SYNACKTIV

Demystifying Objective-C internals

A practical approach

Sthack
23/05/2025

PUBLIC

whoami

SYNACKTIV

- **Victor Cutillas**
- **Security researcher @ Synacktiv**
  - Reverse engineering
  - Exploitation
- **Mostly working on iOS**

Agenda

SYNACKTIV

- Context
- Objective-C 101
- Compilation
- Metadata
- Reversing tips
- Objective-C helper plugin
- Conclusion

Context

Context

SYNACKTIV

- **What is the aim of this talk?**
  - It is **not** to be technically exhaustive
    - You should be able to dive deeper yourself afterwards
  - It is **not** to discuss which tool is the best
    - We are focusing on understanding **static analysis** methodology
  - It is to give iOS newcomers keys to analyze Objective-C code
    - What is the role of the runtime?
    - How to efficiently reverse engineer Objective-C implementations?
    - How to recover types of manipulated variables?
  - An IDA plugin is also presented to ease analysis

Context

SYNACKTIV

- **Objective-C appeared in 1984**
  - Derived from Smalltalk
  - Used in NeXTSTEP, the historical code base of Darwin (macOS)
- **Apple language of reference until 2014**
  - New code bases now use Swift
  - As of iOS 18.4, among 3800+ libraries in the shared cache
    - **90%** rely on Objective-C code
  - Almost all core services are implemented using Objective-C

Context

SYNACKTIV

- **It is a superset of the C language**
  - Compatible with the C syntax
  - It is also possible to write Objective-C++
- **It is object-oriented**
  - Notion of classes, inheritance
- **… and reflective**

  Reflection is the ability of a process to examine, introspect, and modify its own structure and behavior.

  Reflective programming – Wikipedia

  - This implies **metadata**!

Objective-C 101

- **Basic terminology**
  - Class: set of methods, properties & instance variables
  - Instance variable: a field of a class
  - Property: instance variable with compiler-generated getter/setter
- **Objective-C is message-based**
  - Protocol: defines a required set of methods & properties
    - Similar to a C++ interface
  - Selector: pointer to a C-style string discriminating a method
  - Message passing: alternative name for method calling

- **`NSObject` root class**
  - All Objective-C classes inherit from this class
    - The only other root class is `NSProxy` (out of this talk scope)
  - Implemented by `Foundation`
  - Provides a default set of methods

```
raw.m:19:12: warning: class 'A' defined without specifying a base class [-Wobjc-root-class]
@interface A
           ^
raw.m:19:18: note: add a super class to fix this problem
@interface A
            ^
             : NSObject
```

- **`NSObject` root class**
  - Lifetime
    - `+ alloc`
    - `- retain`
    - `- release`
    - `- init`
  - Introspection
    - `+ isSubclassOfClass:`
    - `- isKindOfClass:`
    - `- respondsToSelector:`
    - `- valueForKey:`
  - Describing
    - `- description`
    - `- debugDescription`
  - Some of them are designed to be overridden, some examples:
    - `- description`
    - `- valueForUnknownKey:`

- **Objects lifetime**
  - Automatic Reference Counting (ARC) is used everywhere
  - No explicit calls to retain/release are made in the code
    - The compiler triggers an error if the developer uses them
  - Lifetime-related calls are automatically generated by the compiler

- **Objective-C runtime**
  - Source code is available on GitHub*
  - Objective-C code links to the runtime library
    - `libobjc.A.dylib`
  - The runtime exposes functions called by the compiler
    - `objc_msgSend()`
    - `objc_release()`
    - `objc_retain()`
    - `objc_opt_isKindOfClass()`
    - `objc_autoreleasePoolPush()`
    - …
  - It also exposes a user API: `<objc/runtime.h>`
    - More on that later

* https://github.com/apple-oss-distributions/objc4

- **`Foundation` framework**
  - Implements base objects, below are a few examples
  - Language features
    - Arrays, dictionaries, sets
    - Strings, numbers
    - Archiving, unarchiving
    - Errors
  - Operating system interactions
    - Filesystem
    - Networking
    - Process management
  - Inter-process communication (NSXPC)

Compilation

Compilation

SYNACKTIV

- **Toolchain**
  - Apple uses the LLVM toolchain
    - It is opensource*
    - Apple maintains its own version with proprietary patches
  - Clang is therefore the officially supported compiler
    - It is used by Apple's IDE, Xcode
    - It can also be directly used in the command line

* https://llvm.org/

Compilation

SYNACKTIV

- **Code generation**
  - Consider Objective-C as being internally transpiled to C
  - Here is a method from *UIKitCore* class `UIView`

```
@implementation UIView

- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    [self->_traitChangeRegistry
        registerAuxiliaryChildEnvironmentForTraitInvalidations:obj];
}

@end
```

  - Let's understand its generated implementation step by step

Compilation

SYNACKTIV

- **Code generation**
  - Method calls are converted to `objc_msgSend()` calls

```
- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    [self->_traitChangeRegistry
        registerAuxiliaryChildEnvironmentForTraitInvalidations:obj];
}
```

```
- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    objc_msgSend(self->_traitChangeRegistry,
                 "registerAuxiliaryChildEnvironmentForTraitInvalidations:",
                 obj);
}
```

Compilation

SYNACKTIV

- **Code generation**
  - This previous method call

```
- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    objc_msgSend(self->_traitChangeRegistry,
                 "registerAuxiliaryChildEnvironmentForTraitInvalidations:",
                 obj);
}
```

    - ... is made on an object
    - ... invokes the method identified by its selector
    - ... passes one method argument

Compilation

SYNACKTIV

- **Code generation**
  - Note: the code is a bit simplified for readability
    - `objc_msgSend()` must actually be cast to be called
    - The selector must be wrapped

```
- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    (void (*)(id, SEL, id) objc_msgSend)(
        self->_traitChangeRegistry,
        @selector(registerAuxiliaryChildEnvironmentForTraitInvalidations:),
        obj);
}
```

Compilation

SYNACKTIV

- **Code generation**
  - The example `UIView` method

```
- (void)_registerAuxiliaryChildEnvironmentForTraitInvalidations:(id)obj;
```

    - … is called on an instance of a class
    - … is identified by its selector
    - … expects one argument
    - … does not have a return value
  - Here is the corresponding C implementation declaration

```
void _registerAuxiliaryChildEnvironmentForTraitInvalidations(UIView *self, SEL sel, id obj);
```

  - This is identical to the corresponding `objc_msgSend()` calls

Compilation

SYNACKTIV

- **Code generation**
  - `[[cls alloc] init]` is a common object creation pattern
    - This call chain is optimized by the compiler

```
void _registerAuxiliaryChildEnvironmentForTraitInvalidations(UIView *self, SEL sel, id obj)
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = [[_UITraitChangeRegistry alloc] init];
    objc_msgSend(self->_traitChangeRegistry,
                 "registerAuxiliaryChildEnvironmentForTraitInvalidations:",
                 obj);
}
```

```
void _registerAuxiliaryChildEnvironmentForTraitInvalidations(UIView *self, SEL sel, id obj)
{
    if (self->_traitChangeRegistry == nil)
        self->_traitChangeRegistry = objc_alloc_init(&OBJC_CLASS____UITraitChangeRegistry);
    objc_msgSend(self->_traitChangeRegistry,
                 "registerAuxiliaryChildEnvironmentForTraitInvalidations:",
                 obj);
}
```

Compilation

SYNACKTIV

- **Code generation**
  - ARC automatically calls lifetime-related functions
    - `objc_retain()`, `objc_release()`
    - `objc_autorelease()`, `objc_claimAutoreleasedReturnValue()`

```
void _registerAuxiliaryChildEnvironmentForTraitInvalidations(UIView *self, SEL sel, id obj)
{
    obj = objc_retain(obj);
    if (self->_traitChangeRegistry == nil) {
        _UITraitChangeRegistry *old = self->_traitChangeRegistry;
        self->_traitChangeRegistry = objc_alloc_init(&OBJC_CLASS____UITraitChangeRegistry);
        objc_release(old);
    }
    objc_msgSend(self->_traitChangeRegistry,
                 "registerAuxiliaryChildEnvironmentForTraitInvalidations:",
                 obj);
    objc_release(obj);
}
```

Compilation

SYNACKTIV

- **Code generation**
  - It is possible to convert Objective-C to C++ using a compiler flag
    - This may help you better understand code generation
    - Is is also pretty useful to better understand metadata structures

```
clang -arch arm64 -isysroot "$SDK_ROOT" -I"$SDK_ROOT"/usr/include -fobjc-arc -rewrite-objc  code.m
```

```
static void _I_UIView__registerAuxiliaryChildEnvironmentForTraitInvalidations_(UIView * self, SEL _cmd, __strong id obj) {
    if ((*(_UITraitChangeRegistry *__strong *)((char *)self + OBJC_IVAR_$_UIView$_traitChangeRegistry)) == __null)
        (*(_UITraitChangeRegistry *__strong *)((char *)self + OBJC_IVAR_$_UIView$_traitChangeRegistry)) =
            ((_UITraitChangeRegistry *(*)(id, SEL))(void *)objc_msgSend)((id)((
                _UITraitChangeRegistry *(*)(id, SEL))(void *)objc_msgSend)(
                    (id)objc_getClass("_UITraitChangeRegistry"), sel_registerName("alloc")), sel_registerName("init"));
    ((void (*)(id, SEL, __strong id))(void *)objc_msgSend)(
        (id)(*(_UITraitChangeRegistry *__strong *)((char *)self + OBJC_IVAR_$_UIView$_traitChangeRegistry)),
        sel_registerName("registerAuxiliaryChildEnvironmentForTraitInvalidations:"), (id)obj);
}
```

Compilation

SYNACKTIV

- **Method calls optimization**
  - Since iOS 17, no direct calls to `objc_msgSend()` are made
    - Stubs are generated and shared as much as possible
    - They are shared between shared cache libraries

```
BL      _objc_msgSend$_registerAuxiliaryChildEnvironmentForTraitInvalidations_
```



```
_objc_msgSend$_registerAuxiliaryChildEnvironmentForTraitInvalidations_:
ADRP    X1, #selRef__registerAuxiliaryChildEnvironmentForTraitInvalidations_@PAGE
LDR     X1, [X1,#selRef__registerAuxiliaryChildEnvironmentForTraitInvalidations_@PAGEOFF]
ADRP    X16, #_objc_msgSend_ptr@PAGE
LDR     X16, [X16,#_objc_msgSend_ptr@PAGEOFF]
BR      X16
```

Compilation

SYNACKTIV

- **Objective-C blocks**
  - Anonymous functions, similar to C++ lambdas
  - **Very common** in Apple code
    - Mostly because of interactions with `libdispatch.dylib`
    - This library provides a C API for executing code concurrently
  - Most common use cases are
    - Lazy initialization → `dispatch_once()`
    - Asynchronous function call → `dispatch_async()`
    - Synchronous function call → `dispatch_sync()`
    - Various callbacks → filtering, enumeration, …

Compilation

SYNACKTIV

- **Objective-C blocks**
  - Example method from `NSConcreteFileHandle` (*Foundation*)

```
- (void)closeFile
{
    if (self->_flags & FILE_CLOSED)
        return;
    [self _cancelDispatchSources];
    dispatch_async(self->_monitoringQueue, ^(){
        self->_flags |= FILE_CLOSED;
        close(self->_fd);
    });
}
```

  - The block code will be executed on the given queue (a thread)

Compilation

SYNACKTIV

- Objective-C blocks

```
- (void)closeFile
{
    if (self->_flags & FILE_CLOSED)
        return;
    [self _cancelDispatchSources];
    dispatch_async(self->_monitoringQueue, ^(){
        self->_flags |= FILE_CLOSED;
        close(self->_fd);
    });
}
```

```
void closeFile(NSConcreteFileHandle *self, SEL sel)
{
    if (self->_flags & FILE_CLOSED)
        return;
    objc_msgSend(self, "_cancelDispatchSources");
    Block_layout_closeFile block = {
        .isa = &OBJC_CLASS_____NSStackBlock__,
        .flags = 0xC0000000,
        .invoke = closeFile_block_invoke,
        .descriptor = &closeFile_block_descriptor,
        .self = self,
    };
    dispatch_async(self->_monitoringQueue, &block);
}
```

```
void closeFile_block_invoke(Block_layout_closeFile *block)
{
    block->self->_flags |= FILE_CLOSED;
    close(block->self->_fd);
}
```

Compilation

SYNACKTIV

- **Objective-C blocks**
  - Blocks are objects generated by the compiler

```
struct Block_layout_closeFile {
    Class isa;
    int32_t flags;
    int32_t reserved;
    void (*invoke)(Block_layout_closeFile *block);
    Block_descriptor *descriptor;
    // Captured variables
    NSConcreteFileHandle *self;
};
```

    - `invoke` points to a native function containing the block code
      - Blocks can also take parameters, additionally passed to this function
    - `descriptor` provides metadata about the block (more on that later)
    - The layout of the structure also includes captured variables

Compilation

SYNACKTIV

- Objective-C blocks

```
- (void)closeFile
{
    if (self->_flags & FILE_CLOSED)
        return;
    [self _cancelDispatchSources];
    dispatch_async(self->_monitoringQueue, ^(NSError *error){
        if (error == nil) {
            self->_flags |= FILE_CLOSED;
            close(self->_fd);
        }
    });
}
```



```
void closeFile_block_invoke(Block_layout_closeFile *block, NSError *error)
{
    if (error == nil) {
        block->self->_flags |= FILE_CLOSED;
        close(block->self->_fd);
    }
}
```

Compilation

SYNACKTIV

- **Objective-C blocks**
  - You will encounter two main types of blocks
  - Global blocks
    - Are stored in a read-only section of the Mach-O
    - Typically used for lazy initialization of global data: `dispatch_once()`
  - Stack blocks
    - Created at runtime
    - Used when local variables are captured

Metadata

Metadata

SYNACKTIV

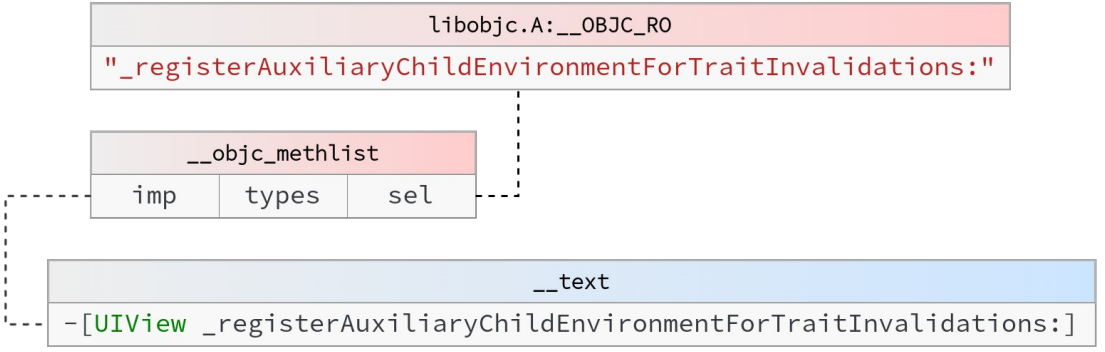
- **Objective-C data is stored in dedicated sections**
  - This allows *dyld* to properly register Objective-C information to the runtime
  - Their name is prefixed with `__objc_`, here are some examples

Section name	Contents
`__objc_classlist` `__objc_protolist`	Pointers to defined classes & protocols
`__objc_selrefs` `__objc_classrefs`	Pointers to selectors & classes used by this Mach-O
`__objc_ivar`	Instance variable offsets
`__objc_data`	Structures of defined classes & metaclasses
`__objc_const`	Constant data: typically classes & metaclasses information

Compilation

SYNACKTIV

- **Listing method calls**
  1) Go to the corresponding `method_t` structure
     - By listing references to the implementation
     - Or directly searching the selector string

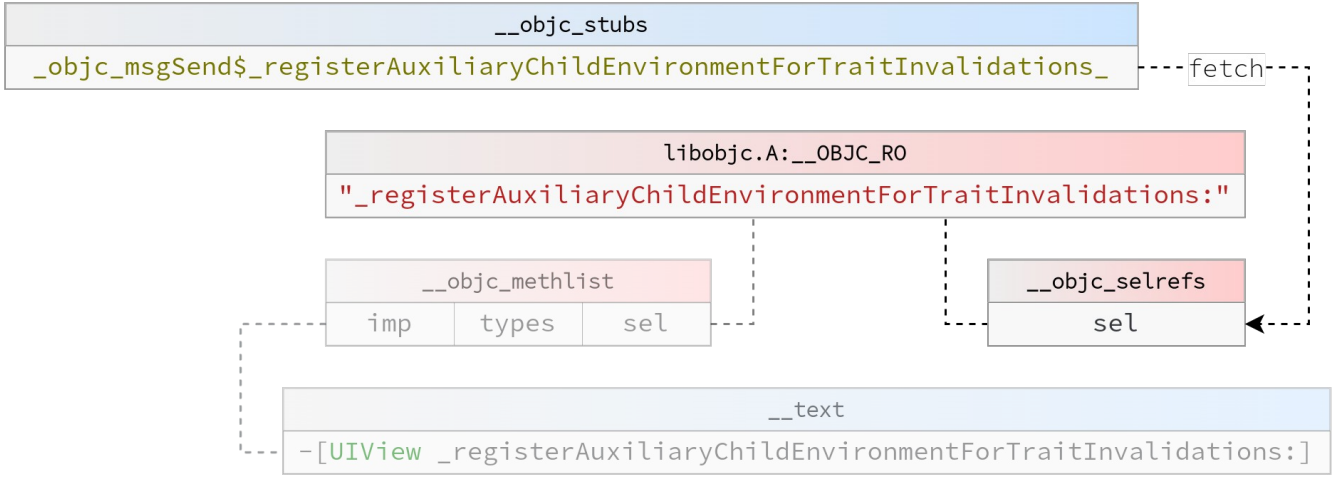


Compilation

SYNACKTIV

- **Listing method calls**

  2) Find all stubs referencing this selector

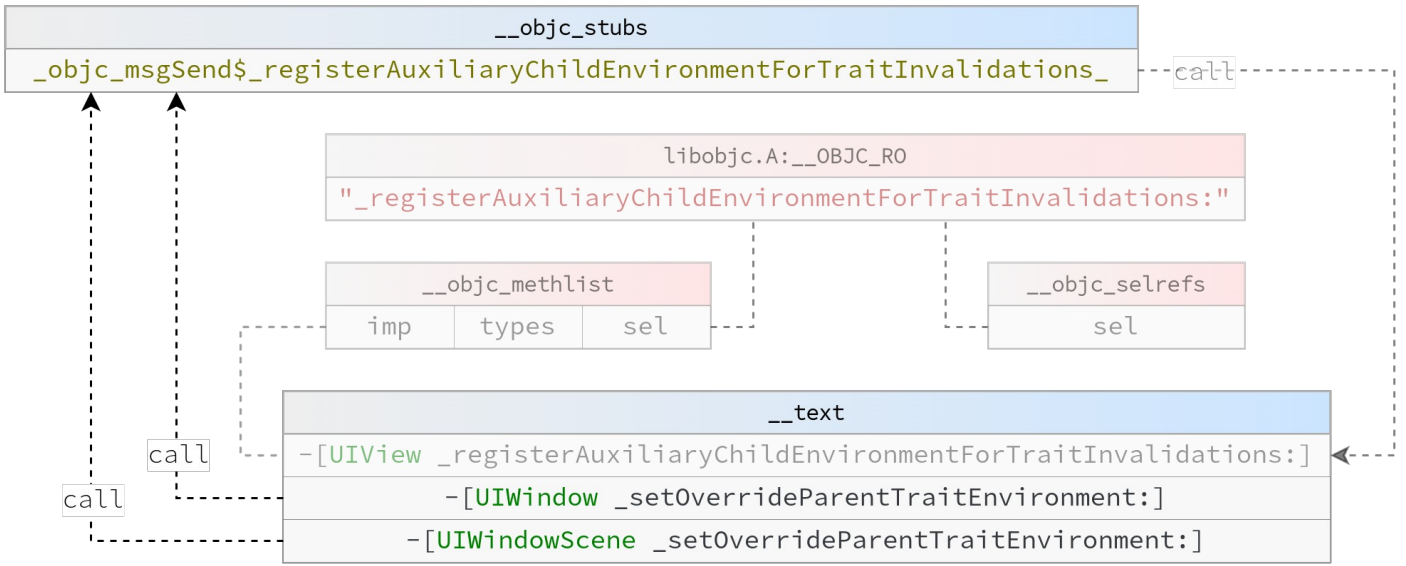


Compilation

SYNACKTIV

- **Listing method calls**

  3) List & filter stub calls if other methods share this selector



Metadata

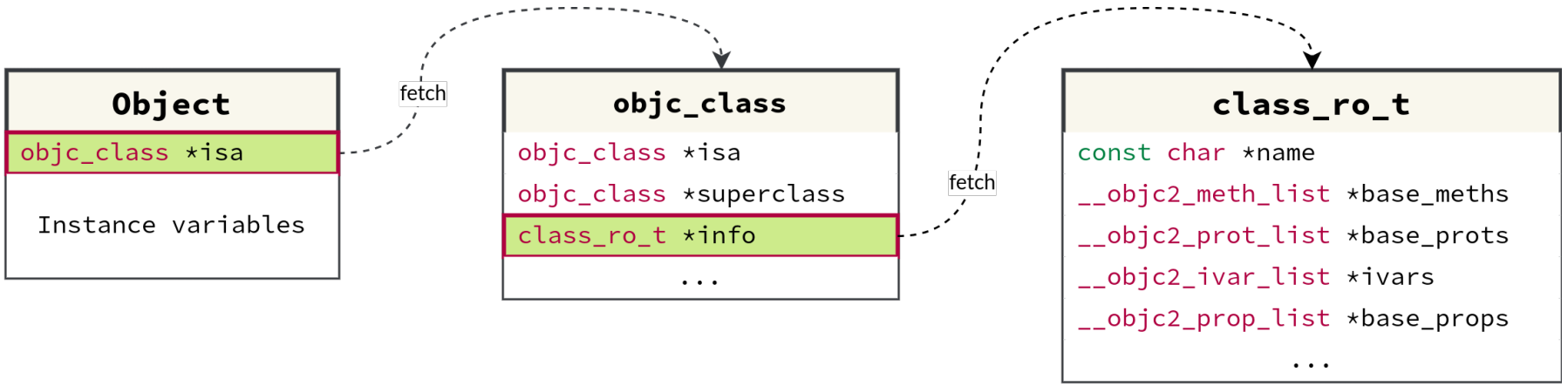
SYNACKTIV

- **Calling a method**
  - Let's figure out how `objc_msgSend()` decides what to call
  - This will walk us through the Objective-C metadata structures
  - Keep in mind that all data structures are public

Metadata

SYNACKTIV

- **Calling a method**
  1) ISA (class) pointer is used to fetch metadata about the object
  2) Class information is then fetched



Metadata

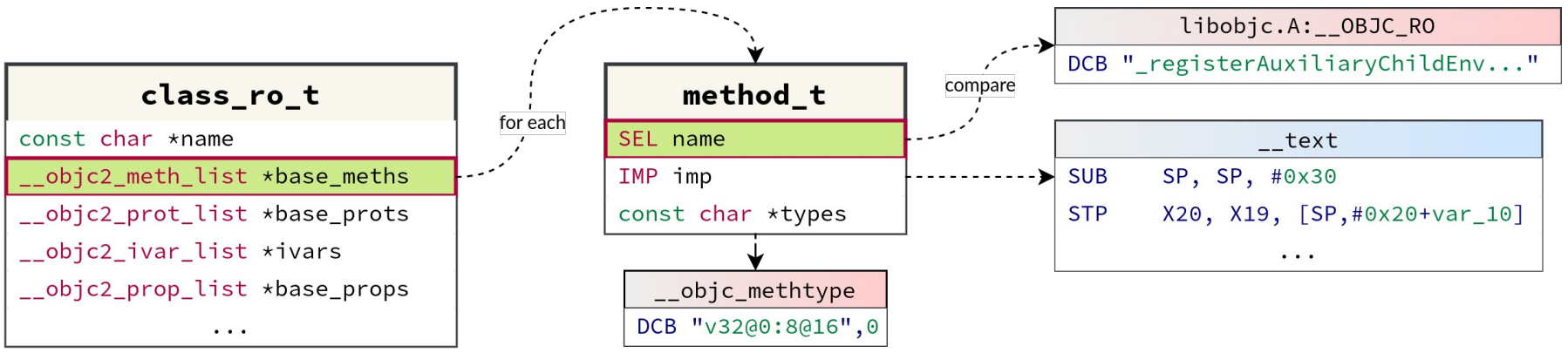
SYNACKTIV

- **Calling a method**
  - The runtime now has access to metadata lists concerning
    - `ivars` Instance variables (structure fields)
    - `base_props` Class properties (*ivars* with getter and/or setter)
    - `base_meths` Defined methods
    - `base_prots` Protocols the class conforms to

Metadata

SYNACKTIV

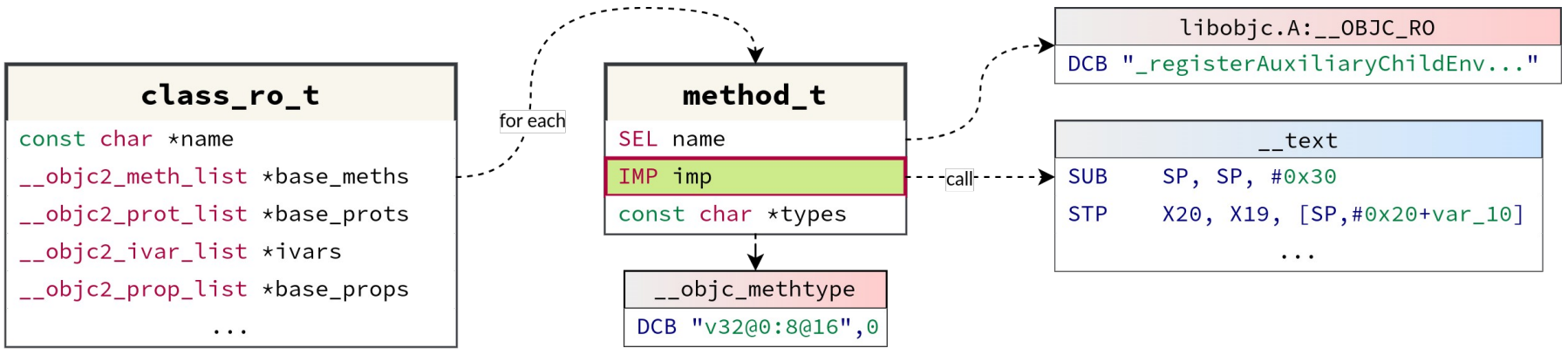
- **Calling a method**
  3) Runtime iterates over the list of all declared methods
  4) Requested selector is compared with each method selector



SYNACKTIV

- **Calling a method**

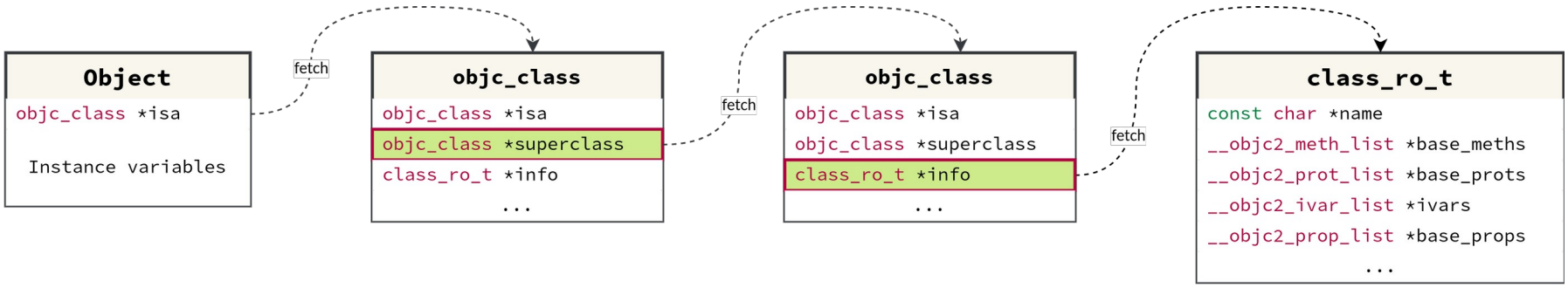
  5) When a match is found, the implementation is called



Metadata

SYNACKTIV

- **Calling a method**
  - 6) Method not found?
    - Start over by looking it up in the *super* class

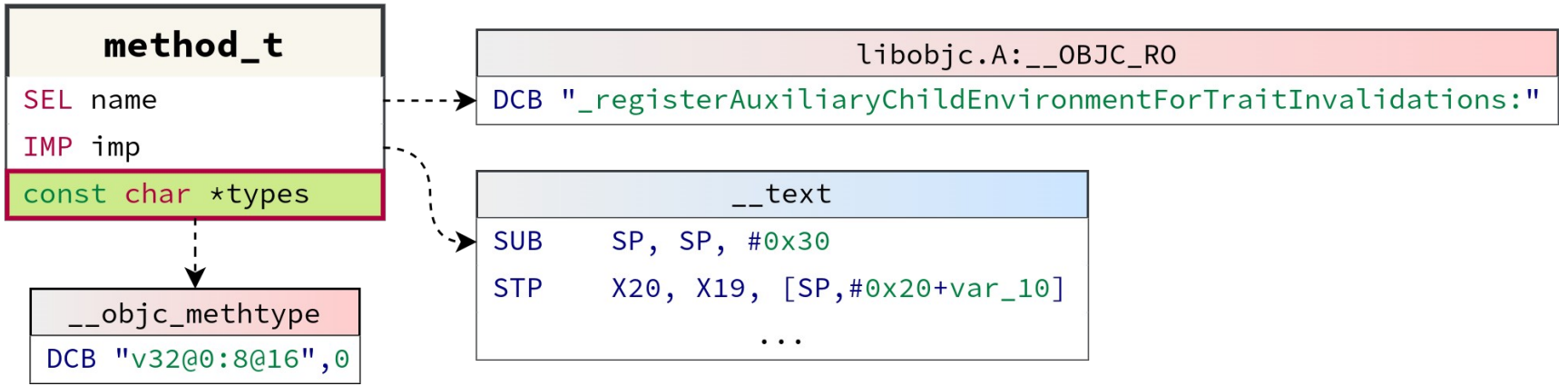


    - No super class? Runtime calls `[obj doesNotRecognizeSelector:]`
      - `NSObject` implementation throws an exception, leading to a crash

Metadata

SYNACKTIV

- **Method signature**
  - Let's take a look at this `types` field

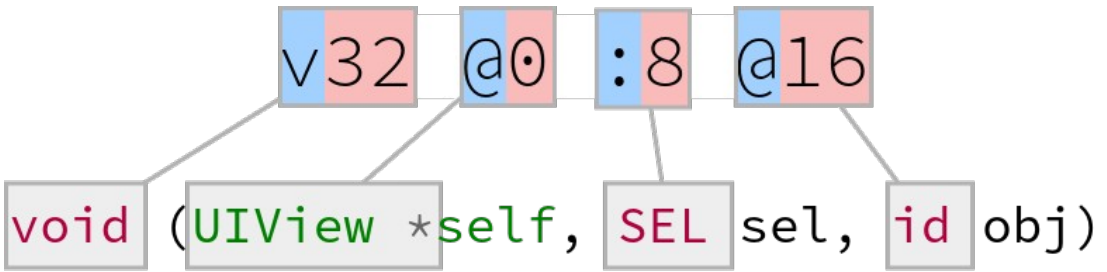


  - This string encodes information about arguments

Metadata

SYNACKTIV

- **Method signature**
  - Type encoding is partially documented
  - Describes return value and arguments: type, location



  - `@0:8` is by definition common to all Objective-C methods

Metadata

SYNACKTIV

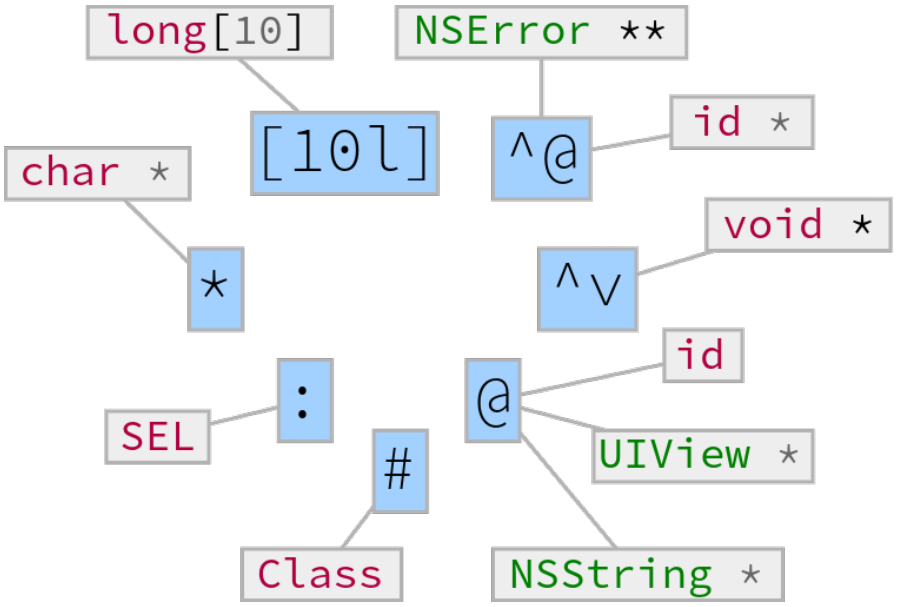
- **Method signature**
  - Side note: variadic Objective-C methods exist
    - Yet, no metadata is available to check that they are variadic
    - Method signatures simply stop before the variadic arguments

```
@interface NSSet

// Signature is "@24@0:8@16"
+ (NSSet *)setWithObject:(id)obj;
// Signature is identical
+ (NSSet *)setWithObjects:(id)obj, ...;

@end
```

Metadata

SYNACKTIV

- **Type encoding**
  - Describes primitive types
    - Numbers
    - Objects (instance, class)
  - … and recursive types
    - Pointers
    - Arrays
    - Structures, unions
  - A public documentation exists



Metadata

SYNACKTIV

- **Type encoding**
  - Encoded structure type

```
^{UIContentViewElementLayoutInfo={CGSize=dd}B{NSDirectionalEdgeInsets=dddd}}
```

  - Argument type: `UIContentViewElementLayoutInfo *`
  - Associated data structures

```
struct UIContentViewElementLayoutInfo {
  CGSize var1;
  BOOL var2;
  NSDirectionalEdgeInsets var3;
};
```

```
struct NSDirectionalEdgeInsets {
  double var1;
  double var2;
  double var3;
  double var4;
};
```

```
struct CGSize {
  double var1;
  double var2;
};
```

Metadata

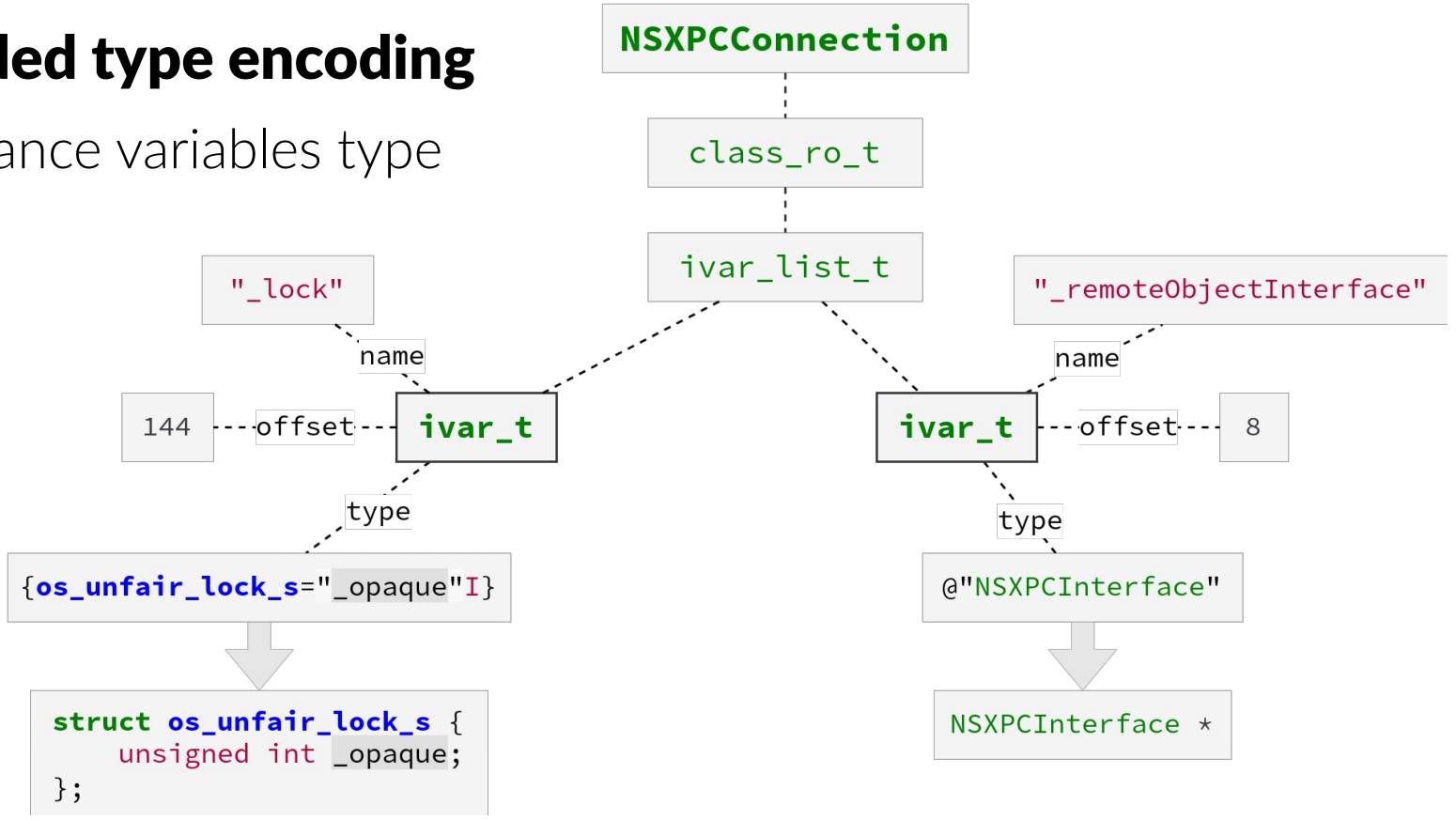
SYNACKTIV

- **Extended type encoding**
  - Some metadata structures contain *extended* type encoding
    - Instance variables information
    - Protocols
    - Block descriptors
  - They contain additional information about objects & structures
    - Protocol the object shall conform to
    - Expected object class
    - Structure field names

Metadata

SYNACKTIV

- **Extended type encoding**
  - Instance variables type



Metadata

SYNACKTIV

- **Extended type encoding**
  - Block signature
    - Here a protocol is specified for the return value

```
@"<NSCopying>"40@?0r^v8{_NSRange=QQ}16^B32
```



```
id<NSCopying>(^)(Block_layout_f00 *, const void *, _NSRange, bool *);
```

  - Protocol method
    - Object class names in protocols are required for NSXPC serialization

```
@"NSData"16@0:8
```



```
NSData *(id, SEL);
```

Reversing tips

Reversing tips – Recovering types

SYNACKTIV

- **Some instance variables are accessed through their metadata**
  - Their offset in the object is fetched at runtime
    - This is not true for all instance variables
  - It is then possible to deduce the associated object type

```
ADRP      X8, #_OBJC_IVAR_$_UIView._traitChangeRegistry@PAGE
LDRSW     X8, [X8,#_OBJC_IVAR_$_UIView._traitChangeRegistry@PAGEOFF]
LDR       X0, [X23,X8]    ; X23 is v5
BL        ___UITraitChangeRegistry_updateAuxiliaryChildrenTraitsIfNeeded_
```

```
-[_UITraitChangeRegistry updateAuxiliaryChildrenTraitsIfNeeded](*((_QWORD *)v5 + 6));
```

    - Here, the decompiler inlines the offset value and adds it to `v5`
      - `v5` can therefore be safely assumed to be `UIView *` (or a subclass)

```
-[_UITraitChangeRegistry updateAuxiliaryChildrenTraitsIfNeeded](v5->_traitChangeRegistry);
```

Reversing tips – Recovering types

SYNACKTIV

- **Some instance variables are accessed through their metadata**
  - Accesses to this instance variable can be easily listed
    - Keep in mind that this is only valid for a subset of instance variables

```
; void __cdecl -[UIView _unregisterAuxiliaryChildEnvironmentForTraitInvalidations:](UIView *self, SEL, id)
__UIView__unregisterAuxiliaryChildEnvironmentForTraitInvalidations__
                                        ; DATA XREF: UIKitCore:__objc_methlist:000000018ADF96A8↓o
                ADRP            X8, #_OBJC_IVAR_$_UIView._traitChangeRegistry@PAGE ; _UITraitChangeRegistry *_traitChangeRegistry;
                LDRSW           X8, [X8,#_OBJC_IVAR_$_UIView._traitChangeRegistry@PAGEOFF] ; _UITraitChangeRegistry *_traitChangeR
                LDR             X8, [X0,X8]
```

xrefs to _OBJC_IVAR_$_UIView._traitChangeRegistry

Direction	Type	Address	Text	
Up	o	-[UIView _wrappedProcessTraitChanges:withBehavior:]:loc_189B4480C	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
Up	o	-[UIView _traitCollectionDidChangeInternal:]+2B0	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
Up	o	-[UIView _registerForTraitTokenChanges:withTarget:action:]+30	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
Up	o	-[UIView _registerForTraitTokenChanges:withHandler:]+28	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
Up	o	-[UIView unregisterForTraitChanges:]+4	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
Up	o	-[UIView _registerAuxiliaryChildEnvironmentForTraitInvalidations:]+1C	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...
	o	-[UIView _unregisterAuxiliaryChildEnvironmentForTraitInvalidations:]	ADRP	X8, #_OBJC_IVAR_$_UIView._traitChangeR...

adrp

Line 7 of 7

OK Cancel Search Help

  - This type recovery method can be automated in your analysis tool

Reversing tips

SYNACKTIV

- **Private frameworks can be imported**
  - Objective-C frameworks can be dynamically loaded via `dlopen()`
    - Use the runtime API to manipulate private classes
  - Example: check if the device sends crash logs to Apple

```
@interface OSASystemConfiguration : NSObject
- (BOOL)optInApple;
@end

int main(void)
{
    dlopen("/System/Library/PrivateFrameworks/OSAnalytics.framework/OSAnalytics",
           RTLD_NOW);
    OSASystemConfiguration *conf = [[objc_getClass("OSASystemConfiguration") alloc] init];
    printf("optInApple = %s\n", conf.optInApple ? "true" : "false");
}
```

Reversing tips

SYNACKTIV

- **Instrumenting methods**
  - The runtime provides a simple way to hook method calls

```objc
// Hooking -[NSArray description]
NSString *(*description_impl)(id, SEL) = NULL;
NSString *description_hook(id self, SEL sel) {
    NSString *desc = description_impl(self, sel);
    printf("%p description: %s", (__bridge void *)self,
           desc.UTF8String);
    return desc;
}

int main(void) {
    Method method = class_getInstanceMethod(objc_getClass("NSArray"),
                                            sel_registerName("description"));
    description_impl = method_setImplementation(method, description_hook);
    NSLog(@"Array: %@", @[ @42, @"hello" ]);  // Calls -[NSArray description]
}
```

```
$ ./hook
0x109b380d0 description: (
    42,
    hello
)
```

Reversing tips

SYNACKTIV

- **Reference counting optimization**
  - Since iOS 16, calls to *refcount*-related functions are specialized

```
id objc_retain(id obj);
void objc_release(id obj);
```

  - `obj` parameter can now be passed in any ARM64 register
    - From `X0` to `X25`
    - Except sensitive registers `X16`, `X17`, `X18`
  - This reduces register manipulation in generated assembly
    - Return value is still stored in `X0`

Reversing tips

SYNACKTIV

- **Reference counting optimization**
  - Specialized flavours are exposed by the runtime
    - See header file `runtime/objc-abi.h`
  - A decent reversing tool should support those calls
    - They are omnipresent in generated assembly code
    - Decompiler output is messy without support

```
_registerAuxiliaryChildEnvironmentForTraitInvalidations:
    SUB     SP, SP, #0x30
    STP     X20, X19, [SP,#0x20+var_10]
    STP     X29, X30, [SP,#0x20+var_s0]
    ADD     X29, SP, #0x20
    MOV     X19, X0
    BL      _objc_retain_x2     ; Argument is retained
    MOV     X1, X0
```



```
_objc_retain_x2:
    ANDS    X0, X2, X2
    B.LE    objc_retain_ret
    LDXR    X16, [X2]
```

Reversing tips

SYNACKTIV

- **Runtime uses Pointer Authentication Code (PAC)**
  - Cryptographic ARM64 hardware protection
    - Available since iPhone XS & XR (A12+ SoCs)
  - Enables developers to **protect pointers with signatures**
  - An attacker controlling a protected pointer must then provide
    - A valid address
    - A valid signature of this address
    - Otherwise a fatal error is triggered (kernel sends a `SIGKILL`)
  - This *dramatically* increases exploitation difficulty

Reversing tips

SYNACKTIV

- **Runtime uses Pointer Authentication Code (PAC)**
  - Data pointers
    - Object class pointers (ISA) are protected
      - Prevents memory corruption & Use-After-Free bugs
    - Caches, custom virtual tables, dynamically registered metadata, etc.
  - Function pointers
    - Exception handling, image loading, internal hooks, trampolines, etc.
  - Distinct discriminators are used for all pointer types
    - Integers scrambled with the pointer they are signed with
    - Provides a semantic enforcement on signed pointers authentication

Objective-C helper plugin

Objective-C helper plugin

SYNACKTIV

- **Decompiler output cleanup**
  - Decompiler output contains many (usually) irrelevant runtime calls
    - Lifetime
      `objc_retain()`, `objc_release()`
    - Return values
      `objc_autorelease()`, `objc_claimAutoreleasedReturnValue()`
    - Property manipulation
      `objc_storeStrong()`, `objc_loadWeakRetained()`
  - They slow down the analyst when reading code
  - … and also prevent type propagation

Objective-C helper plugin

SYNACKTIV

- **Decompiler output cleanup**
  - Return values are made opaque when claimed
    - Ambiguous variable typing
    - Method implementations are unresolved on untyped variables

```
UIViewControllerViewAnimator *from;
id view;
id window;
id controller;
```

```
view = objc_claimAutoreleasedReturnValue(
  -[UIViewControllerViewAnimator view](
    from, "view"));
window = objc_claimAutoreleasedReturnValue(
  objc_msgSend(v77, "window"));
controller = objc_claimAutoreleasedReturnValue(
  +[UIWindowController windowControllerForWindow:](
    &OBJC_CLASS___UIWindowController,
    "windowControllerForWindow:",
    window));
objc_release(v78);
objc_release(v77);
```

Objective-C helper plugin

SYNACKTIV

- **Decompiler output cleanup**
  - Removing those calls enhances the analysis efficiency
    - Greatly improved type propagation
      - Method calls are chained
    - Fewer intermediate variables
    - Around 10% to 20% fewer lines of decompiled code

```
UIViewController *from;
UIWindowController *controller;
```

```
controller = +[UIWindowController windowControllerForWindow:](
  &OBJC_CLASS___UIWindowController,
  "windowControllerForWindow:",
  -[UIView window](-[UIViewController view](from, "view"), "window"));
```

Objective-C helper plugin

SYNACKTIV

- **Decompiler output cleanup**
  - Implemented using a microcode hook (`MMAT_PREOPTIMIZED`)
    - Intermediate language used by Hex-Rays decompiler
  - Runtime calls are replaced by micro instructions "*neutralizing*" them
    - They replicate the caller's state after the call is made
  - Example of a replacement

1) ARM64 assembly

```
BL      _objc_retain_x3
```

2) Generated microcode

```
call    !objc_retain <fast:"id obj" x3.8> => id x0.8
```

3) Neutralization

```
mov     x3.8, x0.8
```

Objective-C helper plugin

SYNACKTIV

- **Other feature**
  - Propagation of cross-references to current method selector
  - Release will be made very soon on Synacktiv's Github page*

* https://github.com/synacktiv/

Conclusion

Conclusion

SYNACKTIV

- **Objective-C provides a lot of metadata**
  - It is very reversing-friendly
  - Structures can be easily recovered
  - Manipulated types are pretty straightforward to determine
- **It will not disappear anytime soon**
  - 90% of shared cache libraries rely on its use
  - It is deeply integrated inside the core of iOS

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://x.com/synacktiv

https://bsky.app/profile/synacktiv.com

https://synacktiv.com

PUBLIC