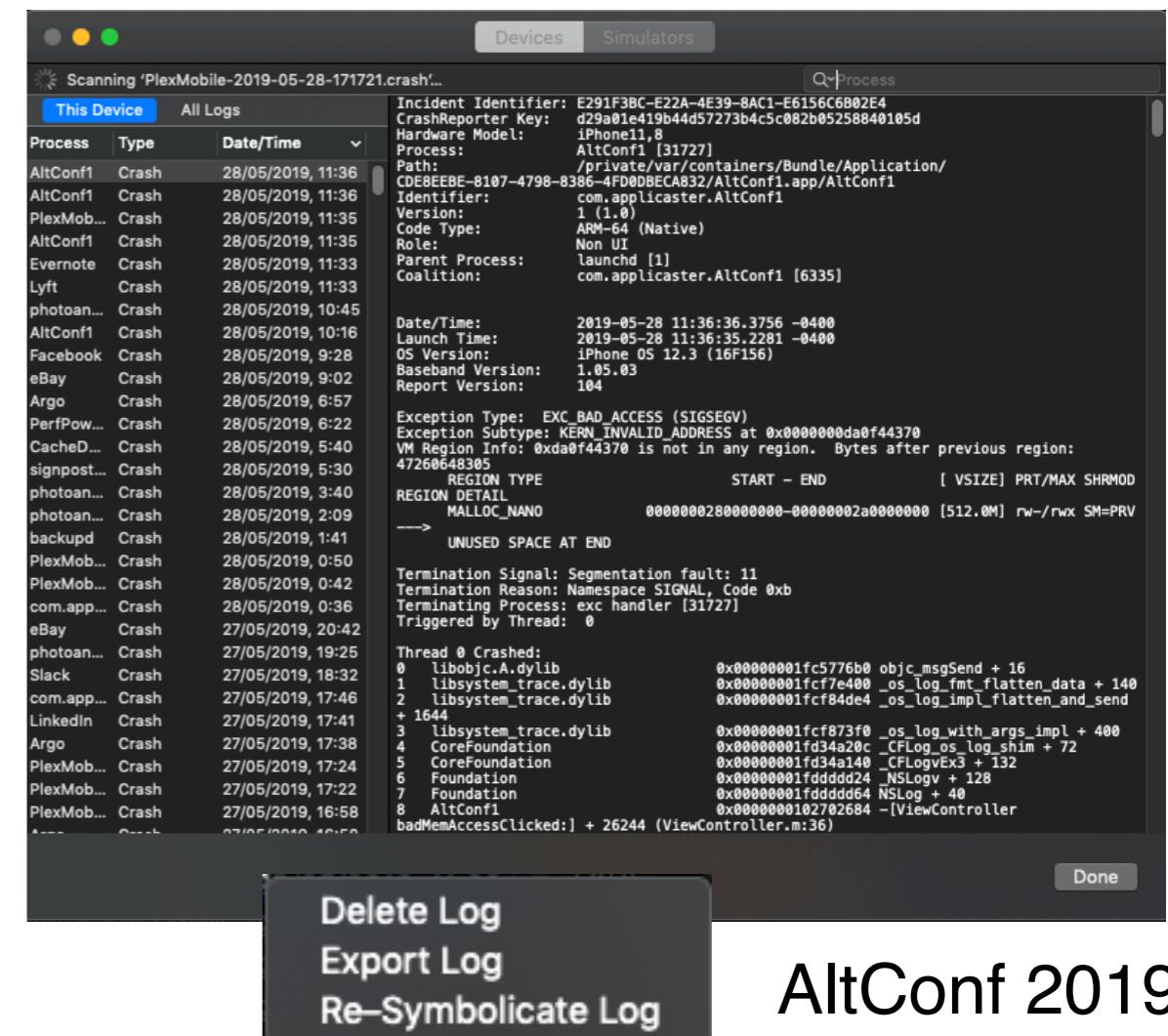
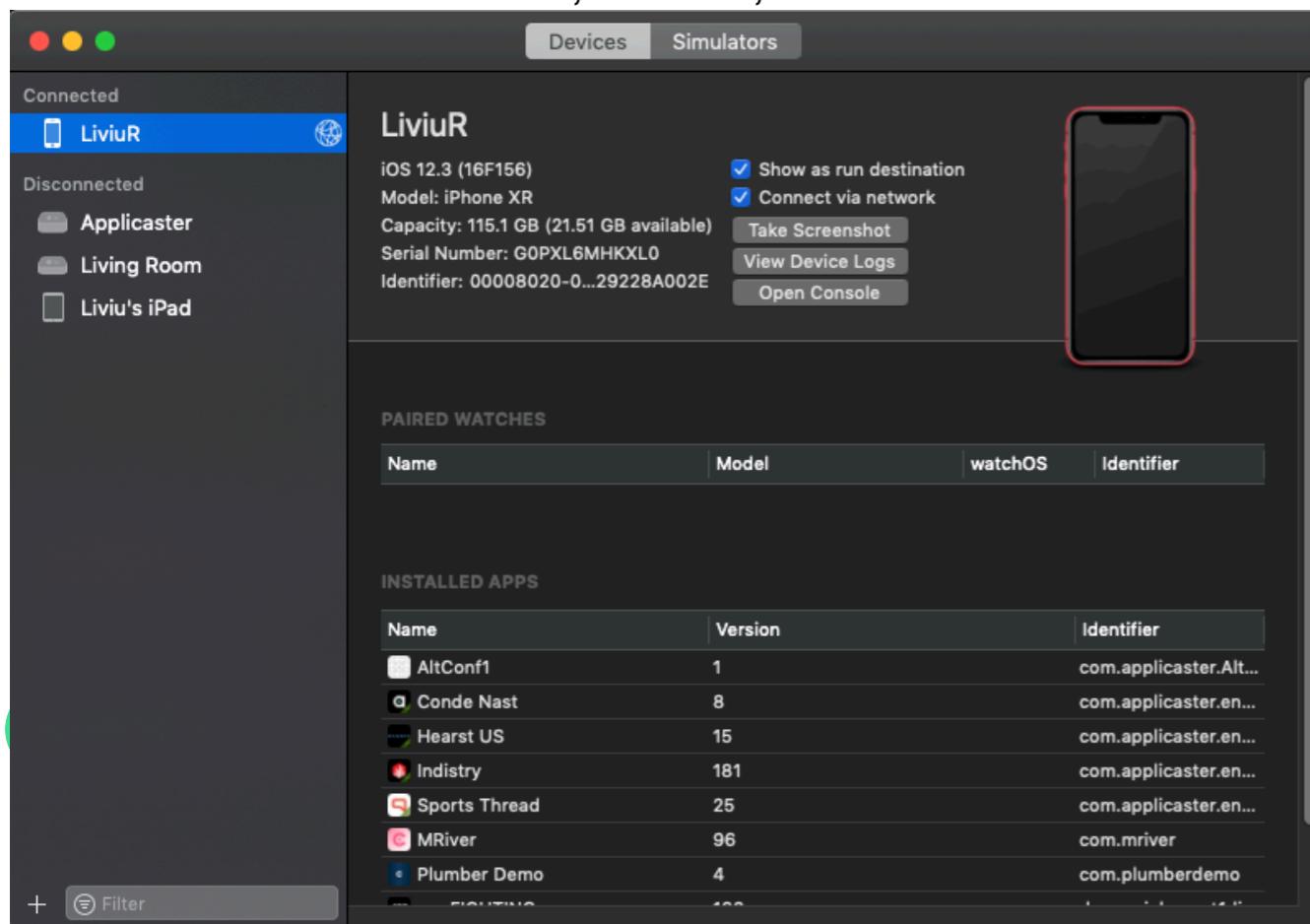
### 3 MAIN METHODS:

- GRAB FROM DEVICE WITH XCODE
- GET FROM ITUNES CONNECT WITH XCODE
- USE CRASH LOG COLLECTING SDK

# COLLECTING CRASH LOGS

## GRABBING CRASHES FROM A DEVICE

1. CONNECT THE DEVICE VIA USB
2. UNLOCK THE DEVICE
3. OPEN XCODE
4. WINDOWS -> DEVICES AND SIMULATORS / CMD-SHIFT-2
5. CHOOSE DEVICE
6. CLICK VIEW DEVICE LOGS. YOU MIGHT NEED TO WAIT FOR DEVICE LOGS TO SYNC
7. RIGHT CLICK TO DELETE, EXPORT, OR RE-SYMBOLICATE LOG

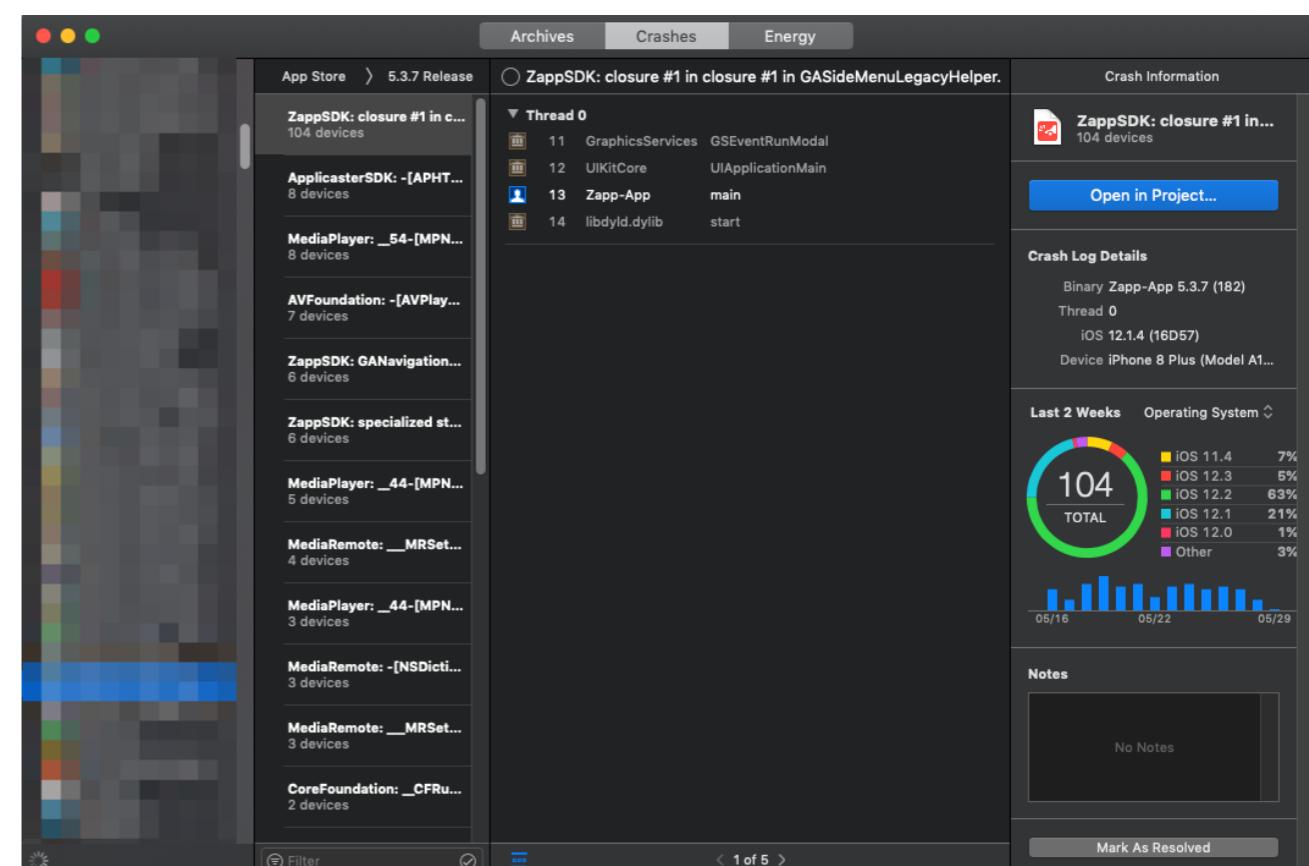
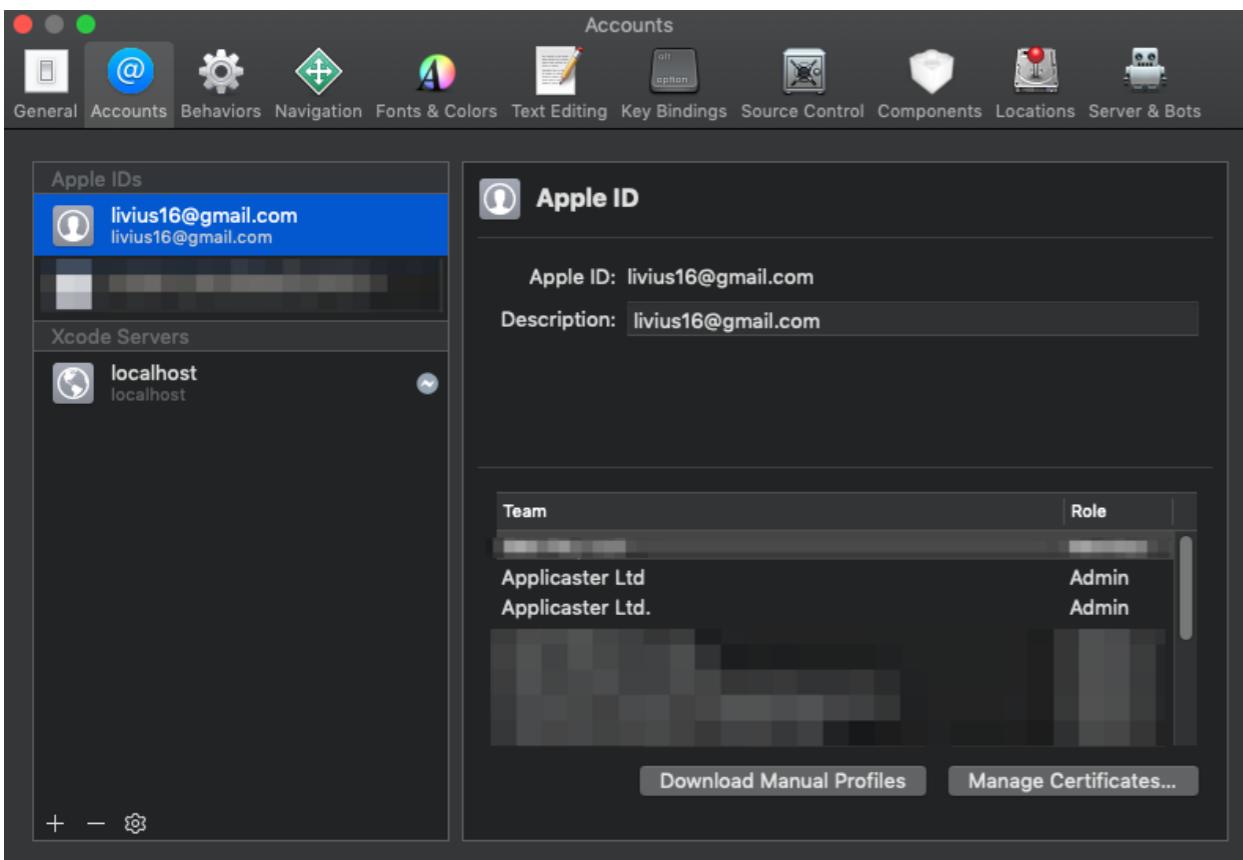


# COLLECTING CRASH LOGS

## HOW TO GRAB CRASH LOGS FROM ITUNES CONNECT THROUGH XCODE

1. OPEN XCODE
2. XCODE -> PREFERENCES -> ACCOUNTS -
  - MAKE SURE YOUR DEVELOPER / ITUNES CONNECT ACCOUNTS ALL EXIST AND ARE LOGGED IN.
3. WINDOWS -> ORGANIZER
4. CHOOSE YOUR APP ON THE LEFT PANE -> CRASHES

YOU'LL GET STACK TRACE INFO, USUALLY 5 LOGS PER CRASH, SOME INFO ABOUT THE AFFECTED DEVICES, AND LIMITED STATISTICS



# COLLECTING CRASH LOGS

---

## USING THIRD PARTY CRASH COLLECTING SERVICES

IF YOU'RE WORKING ON LARGE SCALE APPS - OR WANT TO MONITOR CRASH LOGS DAILY - YOU PROBABLY NEED A CRASH REPORTING SDK AND SERVICE.

THERE ARE A FEW OF COMPANIES GIVING THIS SERVICE IN THE MARKET WITH VARIOUS INTEGRATION LEVELS.

A FEW EXAMPLES IN THE MARKET:

1. HOCKEYAPP (MIGRATING TO VISUAL STUDIO APP CENTER - MICROSOFT)
2. CRASHLYTICS (NOW FIREBASE CRASHLYTICS - GOOGLE)
3. INSTABUG

MOST OF THEM REQUIRE YOU TO INTEGRATE AN SDK INTO THE APP, SET UP WITH AN IDENTIFIER, AND UPLOAD THE “.APP” AND “.DSYM” IN ORDER TO GET SYMBOLICATED CRASH LOGS.

USING CRASH LOG SERVICES ALSO GIVES THE BENEFIT OF DAILY-mails, FULL HISTORY, DOWNLOADING FULL CRASH LOGS, AND ADVANCED STATISTICS.

# SO WHAT WE TALKED ABOUT TODAY

---

## COMMON CRASHES

A SHORT LIST OF COMMON CRASHES WE ENCOUNTER AND HANDLE, INCLUDING DEBUG TRICKS TO HELP CATCH THOSE COMMON CRASHES

## CRASH LOG ANATOMY

HOW DOES A CRASH LOG LOOK LIKE, READING THROUGH IT QUICKLY, AND ADDING SYMBOLS IN CASE THEY AREN'T SYMBOLICATING AUTOMATICALLY

## COLLECTING CRASH LOGS

HOW TO COLLECT CRASH LOGS LOCALLY, FROM ITUNES CONNECT, AND FROM CRASH LOG SERVICES

ALL THE CODE EXAMPLES AND SLIDES ARE AVAILABLE

[HTTP://WWW.GITHUB.COM/APPLICASTER/ALTCNF2019](http://www.github.com/appcaster/altconf2019)

# Questions ?

@livius16

# We're always hiring...

<https://www.appicaster.com/careers>

# Partner?

If you have a product relevant to media apps:  
[l.romasca@appicaster.com](mailto:l.romasca@appicaster.com)

