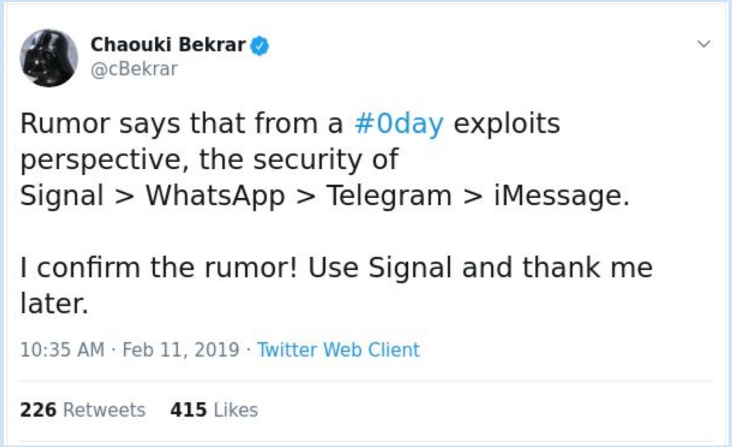
Look, No Hands!

The Remote, Interaction-less Attack Surface of the iPhone

About Me

- Natalie Silvanovich AKA natashenka
- Project Zero member
- Previously did mobile security on Android and BlackBerry
- Messaging enthusiast

iMessage Exploits


Chaouki Bekrar
@cBekrar

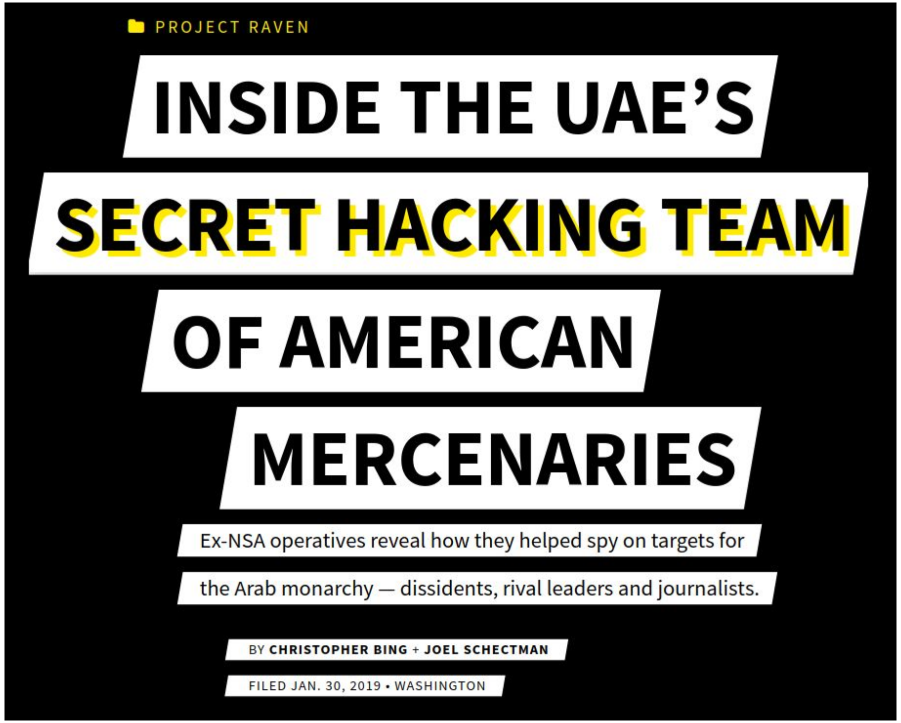
Rumor says that from a #0day exploits perspective, the security of
Signal > WhatsApp > Telegram > iMessage.

I confirm the rumor! Use Signal and thank me later.

10:35 AM · Feb 11, 2019 · Twitter Web Client

226 Retweets 415 Likes

iMessage Exploits


PROJECT RAVEN

INSIDE THE UAE'S SECRET HACKING TEAM OF AMERICAN MERCENARIES

Ex-NSA operatives reveal how they helped spy on targets for the Arab monarchy — dissidents, rival leaders and journalists.

BY CHRISTOPHER BING + JOEL SCHECTMAN

FILED JAN. 30, 2019 • WASHINGTON

"Karma allowed Raven to obtain emails, location, text messages and photographs from iPhones simply by uploading lists of numbers into a preconfigured system, five former project employees said. "

"Karma was particularly potent because it did not require a target to click on any link to download malicious software. The operatives understood the hacking tool to rely on an undisclosed vulnerability in Apple's iMessage text messaging software."

Google

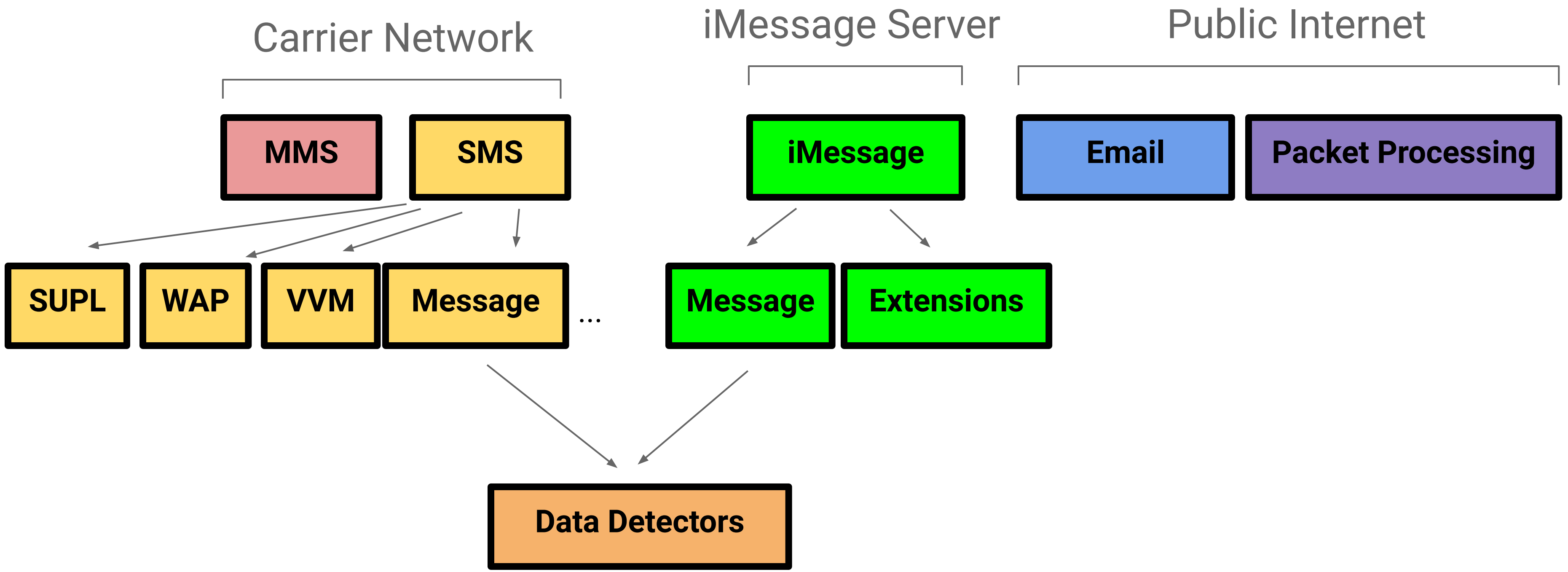
Questions

- Do bugs exist?
  - Where?
  - How do they work?
- What is the remote attack surface of the iPhone
  - Is it just iMessage?
- Are they exploitable?

Fully Remote Bugs

- Also “interaction-less” or “zero click”
- No user interaction required
- Short wait time
- Require a reasonable set of identifiers
  - Email address
  - Phone number

iPhone Remote Attack Surface

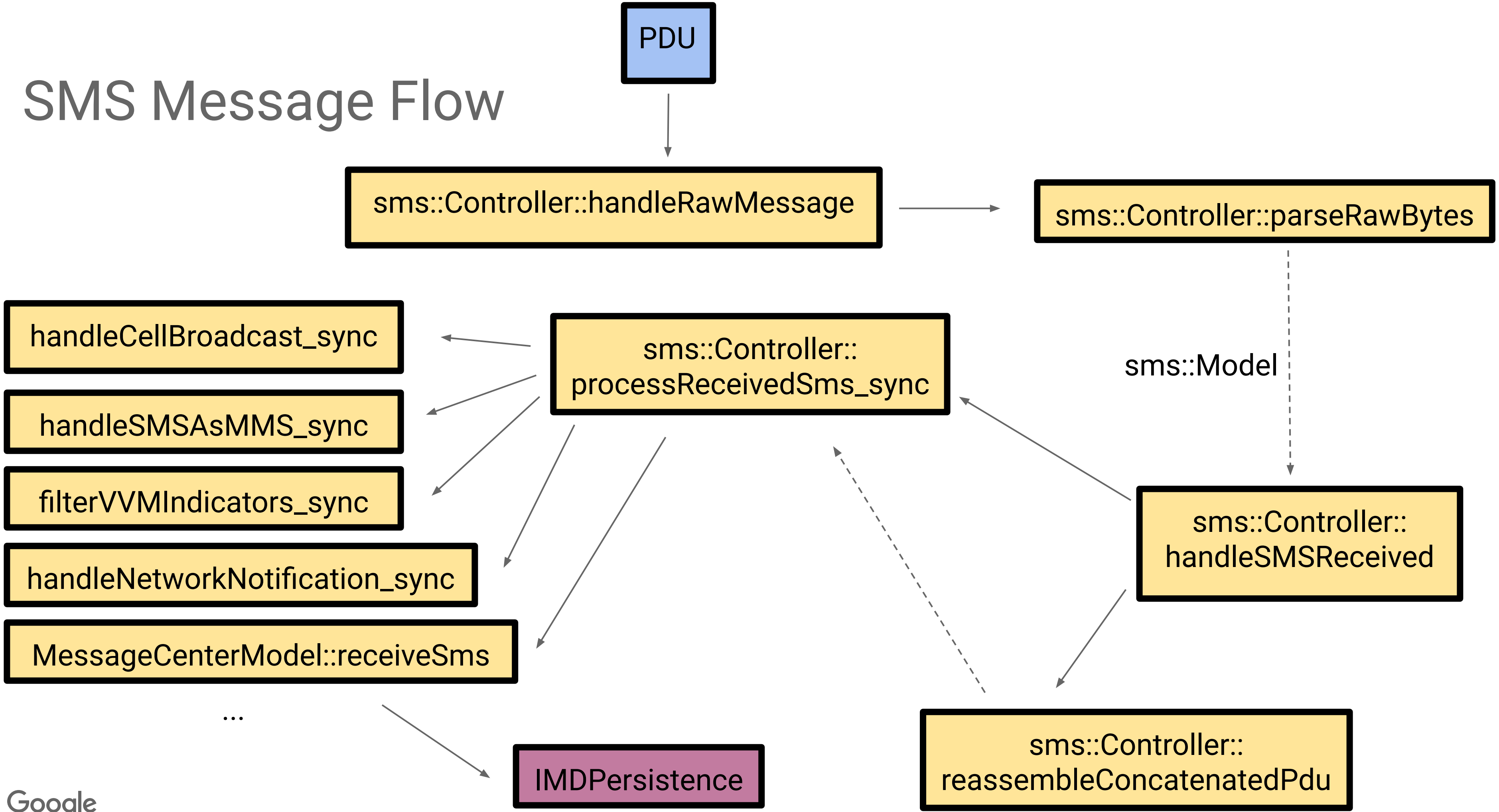


Google

SMS/MMS

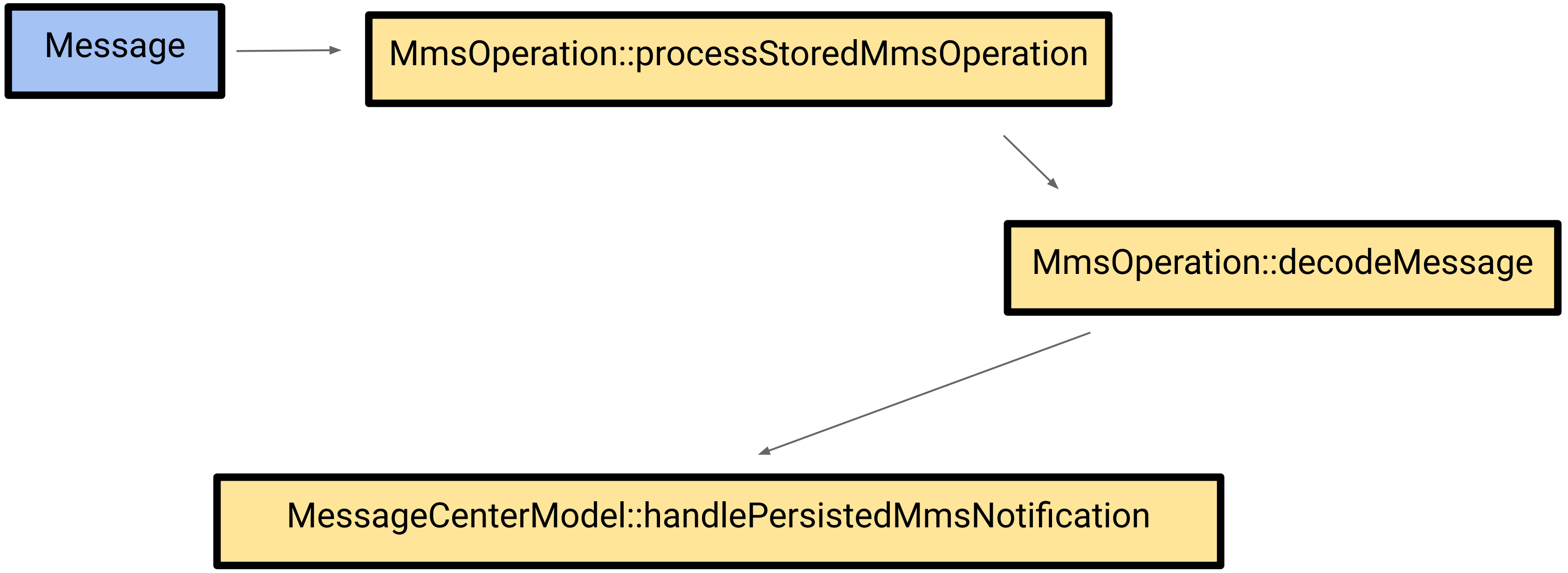
- Started by looking at SMS/MMS
  - This was the wrong call in hindsight
- Most processing is in CommCenter binary

SMS Message Flow



Google

MMS Message Flow


Message
MmsOperation::processStoredMmsOperation
MmsOperation::decodeMessage
MessageCenterModel::handlePersistedMmsNotification

Testing

- Can write applications that call exported symbols
    - Allows limited fuzzing
- Modified Android device to send raw SMS PDUs
    - sendRawPdu in SMSDispatcher.java

SMS Simulation

- CommCenter contains an SMS simulator
  - See sms::Controller::simulateSmsReceived
- Requires a library not included in standard iPhone software
- Implemented library that calls sms::Controller::simulateSmsReceived
- Can then simulate SMS over XPC
- See code on GitHub: https://github.com/googleprojectzero

Google

VVM

- Visual Voicemail is an interesting SMS receiver
- Intended use: carrier sends SMS to indicate new voicemail message available
- VVM SMS messages can be sent from any mobile device

VVM

- Sample message (decoded)

```
STATE?state=Active;server=vvm.att.com
;port=143;pw=asdf;name=5556667777@att
.com
```

- Device contacts IMAP server when SMS is received

VVM

- IMAP is available as a fully remote attack surface
  - Equivalent to connecting to a malicious IMAP server
- **`PrivateFrameworks/VisualVoicemail.framework/IMAP.vvservice/IMAP`** in dyld_shared_cache
- Some limitations
  - Must be supported by carrier*
  - Carrier filtering
  - User must have configured voicemail

VVM

- Reviewed IMAP service in IDA
- Wrote a fuzzer that generated malformed IMAP
  - Used SMS simulation to cause device to continuously ping server
- Found one vulnerability

CVE-2019-8613

Use-after-free in IMAP NAMESPACE processing

- Device sends LIST to get separator
- Device sends NAMESPACE to get prefix
- If NAMESPACE fails, separator is freed
    - Limited info leak
    - Calls selector on freed NSObject

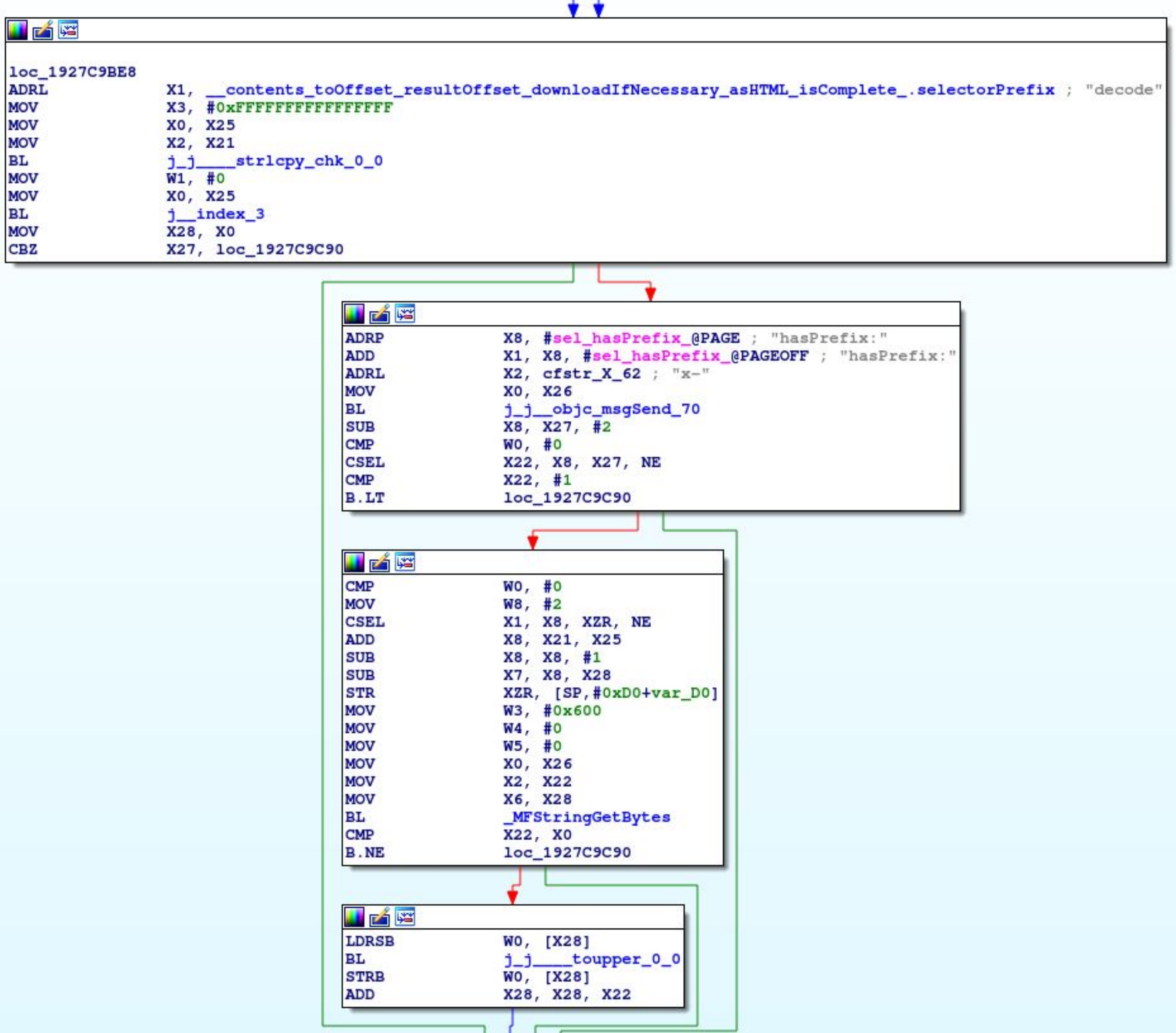
Email Client

- Apple native email client processes incoming messages without user interaction
- Email client must be set up
    - Usage unclear
- Message contents partly controllable by the email sender
    - Filtering can vary by provider
- **`/PrivateFrameworks/MIME.framework/MIME` in `dyld_shared_cache`**

Email Client

- Reviewed in IDA
- Sent malformed MIME messages over SMTP with Python
- Found one vulnerability
    - Looks exploitable in 11.3
    - DOS only in 12

CVE-2019-8626



```
loc_1927C9BE8
ADRL            X1, __contents_toOffset_resultOffset_downloadIfNecessary_asHTML_isComplete_.selectorPrefix ; "decode"
MOV             X3, #0xFFFFFFFFFFFFFFFF
MOV             X0, X25
MOV             X2, X21
BL              j_j____strlcpy_chk_0_0
MOV             W1, #0
MOV             X0, X25
BL              j__index_3
MOV             X28, X0
CBZ             X27, loc_1927C9C90
```

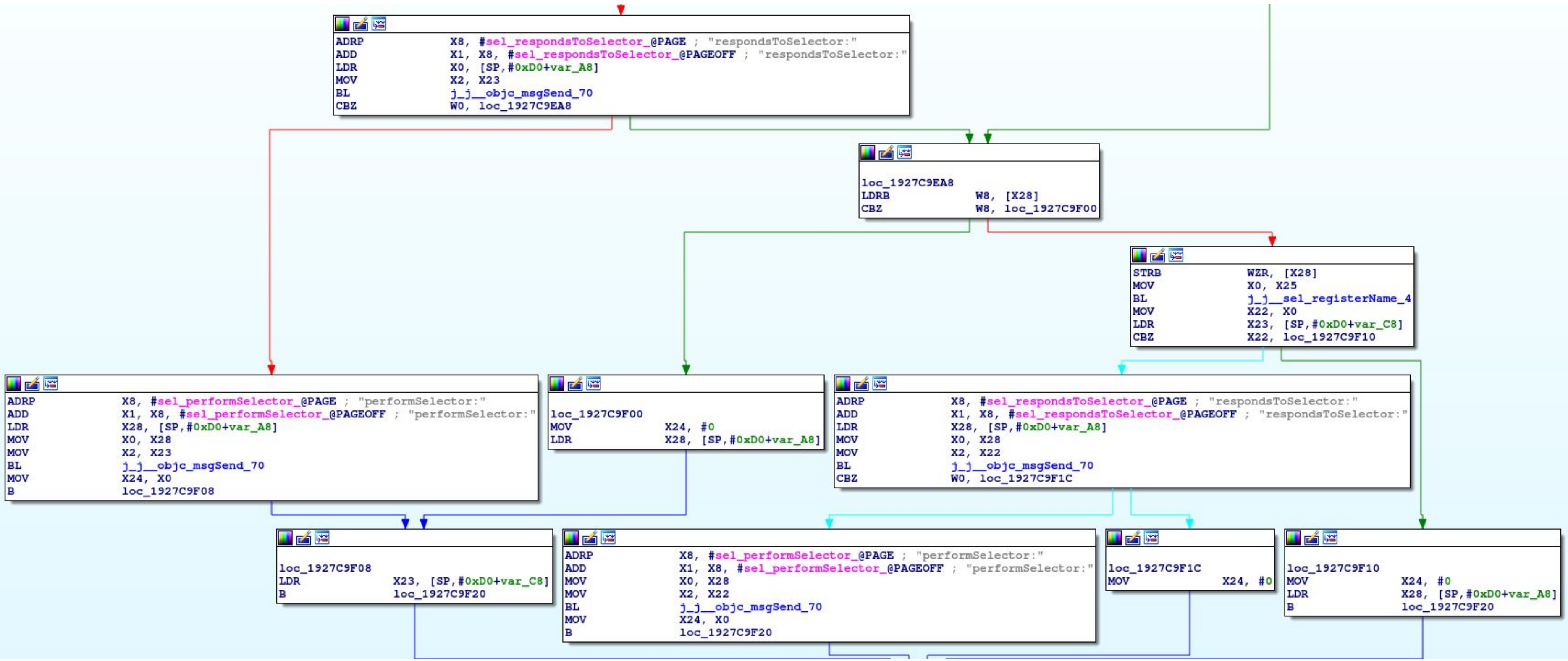
```
ADRP            X8, #sel_hasPrefix_@PAGE ; "hasPrefix:"
ADD             X1, X8, #sel_hasPrefix_@PAGEOFF ; "hasPrefix:"
ADRL            X2, cfstr_X_62 ; "x-"
MOV             X0, X26
BL              j_j__objc_msgSend_70
SUB             X8, X27, #2
CMP             W0, #0
CSEL            X22, X8, X27, NE
CMP             X22, #1
B.LT            loc_1927C9C90
```

```
CMP             W0, #0
MOV             W8, #2
CSEL            X1, X8, XZR, NE
ADD             X8, X21, X25
SUB             X8, X8, #1
SUB             X7, X8, X28
STR             XZR, [SP,#0xD0+var_D0]
MOV             W3, #0x600
MOV             W4, #0
MOV             W5, #0
MOV             X0, X26
MOV             X2, X22
MOV             X6, X28
BL              _MFStringGetBytes
CMP             X22, X0
B.NE            loc_1927C9C90
```

```
LDRSB           W0, [X28]
BL              j_j____toupper_0_0
STRB            W0, [X28]
ADD             X28, X28, X22
```

Google

CVE-2019-8626



```
ADRP            X8, #sel_respondsToSelector_@PAGE ; "respondsToSelector:"
ADD             X1, X8, #sel_respondsToSelector_@PAGEOFF ; "respondsToSelector:"
LDR             X0, [SP,#0xD0+var_A8]
MOV             X2, X23
BL              j_j__objc_msgSend_70
CBZ             W0, loc_1927C9EA8
```

```
loc_1927C9EA8
LDRB            W8, [X28]
CBZ             W8, loc_1927C9F00
```

```
STRB            WZR, [X28]
MOV             X0, X25
BL              j_j__sel_registerName_4
MOV             X22, X0
LDR             X23, [SP,#0xD0+var_C8]
CBZ             X22, loc_1927C9F10
```

```
ADRP            X8, #sel_performSelector_@PAGE ; "performSelector:"
ADD             X1, X8, #sel_performSelector_@PAGEOFF ; "performSelector:"
LDR             X28, [SP,#0xD0+var_A8]
MOV             X0, X28
MOV             X2, X23
BL              j_j__objc_msgSend_70
MOV             X24, X0
B               loc_1927C9F08
```

```
loc_1927C9F00
MOV             X24, #0
LDR             X28, [SP,#0xD0+var_A8]
```

```
ADRP            X8, #sel_respondsToSelector_@PAGE ; "respondsToSelector:"
ADD             X1, X8, #sel_respondsToSelector_@PAGEOFF ; "respondsToSelector:"
LDR             X28, [SP,#0xD0+var_A8]
MOV             X0, X28
MOV             X2, X22
BL              j_j__objc_msgSend_70
CBZ             W0, loc_1927C9F1C
```

```
loc_1927C9F08
LDR             X23, [SP,#0xD0+var_C8]
B               loc_1927C9F20
```

```
ADRP            X8, #sel_performSelector_@PAGE ; "performSelector:"
ADD             X1, X8, #sel_performSelector_@PAGEOFF ; "performSelector:"
MOV             X0, X28
MOV             X2, X22
BL              j_j__objc_msgSend_70
MOV             X24, X0
B               loc_1927C9F20
```

```
loc_1927C9F1C
MOV             X24, #0
```

```
loc_1927C9F10
MOV             X24, #0
LDR             X28, [SP,#0xD0+var_A8]
B               loc_1927C9F20
```

iMessage

- Can send iMessage to email or phone number
- Both Mac and iPhone support iMessage
- Encrypted, peer-to-peer messages
- Many formatting features, including extensions
- Worked with Samuel Groß

Dumping/Sending iMessage Messages

- Samuel Groß wrote iMessage sending and intercepting client
- Used Frida to hook incoming and outgoing messages
  - Works on Mac and iPhone

iMessage Format (binary plist)

```
to: mailto:TARGET@gmail.com
from: tel:+15556667777
{
    gid = "FAA29682-27A6-498D-8170-CC92F2077441";
    gv = 8;
    p =     (
        "tel:+15556667777",
        "mailto:TARGET@gmail.com"
    );
    pv = 0;
    r = "68DF1E20-9ABB-4413-B86B-02E6E6EB9DCF";
    t = "Hello Black Hat";
    v = 1;
}
```

Important iMessage Properties

t	**Plain text message content**
bid	**“Balloon identifier” for plugin**
bp	**Plugin data**
ati	**Attribution info**
p	**Participants**

iMessage Serialization

- bp and ati fields are serialized using NSKeyedArchiver/NSKeyedUnarchiver
- NSKeyedUnarchiver deserialization format is a plist containing dictionaries with class and other properties
- Objects are created by calling [DECODED_CLASS initWithCoder:], which processes other properties
  - Several past bugs

iMessage Serialization

```
<dict>
	<key>$class</key>
	<dict>
		<key>CF$UID</key>
		<integer>7</integer>
	</dict>
	<key>NS.base</key>
	<dict>
		<key>CF$UID</key>
		<integer>0</integer>
	</dict>
	<key>NS.relative</key>
	<dict>
		<key>CF$UID</key>
		<integer>6</integer>
	</dict>
</dict>
```

```
<string>http://natashenka.ca</string>
<dict>
	<key>$classes</key>
	<array>
		<string>NSURL</string>
		<string>NSObject</string>
	</array>
	<key>$classname</key>
	<string>NSURL</string>
</dict>
```

iMessage Serialization Security Features

- NSSecureCoding
  - Requires class to implement a specific method (that cannot be inherited) for its initWithCoder: to be generally available
  - Avoids accidental initWithCoder: exposure
  - Requires list of allowed classes to be provided while decoding *recursively*

Secure versus Insecure Decoding

- Safe
  - initForReadingFromData:
  - unarchivedObjectOfClasses:fromData:error:
- Unsafe
  - initWithData:
  - unarchiveObjectWithData:error
  - initForReadingWithData:

Google

Secure versus Insecure Decoding

- Safe
  - **initForReadingFromData:**
  - unarchivedObjectOfClasses:fromData:error:
- Unsafe
  - initWithData:
  - unarchiveObjectWithData:error
  - **initForReadingWithData:**

Secure versus Insecure Decoding

- Safe
  - **initForReadingFromData:**
  - unarchivedObjectOfClasses:fromData:error:
- Unsafe
  - initWithData:
  - unarchiveObjectWithData:error
  - **initForReadingWithData:**

Where does deserialization happen?

- In SpringBoard, for **bp**
  - SpringBoard can also call _previewText for extensions
  - Practically, only Link Presentation supports this
  - SpringBoard is unsandboxed
- In MobileSMS, for **bp** (but requires one click)
- In imagent, for **ati**

Google

Idea 1

Find an insecure deserialization call and create a WebKit instance

- Did not find any insecure calls in SpringBoard or imagent

Idea 2

Find an extension that misuses a deserialized object

- CVE-2019-8624 -- out-of-bounds read in DigitalTouch tap message processing
  - Code handling deserialized objects trusts length field over byte array length
  - Very low-quality bug

Idea 2

Find an extension that misuses a deserialized object

- Looked at Link Presentation layer for use of WebKit instances, but does not seem to load received URLs

Idea 3

Find a bug in supported deserialization code

- Reviewed all available initWithCoder: implementations

Which initWithCoder: implementations are available?

- Classes in allowed class list and their subclasses
  - NSDictionary, NSString, NSData, NSNumber, NSURL, NSUUID, NSValue for messaging generally
  - Must support secure coding
- Libraries loaded by the process
  - Not the entire dyld_shared_cache

Idea 3

Find a bug in supported deserialization code

- CVE-2019-8661 -- heap overflow when deserializing URL
- Mac only

CVE-2019-8661

- [NSURL initWithCoder:] supports several decoding methodologies, including decoding a bookmark from a byte array
- On Mac, bookmarks can include alias files, which have a buggy decoder (CarbonCore)
- Bookmarks are never used by iMessage legitimately

Idea 3

Find a bug in supported deserialization code

- CVE-2019-8646 -- NSKeyedUnarchiver deserialization allows file backed NSData objects
- Remote info leak and file access!

CVE-2019-8646

- _NSDataFileBackedFuture subclasses NSData
  - Private class
- Two problems:
  - Trusts deserialized length, even though file could be shorter
  - Can bypass check that URL is local file

CVE-2019-8646

1) Create NSData with local file
2) Append NSData to NSURL
3) Use bug again to visit new NSURL
4) URL parameters contain leaked file or memory

Idea 4

Wait, what happens if a class subclasses an allowed class but doesn't extend initWithCoder?!?!

- Regular inheritance rules apply
  - e.g. different initWithCapacity implementation could get called
  - Some direct inheritance checks, especially in placeholders

Idea 4

Wait, what happens if a class subclasses an allowed class but doesn’t extend initWithCoder?!?!

- CVE-2019-8647 -- NSArray deserialization can invoke subclass that does not retain references
  - [_PFArray initWithObjects:count:] is a private method which should only get called when objects are appropriate

Idea 5

What if an object has cycles in it?

- Deserialization gets complicated

NSKeyedArchiver Object caching

```
NSObject* a = [NSSomeClass alloc];
temp_dict[key] = a; //No references!!
NSObject* obj = [a initWithCoder:];
temp_dict[key] = NIL;
obj_dict[key] = obj;
return obj;
```

Google

NSKeyedArchiver Object caching

```
if(temp_dict[key])
   return [temp_dict[key] copy];
if(obj_dict[key])
   return [obj_dict[key] copy];
NSObject* a = [NSSomeClass alloc];
temp_dict[key] = a; //No references!!
NSObject* obj = [a initWithCoder:];
temp_dict[key] = NIL;
obj_dict[key] = obj;
return obj;
```

Problems with cycles

- Object can be used before initWithCoder: is complete
- initWithCoder: isn’t guaranteed to return object created by alloc
- temp_dict has no references
    - What if object returned by alloc is released by initWithCoder: ?*

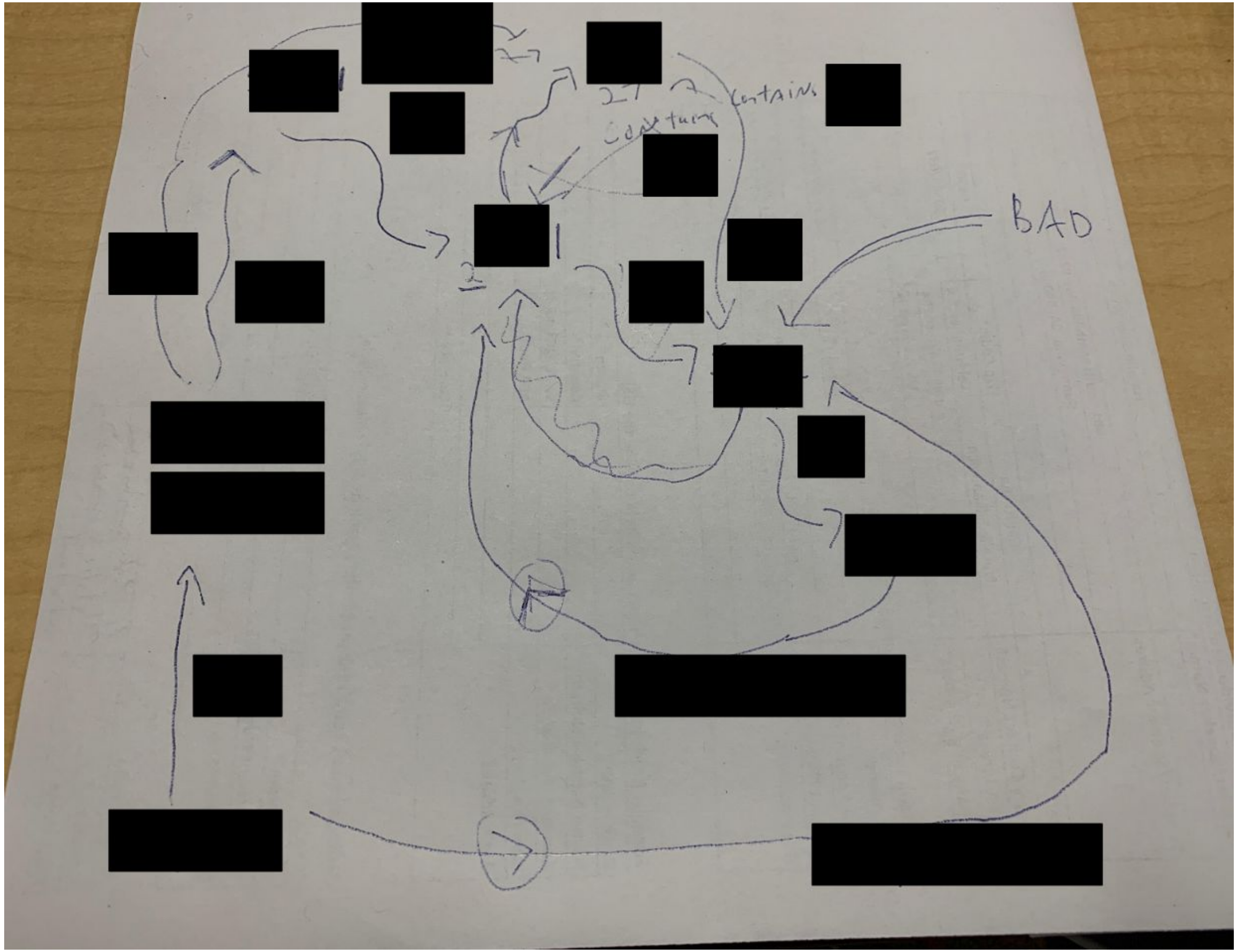
* The docs say doing this is okay

Idea 5

What if an object has cycles in it?

- CVE-2019-8641 -- decoding CLASS can read object out of bounds
  - Buffer length is calculated based on a singly linked list
  - If initWithCoder: isn't finished, the list isn't complete
  - Buffer is too short

Google


[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]
1
2
CONTAINS
Contains
2
1
BAD

Idea 5

What if an object has cycles in it?

- CVE-2019-8660 -- memory corruption when decoding NSKnownKeysDictionary1
    - Length of key data is deserialized separately from data
    - New buffer length is calculated with deserialized length
    - Length consistency is checked after the object can be used in a cycle

What’s the attack surface of an NSURL

```
NSURL* myurl = [NSKeyedUnarchiver
unarchivedObjectOfClasses:@[NSURL]
fromData:mydata error:NIL];
```

```
clang app.m -fobjc-arc -framework
UserNotifications
```

Google

What’s the attack surface of an NSURL?

- **`[NSURL initWithCoder:]`**
    - Top level class
- **`[MyURLSubClass initWithCoder:]`**
    - App-defined subclass
- **`[UNSecurityScopedURL initWithCoder:]`**
    - Subclass from UserNotifications framework

What’s the attack surface of an NSURL?

```
[NSURL initWithCoder:](NSURL *u, id decoder){
   NSData* book = [decoder decodeObjectOfClass:[NSData class]
forKey:@"NS.minimalBookmarkData"];
   if(book)
      return [URLByResolvingBookmarkData:data];
   NSString* base = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.base"];
   NSString* relative = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.relative"];
   return [NSURL initWithString:base relativeToURL:relative];
}
```

What’s the attack surface of an NSURL?

```
[NSURL initWithCoder:](NSURL *u, id decoder){
    NSData* book = [decoder decodeObjectOfClass:[NSData class]
forKey:@"NS.minimalBookmarkData"];
    if(book)
        return [URLByResolvingBookmarkData:data];
    NSString* base = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.base"];
    NSString* relative = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.relative"];
    return [NSURL initWithString:base relativeToURL:relative];
}
```

What's the attack surface of an NSURL?

```
[NSURL initWithCoder:](NSURL *u, id decoder){
    NSData* book = [decoder decodeObjectOfClass:[NSData class]
forKey:@"NS.minimalBookmarkData"];
    if(book)
        return [URLByResolvingBookmarkData:data];
    NSString* base = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.base"];
    NSString* relative = [decoder decodeObjectOfClass:[NSString
class] forKey:@"NS.relative"];
    return [NSURL initWithString:base relativeToURL:relative];
}
```

What’s the attack surface of an NSURL?

- Bookmark parsing
- **`[_NSDispatchData initWithCoder:]`, `[__NSLocalizedString initWithCoder:]`, `[NSLocalizableString initWithCoder:]`,`[UNLocalizedString initWithCoder:]`**
    - Subclasses of NSString and NSData in Foundation and UserNotification framework

What’s the attack surface of an NSURL?

- Etc.
    - Continue down initWithCoder: implementations
    - `[UNLocalizedString initWithCoder:]` decodes an NSArray
    - `[__NSLocalizedString initWithCoder:]` decodes a NSDictionary, an NSDate and an NSNumber

What’s the attack surface of an NSURL?

- `[NSBigMutableString initWithString:], [NSDebugString initWithString:], [NSPlaceholderMutableString initWithBytes:length:encoding:], [NSPlaceholderString initWithBytes ...]`
  - Classes from Foundation, CoreFoundation and UserNotifications with initWithString/initWithBytes
  - Similar for NSArray, NSDictionary, NSDate, NSNumber and any classes they decode

What’s the attack surface of an NSURL?

- Legitimate URLs almost certainly contain one instance of NSString

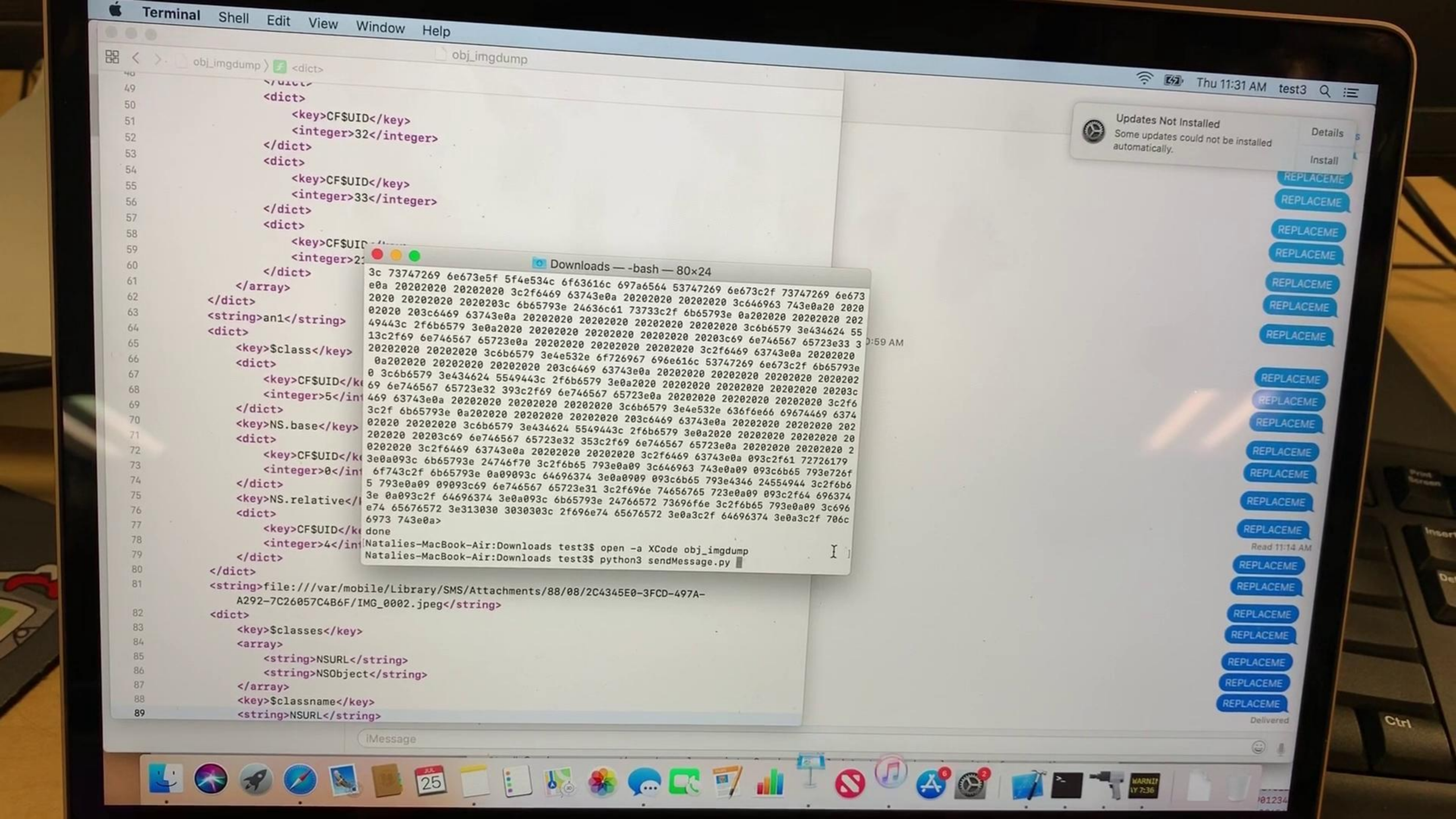
Securing Deserialization

- Imagine adding a few extra allowed classes
- Imagine importing a few more libraries
- Imagine being a developer trying to secure this

NSKeyedArchiver serialization cannot be secure

- **Securing a class in the face of NSKeyedArchiver is an intractable problem**
  - There are too many interdependencies between unrelated components
  - Requires full knowledge of all other components
  - Makes small changes to low-risk components have unexpected consequences

Demo



```
            <dict>
                <key>CF$UID</key>
                <integer>32</integer>
            </dict>
            <dict>
                <key>CF$UID</key>
                <integer>33</integer>
            </dict>
            <dict>
                <key>CF$UID
                <integer>2
            </dict>
        </array>
    </dict>
    <string>an1</string>
    <dict>
        <key>$class</key>
        <dict>
            <key>CF$UID</k
            <integer>5</in
        </dict>
        <key>NS.base</key>
        <dict>
            <key>CF$UID</k
            <integer>0</in
        </dict>
        <key>NS.relative</
        <dict>
            <key>CF$UID</k
            <integer>4</in
        </dict>
    </dict>
    <string>file:///var/mobile/Library/SMS/Attachments/88/08/2C4345E0-3FCD-497A-
        A292-7C26057C4B6F/IMG_0002.jpeg</string>
    <dict>
        <key>$classes</key>
        <array>
            <string>NSURL</string>
            <string>NSObject</string>
        </array>
        <key>$classname</key>
        <string>NSURL</string>
```

Downloads — -bash — 80×24

```
3c 73747269 6e673e5f 5f4e534c 6f63616c 697a6564 53747269 6e673c2f 73747269 6e673
e0a 20202020 20202020 3c2f6469 63743e0a 20202020 20202020 3c646963 743e0a20 2020
2020 20202020 2020203c 6b65793e 24636c61 73733c2f 6b65793e 0a202020 20202020 202
02020 203c6469 63743e0a 20202020 20202020 20202020 20202020 3c6b6579 3e434624 55
49443c 2f6b6579 3e0a2020 20202020 20202020 20202020 20203c69 6e746567 65723e33 3
13c2f69 6e746567 65723e0a 20202020 20202020 20202020 3c2f6469 63743e0a 20202020
20202020 20202020 3c6b6579 3e4e532e 6f726967 696e616c 53747269 6e673c2f 6b65793e
 0a202020 20202020 20202020 203c6469 63743e0a 20202020 20202020 20202020 2020202
0 3c6b6579 3e434624 5549443c 2f6b6579 3e0a2020 20202020 20202020 20202020 20203c
69 6e746567 65723e32 393c2f69 6e746567 65723e0a 20202020 20202020 20202020 3c2f6
469 63743e0a 20202020 20202020 20202020 3c6b6579 3e4e532e 636f6e66 69674469 6374
3c2f 6b65793e 0a202020 20202020 20202020 203c6469 63743e0a 20202020 20202020 202
02020 20202020 3c6b6579 3e434624 5549443c 2f6b6579 3e0a2020 20202020 20202020 20
202020 20203c69 6e746567 65723e32 353c2f69 6e746567 65723e0a 20202020 20202020 2
0202020 3c2f6469 63743e0a 20202020 20202020 3c2f6469 63743e0a 093c2f61 72726179
3e0a093c 6b65793e 24746f70 3c2f6b65 793e0a09 3c646963 743e0a09 093c6b65 793e726f
 6f743c2f 6b65793e 0a09093c 64696374 3e0a0909 093c6b65 793e4346 24554944 3c2f6b6
5 793e0a09 09093c69 6e746567 65723e31 3c2f696e 74656765 723e0a09 093c2f64 696374
3e 0a093c2f 64696374 3e0a093c 6b65793e 24766572 73696f6e 3c2f6b65 793e0a09 3c696
e74 65676572 3e313030 3030303c 2f696e74 65676572 3e0a3c2f 64696374 3e0a3c2f 706c
6973 743e0a>
done
Natalies-MacBook-Air:Downloads test3$ open -a XCode obj_imgdump
Natalies-MacBook-Air:Downloads test3$ python3 sendMessage.py
```

REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
Read 11:14 AM
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
REPLACEME
Delivered

Conclusions

- Fully remote iPhone bugs exist
    - 10 bugs total reported
- The remote attack surface includes SMS, MMS, VVM, Email and iMessage
- Design problems with iMessage serialization make it an especially bug prone surface

Conclusions

- The are methods for an attacker to send malformed messages in most formats
- Released tools for remote iOS research: https://github.com/googleprojectzero
- Especially dangerous attack surface

Questions

https://googleprojectzero.blogspot.com/

@natashenka

natashenka@google.com

Google