Thread Sanitizer and Static Analysis

Help with finding bugs in your code

Session 412

Anna Zaks Manager, Program Analysis Team
Devin Coughlin Engineer, Program Analysis Team
What Is This Talk About?

Runtime Sanitizers
Static Analyzer
Runtime Sanitizers
Sanitizers

Find bugs at run time
Similar to Valgrind
Low overhead
Work with Swift 3 and C/C++/Objective-C
Integrated into Xcode IDE
Address Sanitizer (ASan)

Introduced last year
Finds memory corruption issues
Effective at finding critical bugs
Now has full support for Swift
Threading Issues

Hard to consistently reproduce
Difficult to debug
Lead to unpredictable results
Thread Sanitizer (TSan)
Thread Sanitizer (TSan)

- Use of uninitialized mutexes
- Thread leaks (missing `pthread_join`)
- Unsafe calls in signal handlers (ex: `malloc`)
- Unlock from wrong thread
- Data races
Demo
Thread Sanitizer in Xcode
Thread Sanitizer (TSan) in Xcode

1. Edit Scheme – Diagnostics tab
2. “Enable Thread Sanitizer” checkbox
3. Build and Run
4. View all of the generated runtime issues
5. Can choose to break on every issue
TSan Build Flow

clang

swift

TSan dylib

exec
Usage from Command-Line

Compile and link with TSan

```
$ clang -fsanitize=thread source.c -o executable
$ swiftc -sanitize=thread source.swift -o executable
$ xcodebuild -enableThreadSanitizer YES
```

Stop after the first error

```
$ TSAN_OPTIONS=halt_on_error=1 ./executable
```
Platform Support for TSan

64-bit macOS
64-bit iOS and tvOS simulators

No device support
No watchOS support
Fixing Data Races
Data Race

Multiple threads access the same memory without synchronization
At least one access is a write
May end up with any value or even memory corruption!
Reasons for Data Races

Can indicate a problem with the structure of your program
Often means that synchronization is missing
Data Race Example

```swift
var data: Int? = nil

func producer() {
    // More code here.
    data = 42
}
```

```swift
func consumer() {
    // More code here.
    print(data)
}
```
Data Race Example

```swift
var data: Int? = nil

func producer() {
    // More code here.
    data = 42
    dataIsAvailable = true
}

func consumer() {
    // More code here.
    while !dataIsAvailable {
        usleep(1000)
    }
    print(data)
}
```

Order is not guaranteed
Data Race Example

```swift
var data: Int? = nil

func producer() {
    // More code here.
    serialDispatchQueue.async {
        data = 42
    }
}

func consumer() {
    // More code here.
    serialDispatchQueue.sync {
        print(data)
    }
}
```
Data Race in Lazy Initialization Code

Both threads could be setting the value at the same time

```
Singleton *getSingleton() {
    static Singleton *sharedInstance = nil;

    if (sharedInstance == nil) {
        sharedInstance = [[Singleton alloc] init];
    }

    return sharedInstance;
}
```
Data Race in Lazy Initialization Code

Singleton *getSingleton() {
    static Singleton *sharedInstance = nil;

    if (sharedInstance == nil) {
        Singleton *localObject = [[Singleton alloc] init];

        // Only assign if sharedInstance is still nil.
        atomic_compare_and_set(&sharedInstance, nil, localObject);
    }

    return sharedInstance;
}
Data Race in Lazy Initialization Code

```c
Singleton *getSingleton() {
    static dispatch_once_t predicate;
    static Singleton *sharedInstance = nil;

    dispatch_once(&predicate, ^{
        sharedInstance = [[self alloc] init];
    });

    return sharedInstance;
}
```
Lazy Initialization in Swift

Global variables

```swift
var sharedInstance = Singleton()

func getSingleton() -> Singleton {
    return sharedInstance
}
```

Class constants

```swift
class Singleton {
    static let sharedInstance = Singleton()
}
```
Choosing the Right Synchronization API

1. Use GCD
   Dispatch racy accesses to the same serial queue

2. Use pthread API, NSLock
   `pthread_mutex_lock()` to synchronize accesses

3. New `os_unfair_lock` (use instead of `OSSpinLock`)

4. Atomic operations
There is No Such Thing as a “Benign” Race

On some architectures (ex., x86) reads and writes are atomic
But even a “benign” race is *undefined behavior* in C
May cause issues with new compilers or architectures
FIX ALL THE BUGS!
Behind the Scenes
For every access:

- Records the information about that access
- Checks if that access participates in a race

```
*p = 'a';
RecordAndCheckWrite(p);
*p = 'a';
```
Thread 1
Thread 2
Thread 3
Thread 4

Application memory: 8 bytes

Shadow state:
Up to four 8 byte objects
Detecting a Race

Every thread stores (in thread local):

- Thread’s own timestamp
- The timestamps for other threads that establish the points of synchronization
- Timestamps are incremented on every memory access

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>th1_2</td>
<td>th2_22</td>
<td>th3_55</td>
</tr>
<tr>
<td>th2_0</td>
<td>th1_0</td>
<td>th1_0</td>
</tr>
<tr>
<td>th3_0</td>
<td>th3_0</td>
<td>th2_0</td>
</tr>
</tbody>
</table>
Detecting a Race

Thread 1 Writes

Thread 2 Writes

Thread 3 Writes

Check for Races

Detecting a Race
Thread Sanitizer

Timing does not matter
Can detect races even if they did not manifest during the particular run
The more run time coverage the better
Run all your tests with TSan!
Finding Bugs with Static Analysis
Find Bugs Without Running Code

Does not require running code (unlike sanitizers)
Great at catching hard to reproduce edge-case bugs
Supported only for C, C++, and Objective-C
Missing Localizability
Startling for Users
Demo
Clang Static Analyzer in Xcode
Clang Static Analyzer in Xcode

1. Product > Analyze or
   Product > Analyze “SingleFile.m”

2. View in Issue Navigator

3. Explore Issue Path
Find Missing Localizability

Find unlocalized user-facing string:

```
[button setTitle:@“Cancel”];
```

User-facing text should use localized string macro

Find missing localization context comment:

```
NSString *t = NSLocalizedString(@“Cancel”, nil);
```

Localized string macro should include a non-empty comment for translators
Enable Checks in Build Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyProject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Stores</td>
<td>Yes</td>
</tr>
<tr>
<td>Improper Memory Management</td>
<td>Yes - $(CLANG_ANALYZER_MALLOC)</td>
</tr>
<tr>
<td>Missing Localizability</td>
<td>Yes</td>
</tr>
<tr>
<td>Missing Localization Context Comment</td>
<td>Yes</td>
</tr>
<tr>
<td>Misuse of 'nonnull'</td>
<td>Yes</td>
</tr>
<tr>
<td>Misuse of Grand Central Dispatch</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Checking -dealloc in Manual Retain/Release
Do Not Release ‘assign’ Properties

Release of synthesized ivar in -dealloc is over-release:

```objc
@property(assign) id delegate;

- (void)dealloc {
    [_delegate release];
    // The '_delegate' ivar was synthesized for an assign property but was released in 'dealloc'
    [super dealloc];
}
```
Do Release ‘retain/copy’ Properties

Leak if no release of ivar for retain/copy property in -dealloc:

```objc
@property(assign) id delegate;
@property(copy) NSString *title;
-(void)dealloc {
    The '_title' ivar was copied by a synthesized property but not released
    [super dealloc];
}
```
Update to Automated Reference Counting
Nullability Violations
Indicate whether a value is expected to be nil:

```objc
@interface CLLocation : NSObject
@property(readonly, nonnull) NSDate *timestamp;
@property(readonly, nullable) CLFloor *floor;
@end
```
Why Annotate Nullability?

New programming model communicates expectation to callers
Violation can cause crashes or other unexpected behavior
Swift enforces model in type system with optionals

```swift
class CLLocation : NSObject {
    public var timestamp: NSDate { get }
    public var floor: CLFloor? { get }
}
```
Finding Nullability Violations

Particularly useful in mixed Swift/Objective-C projects

Logical problem in code

Incorrect annotations
Violation: Branching with nil Default

```swift
- (nonnull NSString *)shortDescription {
    NSString *name = nil;

    if (self.cityName)
        name = self.cityName;
    if (self.countryName)
        name = self.countryName;

    return name;  // Null is returned from a method that is expected to return a non-null value
}
```
Violation: Branching with nil Default

- (nonnull NSString *)shortDescription {
  NSString *name = NSLocalizedString(@"Earth", @"The planet");

  if (self.cityName)
    name = self.cityName;
  if (self.countryName)
    name = self.countryName;

  return name;
}
Violation: Incorrect Annotation

```objective-c
 NS_ASSUME_NONNULL_BEGIN
@property(readonly) PressureData *pressure;
NS_ASSUME_NONNULL_END

- (PressureData *)pressure {
    if ([self hasBarometer])
        return [self measurePressure];
    return nil; // Null is returned from a method that is expected to return a non-null value
}
```
NS_ASSUME_NONNULL_BEGIN
@property(readonly, nullable) PressureData * pressure;
NS_ASSUME_NONNULL_END

- (PressureData *) pressure {
  if ([self hasBarometer]) {
    return [self measurePressure];
  }
  return nil;
}
Nullability of Your API is a Contract

- Do not change just to silence the analyzer
- Do carefully consider nullability of API
Return nil defensively for backwards compatibility:

```c
-(NSString * _Nonnull)stringAtIndex:(int) index {
    if (index < 0 || index >= _count) {
        return (NSString * _Nonnull) nil;
    }
    ...
}
```
Wrapping Up
These Tools Find Real Bugs!

Address Sanitizer and Thread Sanitizer
Clang Static Analyzer

Use on your code!
More Information

https://developer.apple.com/wwdc16/412
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internationalization Best Practices</td>
<td>Mission</td>
<td>Tuesday 9:00AM</td>
</tr>
<tr>
<td>Visual Debugging with Xcode</td>
<td>Presidio</td>
<td>Wednesday 4:00PM</td>
</tr>
<tr>
<td>Debugging Tips and Tricks</td>
<td>Pacific Heights</td>
<td>Friday 1:40PM</td>
</tr>
<tr>
<td>Using Time Profiler in Instruments</td>
<td>Nob Hill</td>
<td>Friday 3:00PM</td>
</tr>
<tr>
<td>Concurrent Programming with GCD in Swift 3</td>
<td>Pacific Heights</td>
<td>Friday 4:00PM</td>
</tr>
<tr>
<td>Labs</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Thread Sanitizer, Static Analysis, and LLVM Compiler Lab</td>
<td>Developer Tools Lab B</td>
<td>Thursday 12:00PM</td>
</tr>
<tr>
<td>Thread Sanitizer, Static Analysis, and LLVM Compiler Lab</td>
<td>Developer Tools Lab C</td>
<td>Friday 3:00PM</td>
</tr>
<tr>
<td>LLVM Compiler, Objective-C, and C++ Lab</td>
<td>Developer Tools Lab C</td>
<td>Friday 4:30PM</td>
</tr>
<tr>
<td>GCD Lab</td>
<td>Frameworks Lab D</td>
<td>Friday 5:00PM</td>
</tr>
</tbody>
</table>