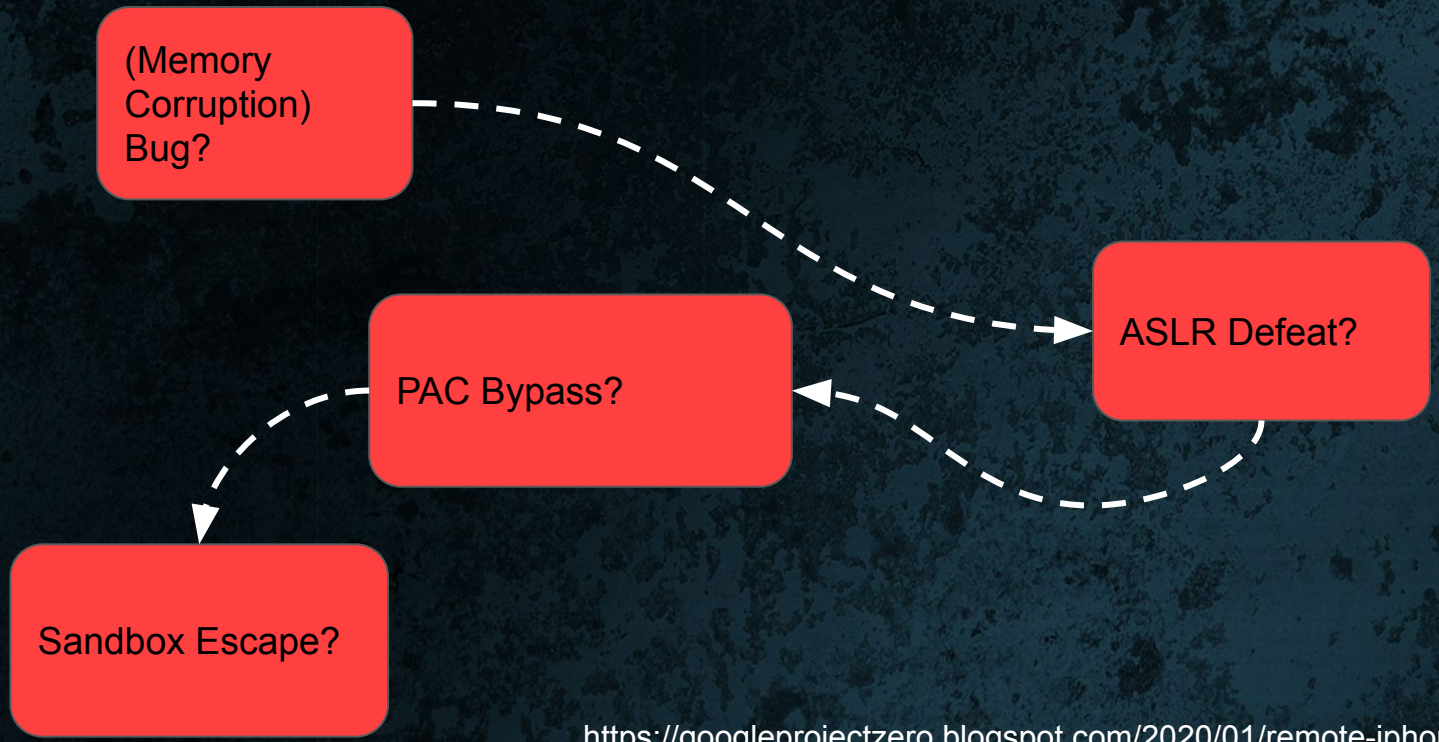

BLUEHAT
IL 2022

A Brief History of iMessage Exploitation

Samuel Groß (@5aelo), Ian Beer (@i41nbeer)

iMessage Exploit Flow ~ 2019

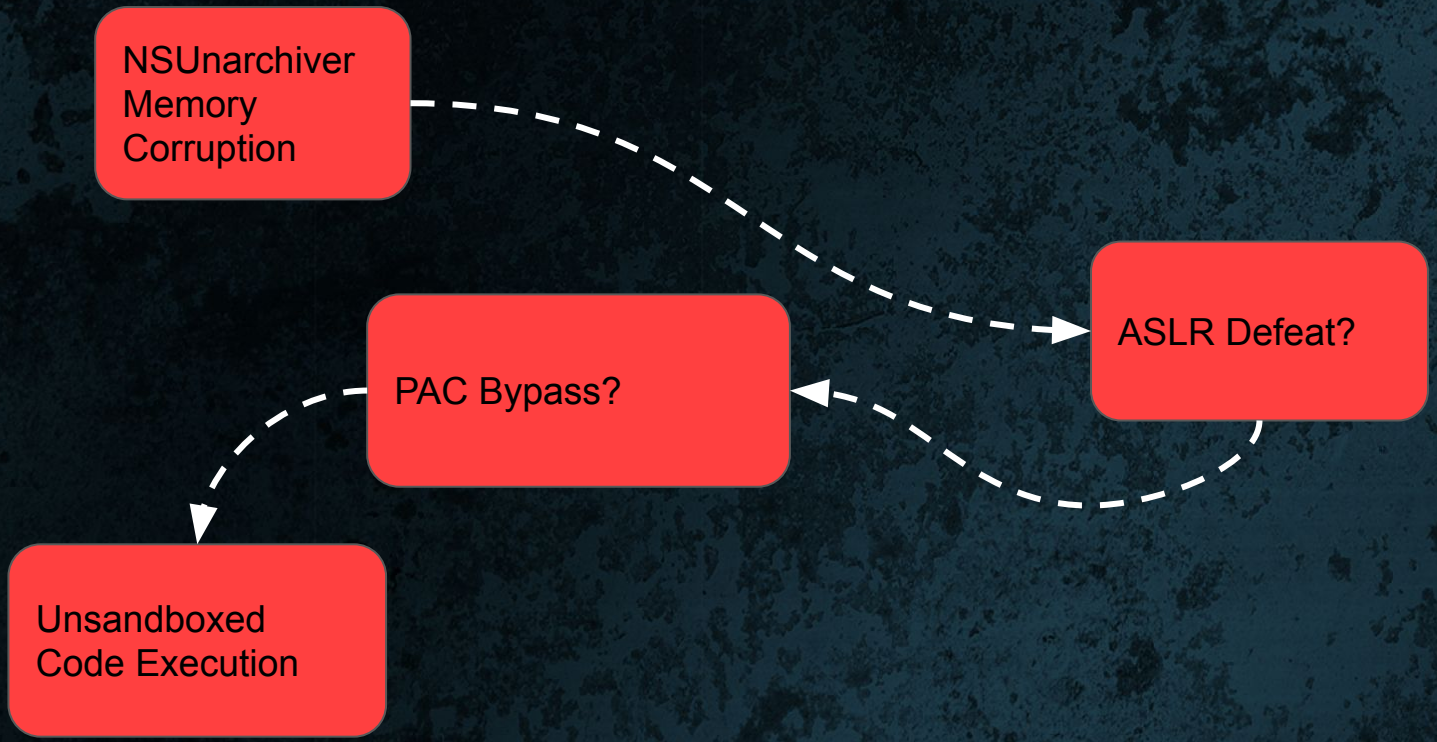


https://googleprojectzero.blogspot.com/2020/01/remote-iphone-exploitation-part-1.html

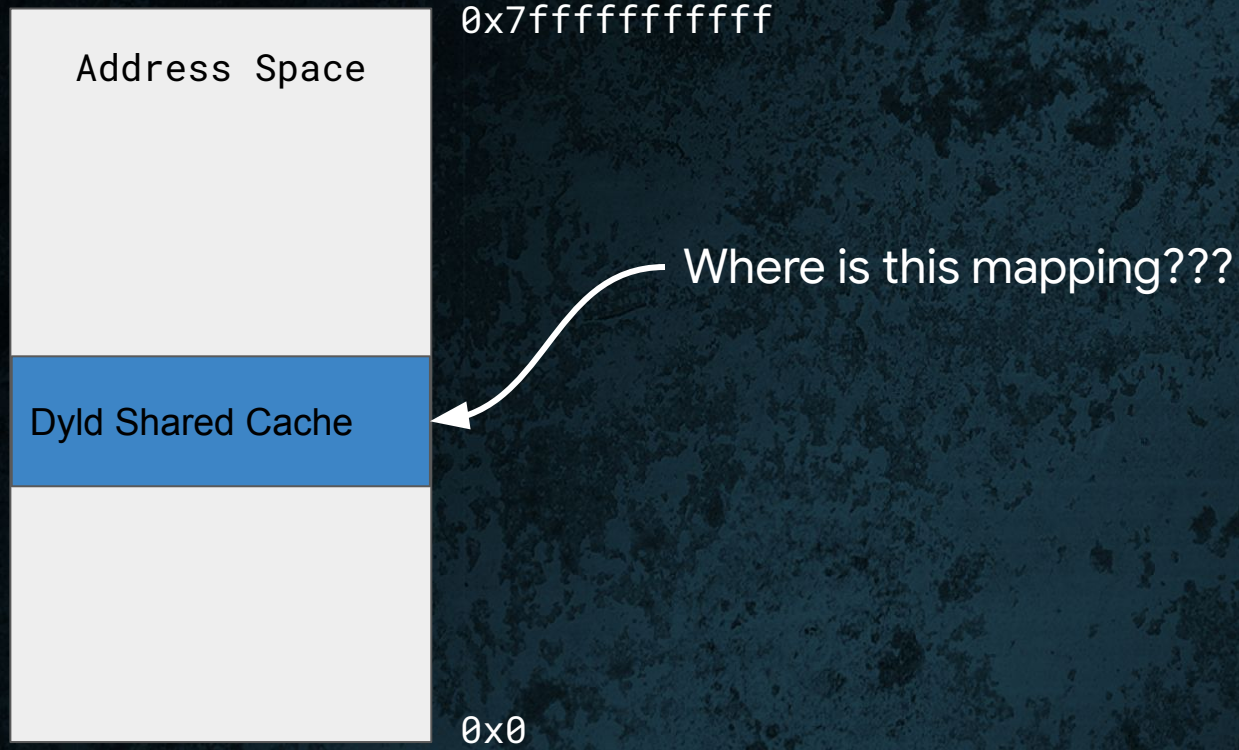
Attack Surface: Deserialization

< TODO iMessage plist data >

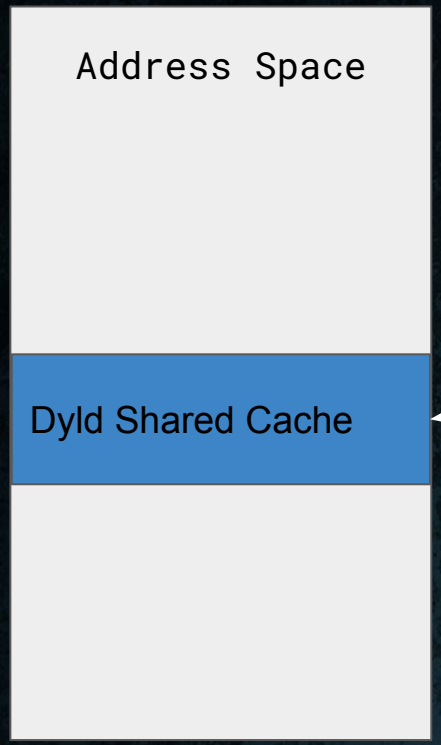
iMessage Exploit Flow ~ 2019



Exploitation (~ 2019): Defeating ASLR



Exploitation (~ 2019): Defeating ASLR

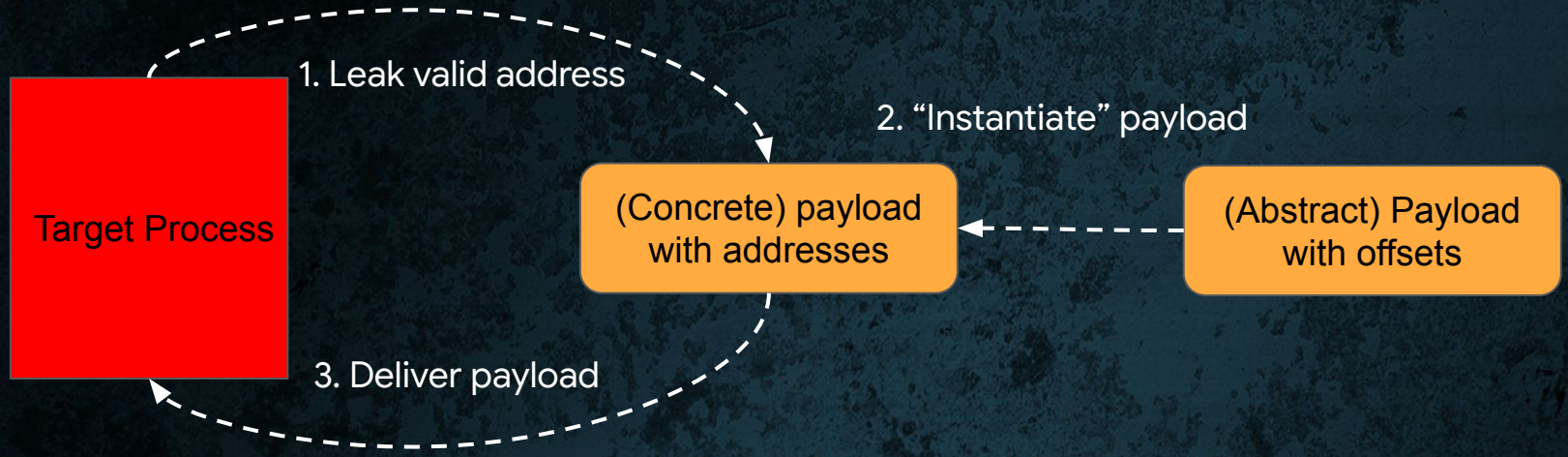


Where is this mapping???

Important: this mapping is only randomized once per-boot...

Why is ASLR a Problem?

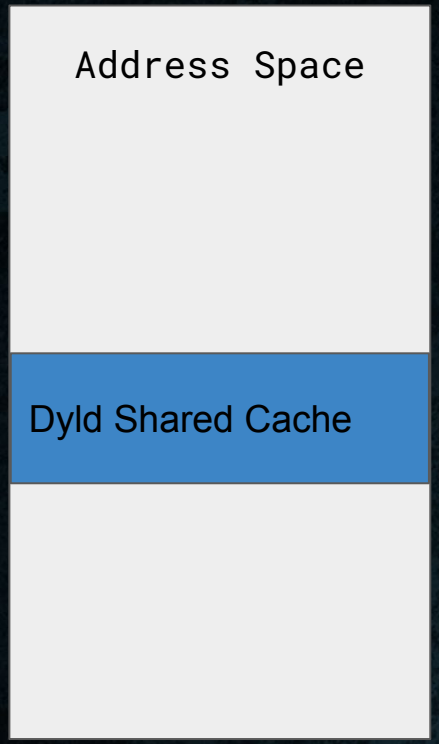
- **Need communication channel between target process and exploit logic**
- Usually no (big) problem for e.g. browser exploits: exploit logic implemented in JavaScript => Runs inside the targeted process
- It is a problem for something like iMessage though...



Delivery Receipts as Communication Channel

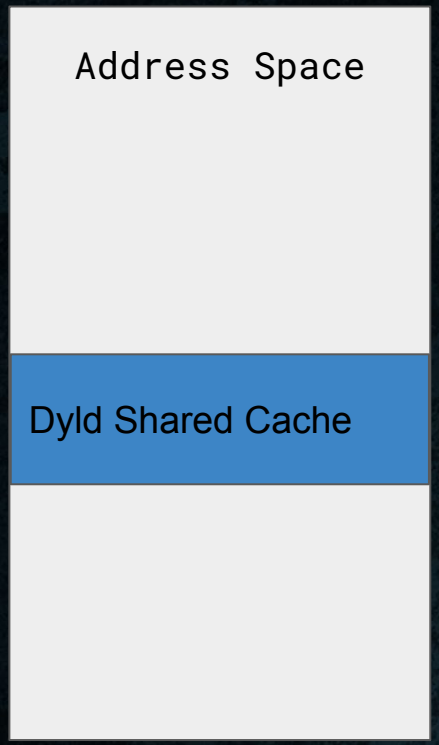
- When iMessage process receives a message, it sends a *delivery receipt* to the sender
- If process crashes before sending the receipt, the delivery receipt message is never sent
- => **1-bit communication channel**: crashed or didn't crash

Crash Oracle + Binary Search = ASLR defeat



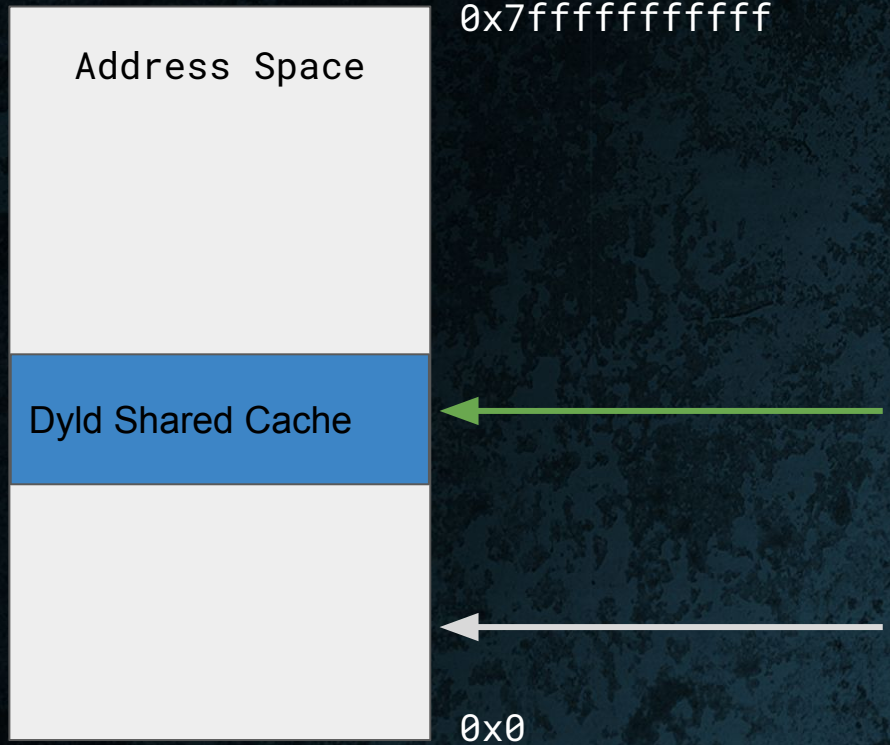
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



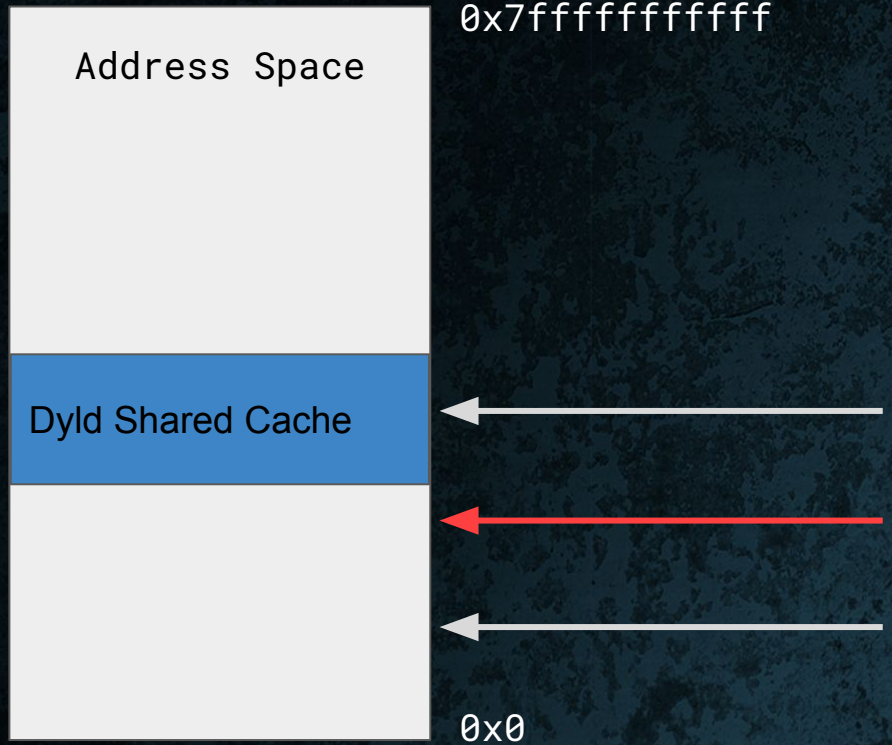
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



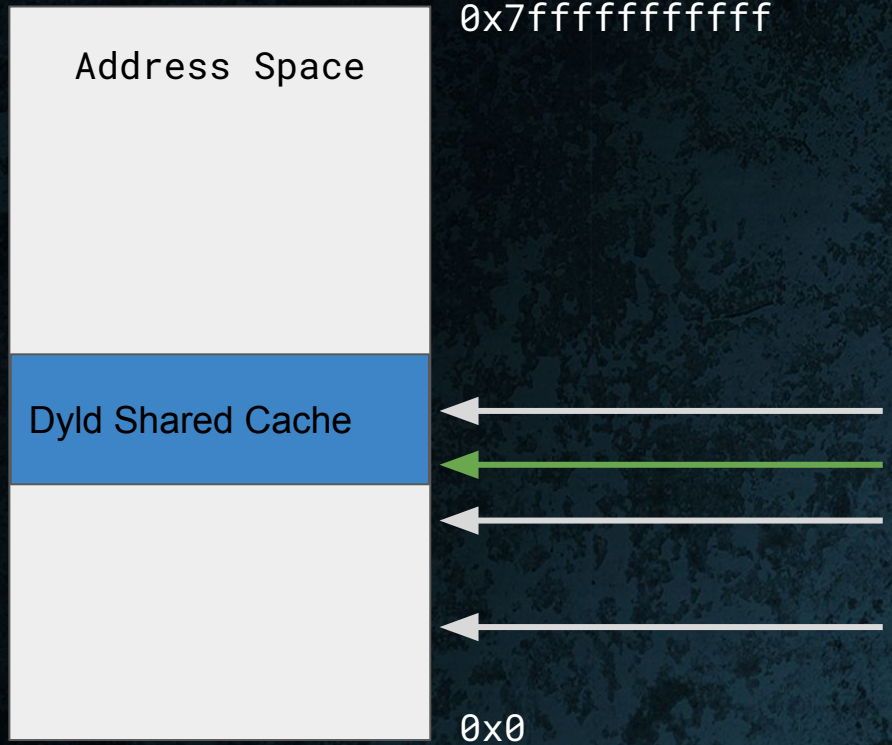
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



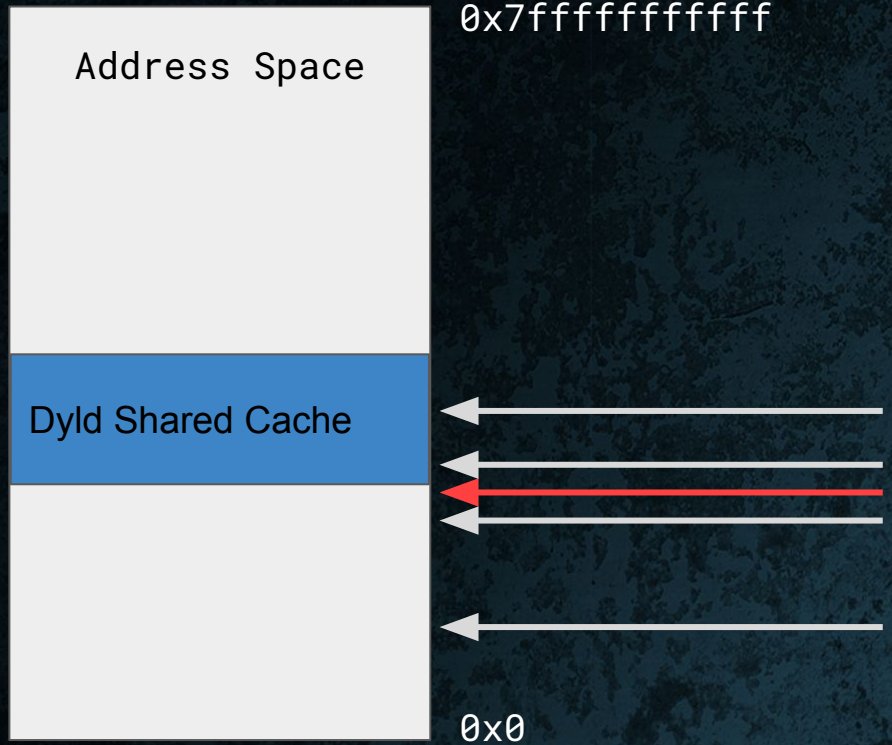
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



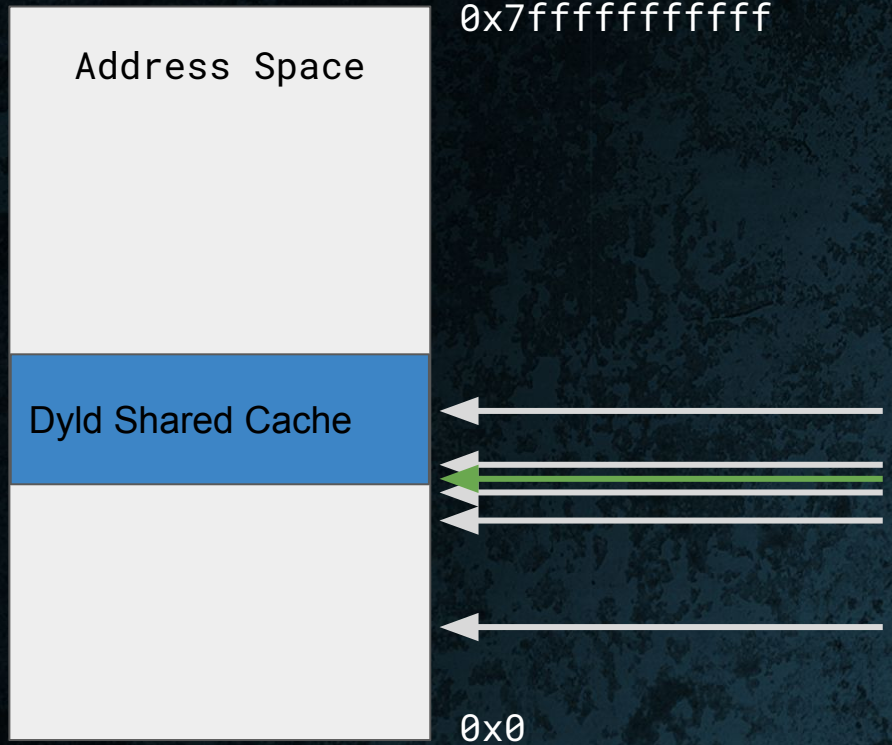
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



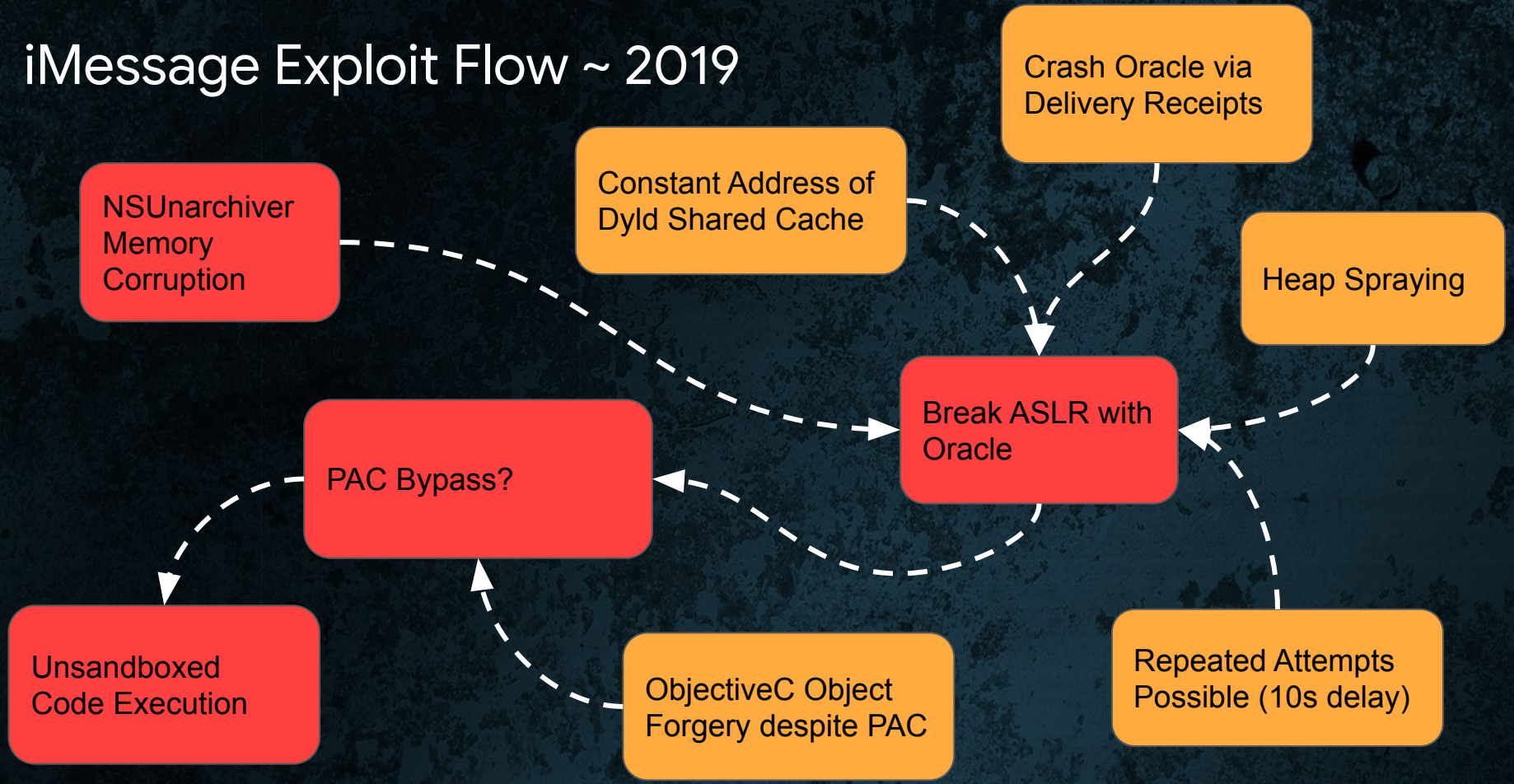
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

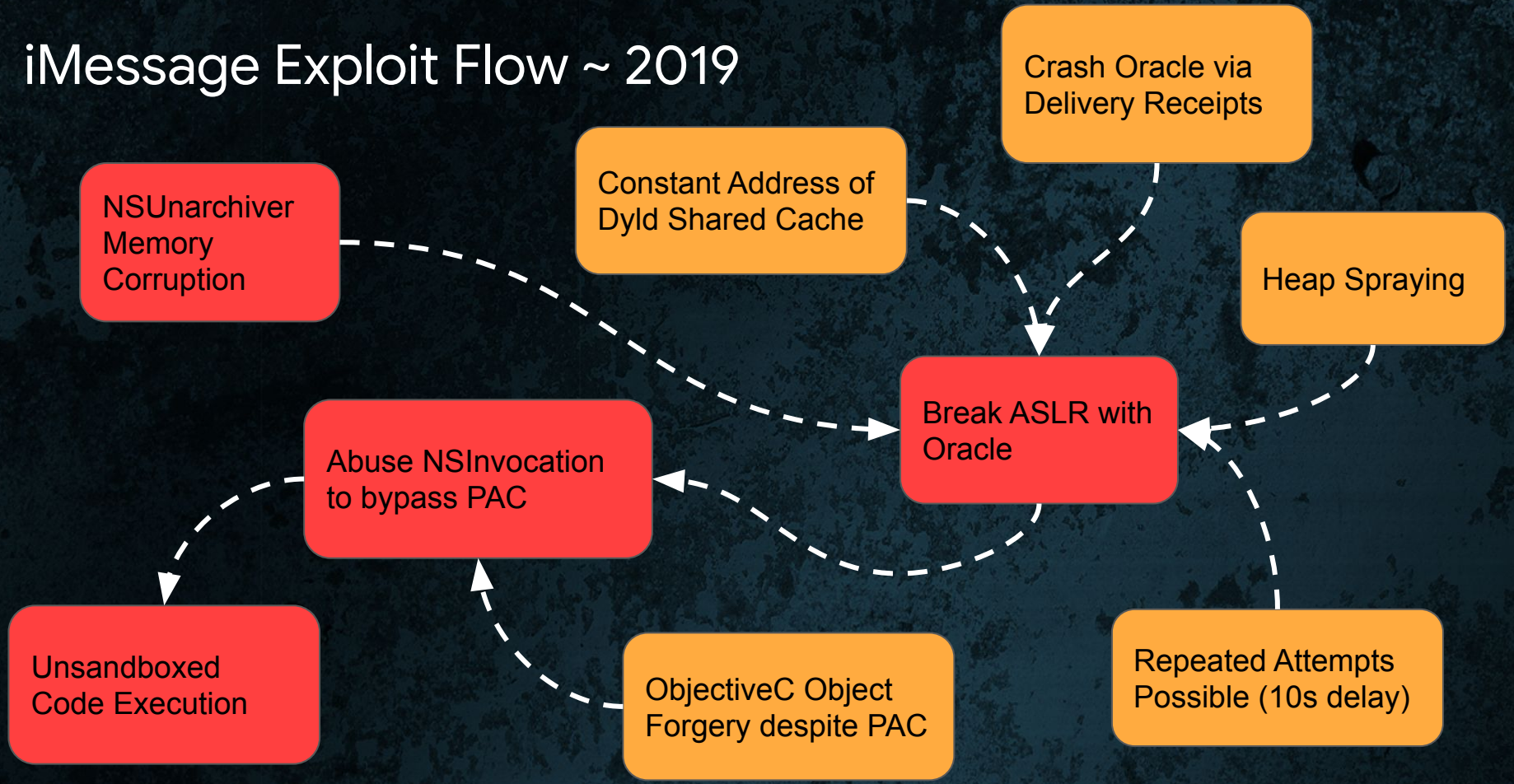
iMessage Exploit Flow ~ 2019



Defeating PAC (Pointer Authentication)

- PAC: cryptographic signature in unused bits of pointer
- Can no longer forge e.g. code pointers => breaks ROP, JOP, ...
- **But really, *arbitrary* code execution isn't necessary**
- (Mostly) enough to call existing functions and method
- TODO

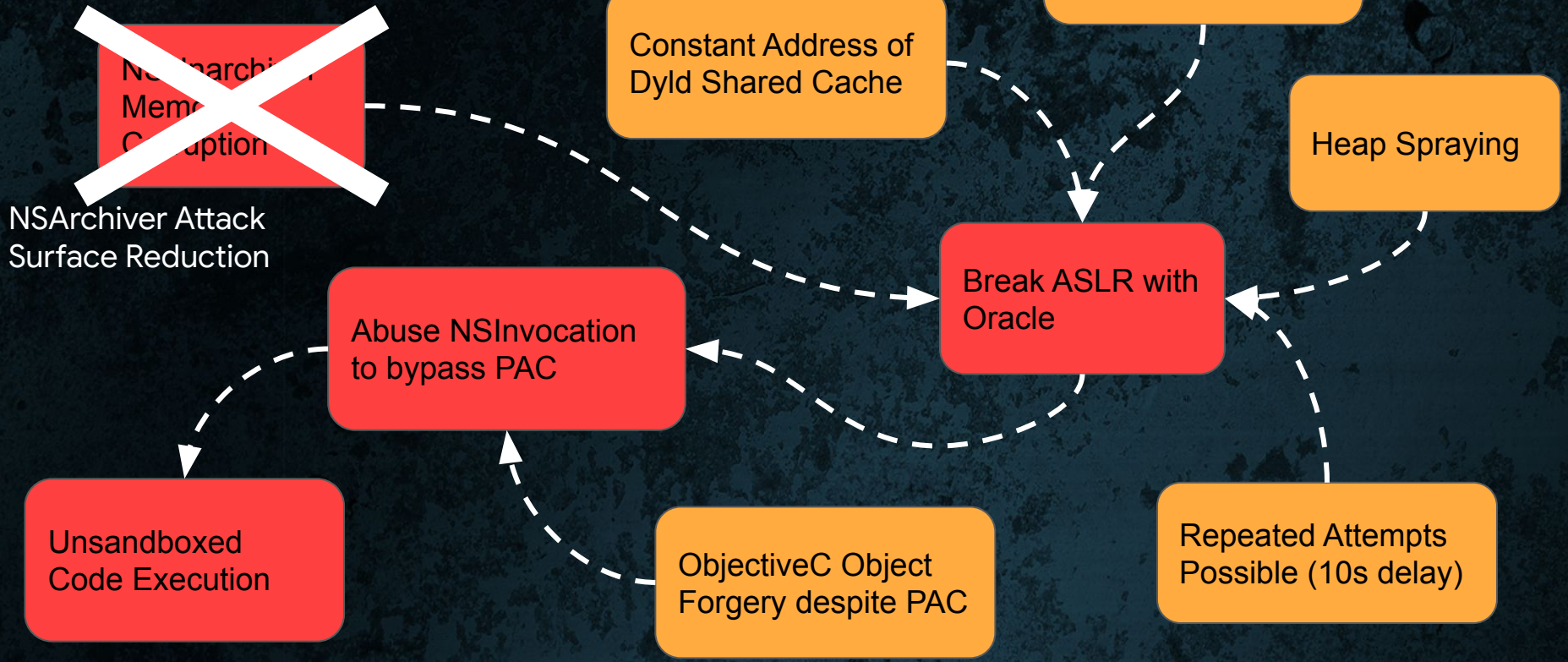
iMessage Exploit Flow ~ 2019



NSKeyedUnarchiver Attack Surface Reduction (~late 2019)

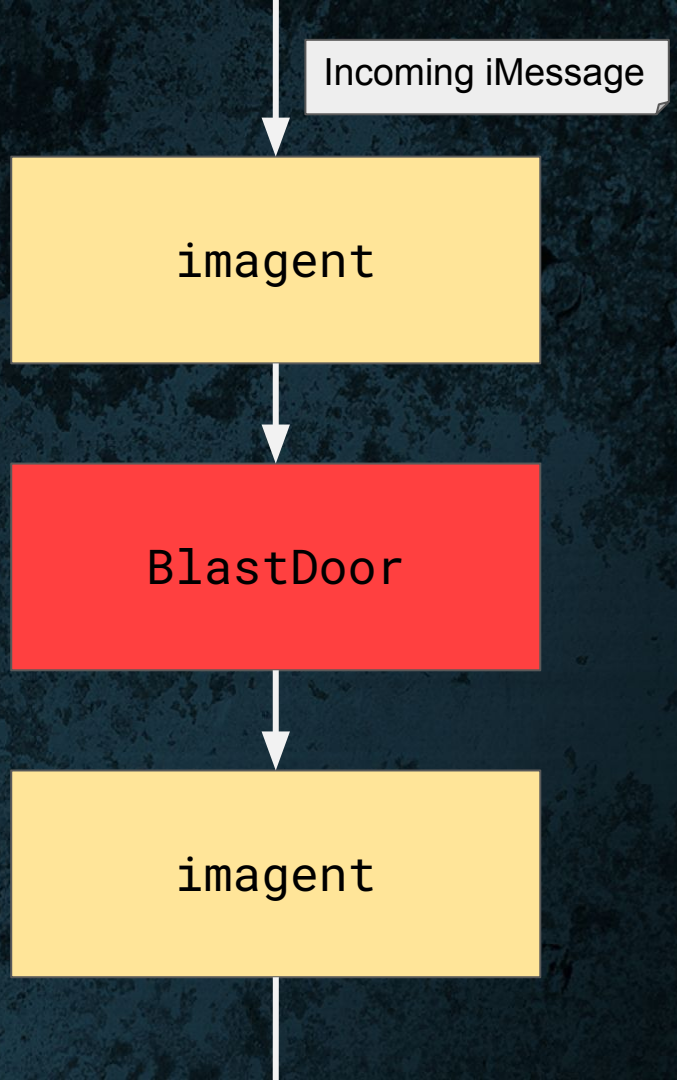
- No longer allow child classes to be deserialized :)

iMessage Exploit Flow ~ 2019

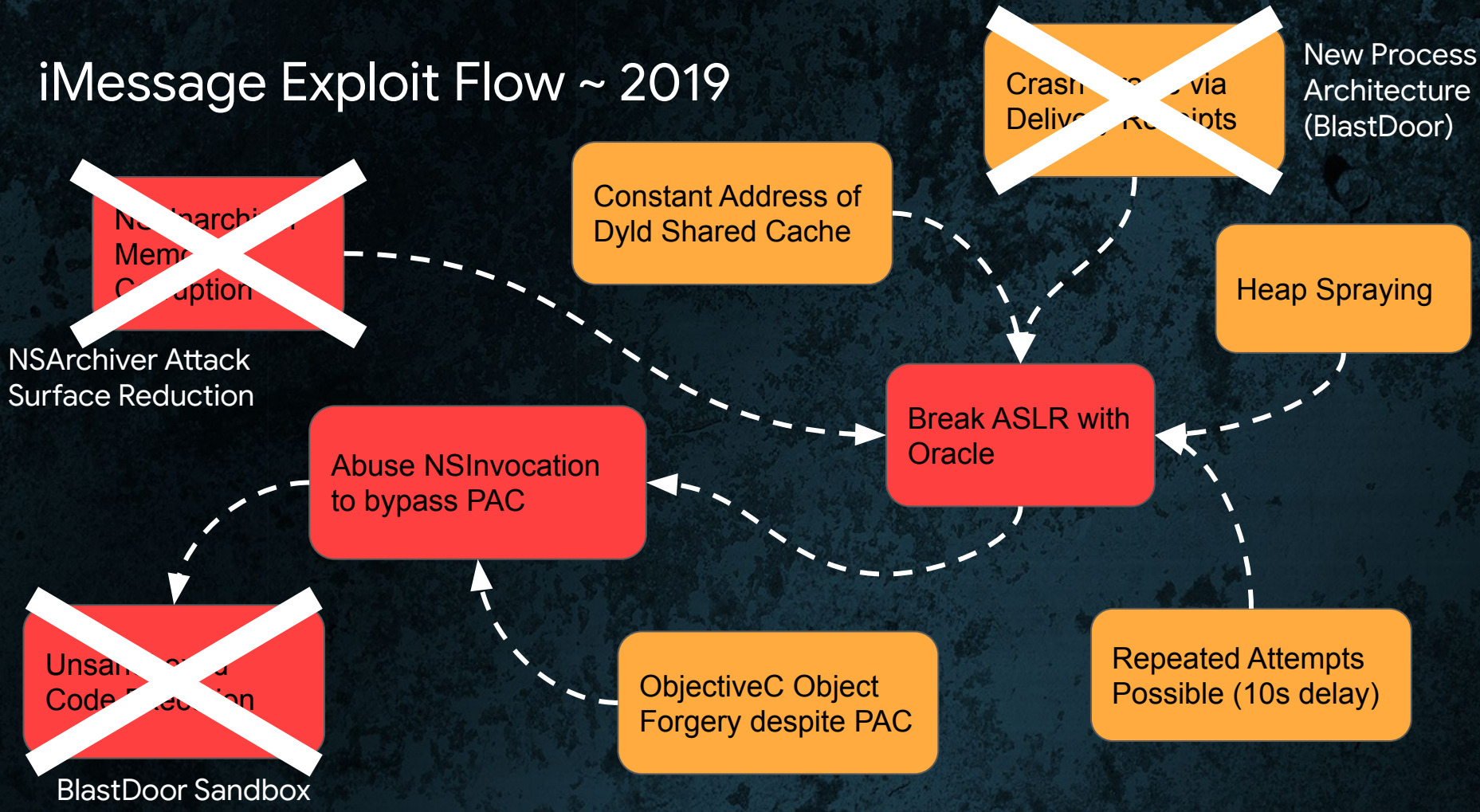


Blastdoor (iOS 14, ~ mid 2020)

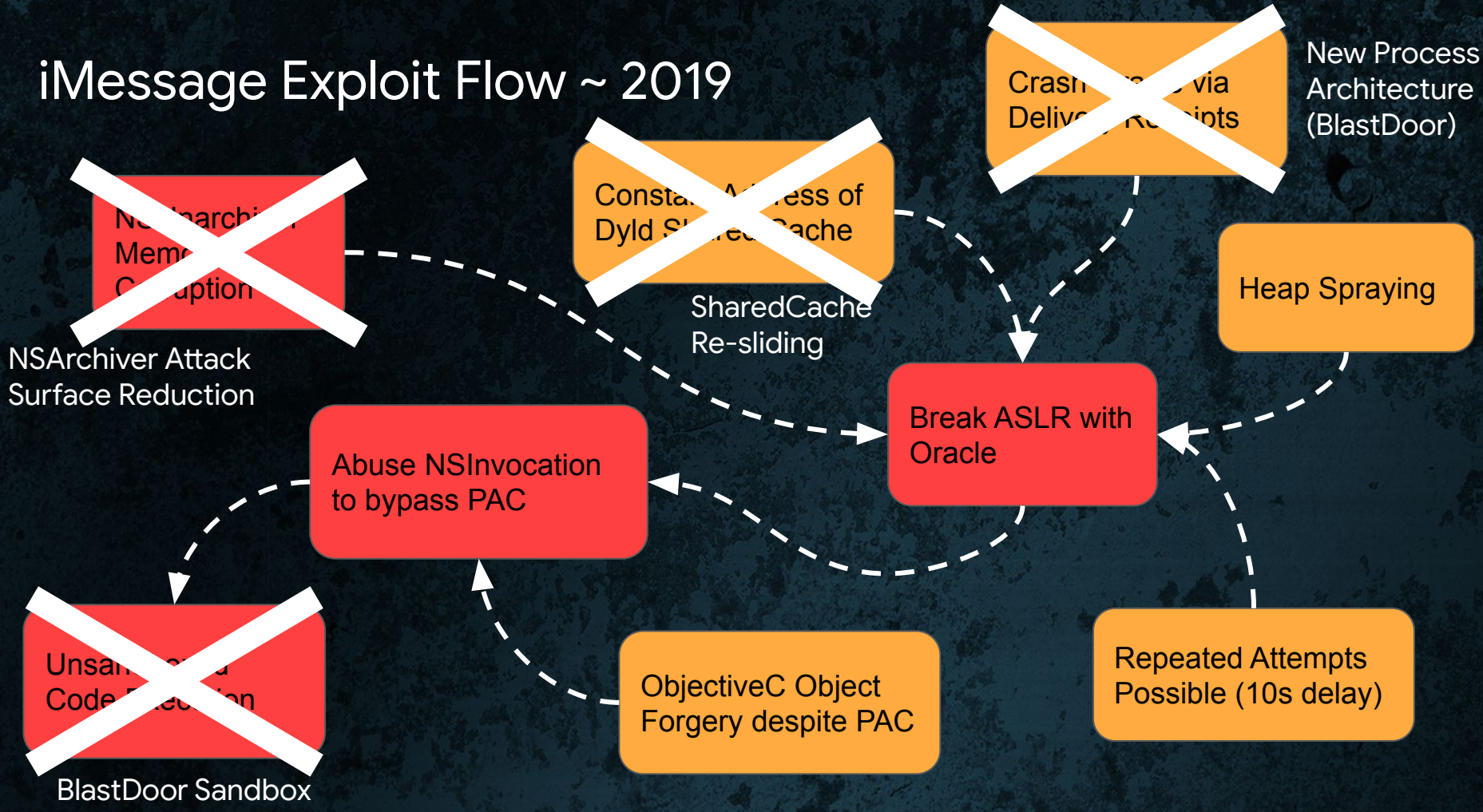
- Re-architectured iMessage processing
- Idea: complex parsing now happens in a tightly sandboxed process: MessagesBlastDoorService
- High-level logic implemented in Swift
- Also breaks crash oracle: crashing process (BlastDoor) is not the process sending the delivery receipt (imagent)



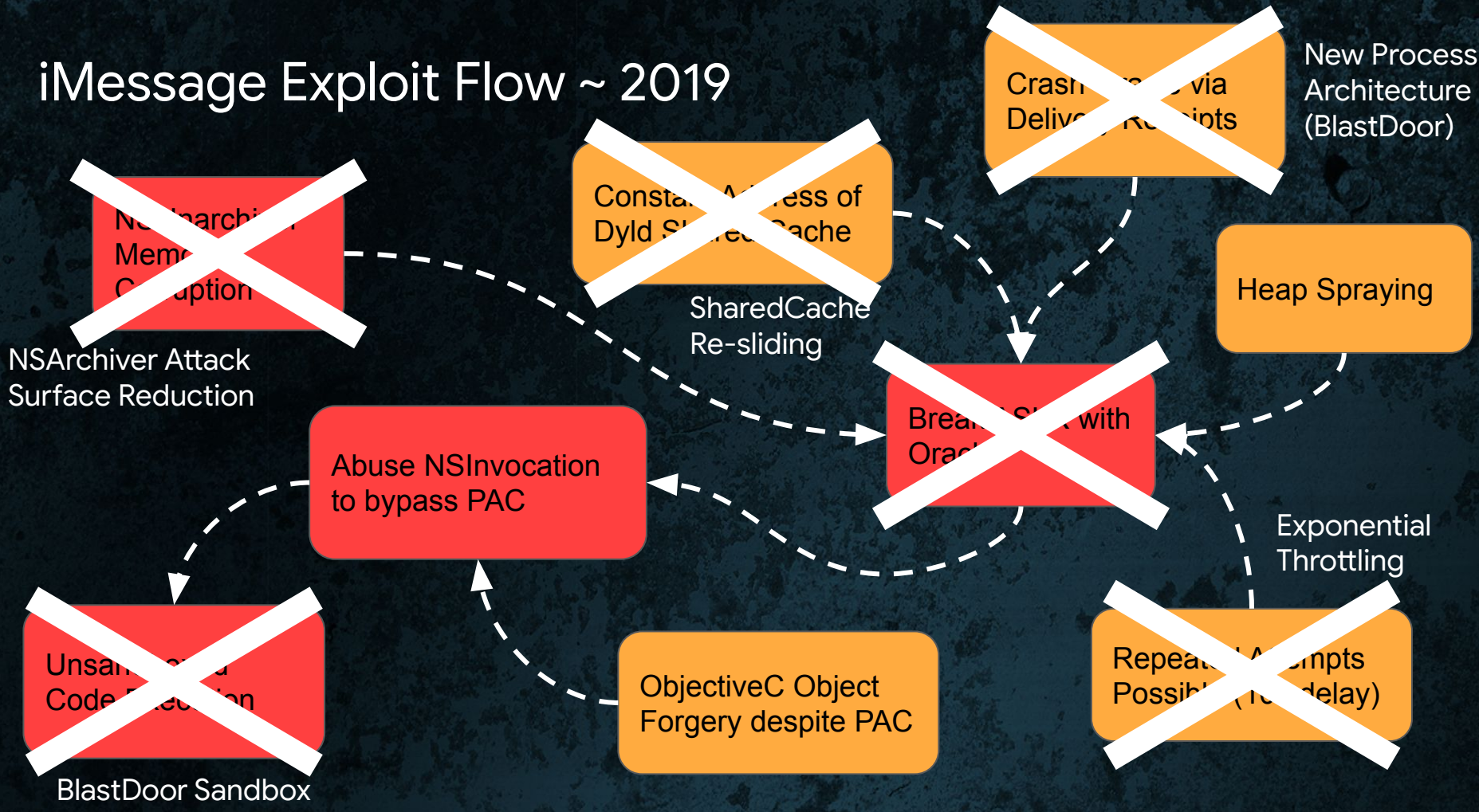
iMessage Exploit Flow ~ 2019



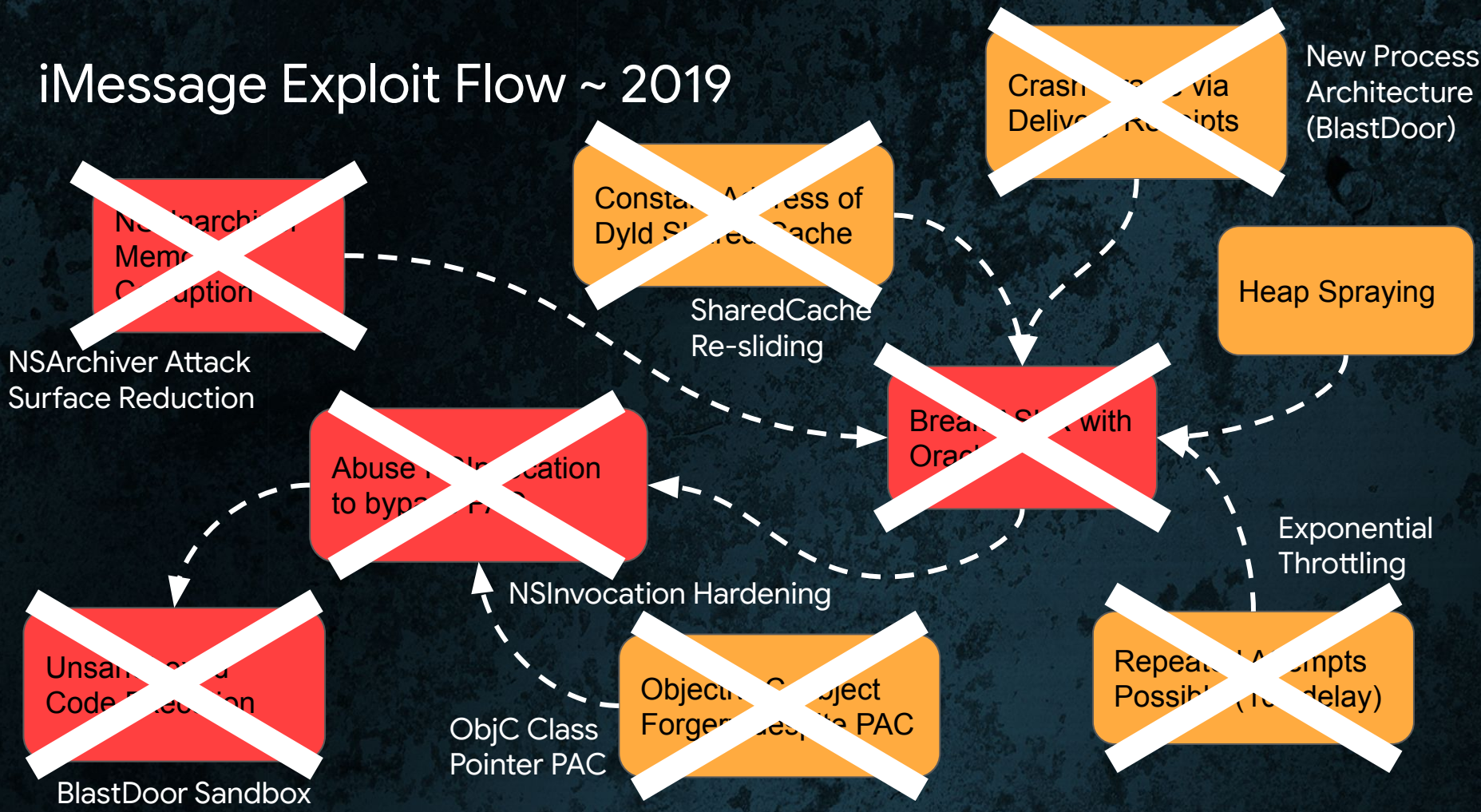
iMessage Exploit Flow ~ 2019



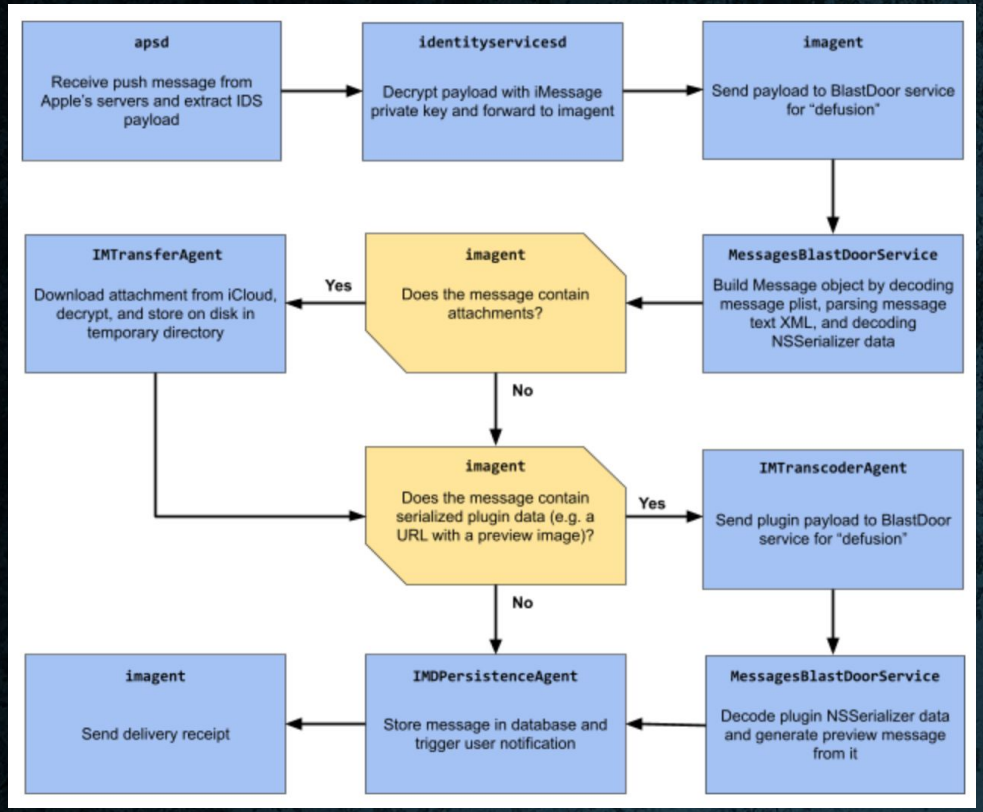
iMessage Exploit Flow ~ 2019



iMessage Exploit Flow ~ 2019



A Fundamentally Different Exploit


apsd
Receive push message from Apple's servers and extract IDS payload
identityservicesd
Decrypt payload with iMessage private key and forward to imagent
imagent
Send payload to BlastDoor service for "defusion"
MessagesBlastDoorService
Build Message object by decoding message plist, parsing message text XML, and decoding NSSerializer data
imagent
Does the message contain attachments?
Yes
IMTransferAgent
Download attachment from iCloud, decrypt, and store on disk in temporary directory
No
imagent
Does the message contain serialized plugin data (e.g. a URL with a preview image)?
Yes
IMTranscoderAgent
Send plugin payload to BlastDoor service for "defusion"
No
MessagesBlastDoorService
Decode plugin NSSerializer data and generate preview message from it
IMDPersistenceAgent
Store message in database and trigger user notification
imagent
Send delivery receipt

one_loop.gif

```
...
00000300  08 10 00 00 10 00 08 18  |........|
00000308  00 08 00 00 00 21 ff 0b  |.....!..|
00000310  4e 45 54 53 43 41 50 45  |NETSCAPE|
00000318  32 2e 30 03 01 01 00 00  |2.0.....|
...
```

infinite_loop.gif

```
...
00000300  08 10 00 00 10 00 08 18  |........|
00000308  00 08 00 00 00 21 ff 0b  |.....!..|
00000310  4e 45 54 53 43 41 50 45  |NETSCAPE|
00000318  32 2e 30 03 01 00 00 00  |2.0.....|
...
```

"Netscape Navigator has an Application Extension Block that tells Navigator to loop the entire GIF file.

The Netscape block MUST APPEAR IMMEDIATELY AFTER THE GLOBAL COLOR TABLE OF THE LOGICAL SCREEN DESCRIPTOR.

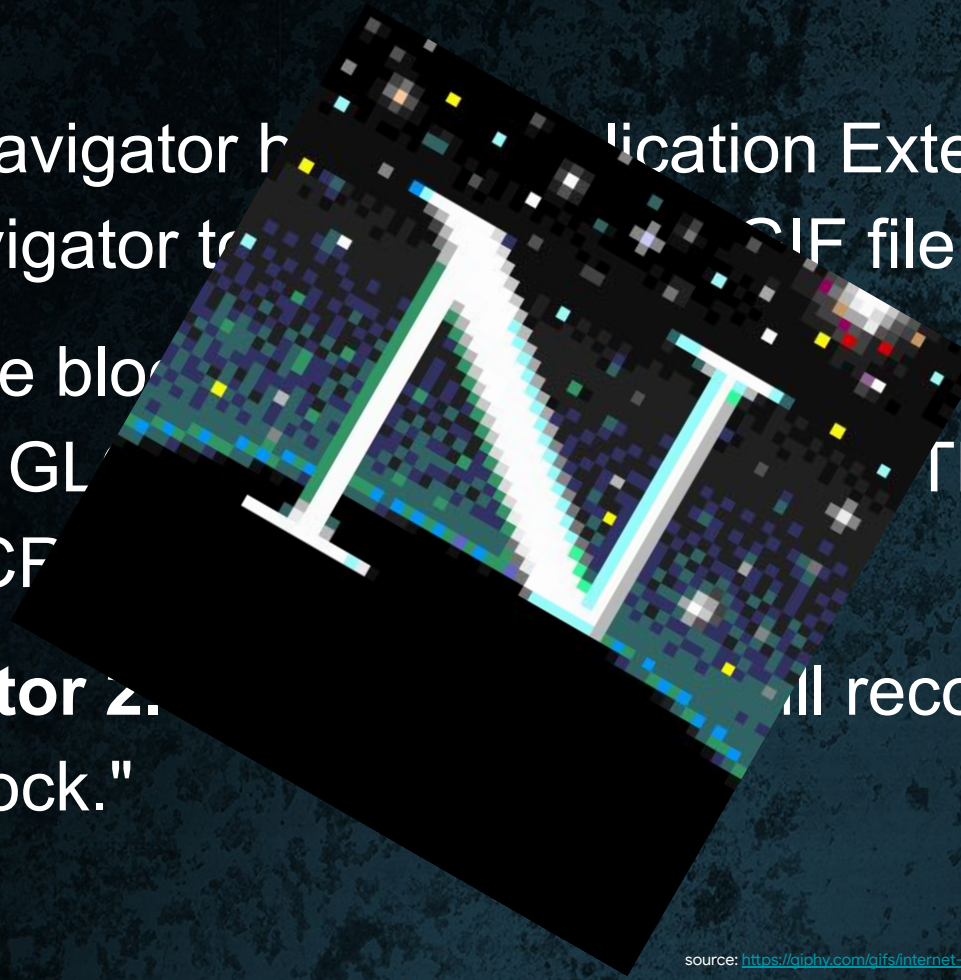
Only **Navigator 2.0 Beta4 or better** will recognize this Extension block."

https://www6.uniovi.es/gifanim/gifabout.htm

"Netscape Navigator h lication Extension Block that tells Navigator t IF file.

The Netscape blo IATELY AFTER THE GL THE LOGICAL SCF

Only **Navigator 2.** ll recognize this Extension block."



source: https://giphy.com/gifs/internet-netscape-anjRJ4nv9WJzO

https://www6.uniovi.es/gifanim/gifabout.htm

Implementation of infinite looping in iMessage:

```
[IMGIFUtils copyGifFromPath:toDestinationPath:error]

objc_msgSend(a1,
          sel_readFileProperties_fromImageSource_withUpdatedLoopCount_error_,
          &v36,
          v16,
          0LL,    // New loop counter to use
          &v35);
```

```
20: IMSharedUtilities  copyGifFromPath:toDestinationPath:error:
19: IMSharedUtilities  readFileProperties:fromImageSource:withUpdatedLoopCount:error:
18: IMSharedUtilities  readFileProperties:fromImageSource:error:
17: ImageIO            _CGImageSourceCopyProperties
16: ImageIO            IIOImageSource::copyProperties
15: ImageIO            IIOImageSource::getProperties
14: ImageIO            IIO_Reader_PDF::updateSourceProperties
13: ImageIO            CreateSessionPDFRef
12: CoreGraphics       _CGPDFDocumentCreateWithProvider
11: CoreGraphics       _pdf_xref_create
10: CoreGraphics       _CGPDFXRefStreamCreate
 9: CoreGraphics       _xref_stream_create
 8: CoreGraphics       _xref_stream_read_section
 7: CoreGraphics       _CGPDFSourceGetc
 6: CoreGraphics       _CGPDFSourceRefill
 5: CoreGraphics       _jbig2_filter_refill
 4: CoreGraphics       read_bytes
 3: CoreGraphics       JBIG2Stream::reset
 2: CoreGraphics       JBIG2Stream::readSegments
 1: CoreGraphics       JBIG2Stream::readTextRegionSeg
 0: CoreGraphics       JBIG2Stream::readTextRegionSeg
```

iMessage

ImageIO

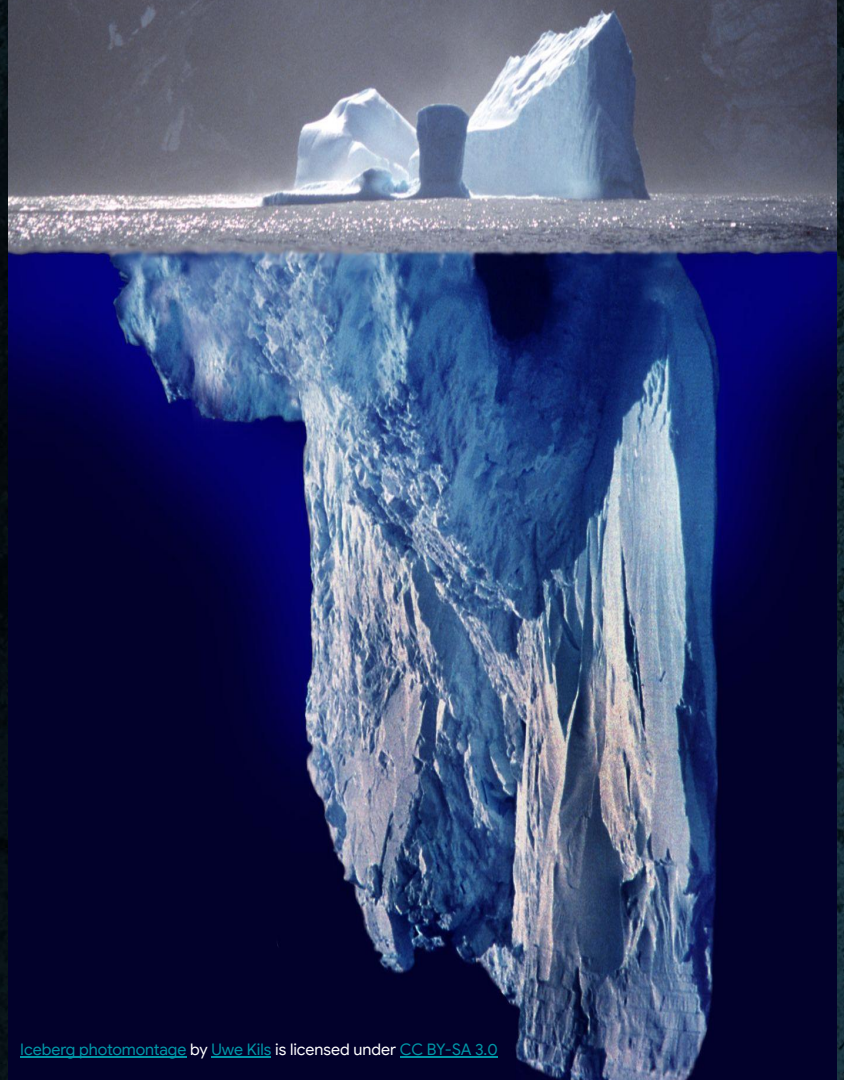
CoreGraphics

XPdf

alter loop-count property of an animated GIF

process arbitrary JBIG2

```
20: IMSharedUtilities
19: IMSharedUtilities
18: IMSharedUtilities
17: ImageIO
16: ImageIO
15: ImageIO
14: ImageIO
13: ImageIO
12: CoreGraphics
11: CoreGraphics
10: CoreGraphics
 9: CoreGraphics
 8: CoreGraphics
 7: CoreGraphics
 6: CoreGraphics
 5: CoreGraphics
 4: CoreGraphics
 3: CoreGraphics
 2: CoreGraphics
 1: CoreGraphics
 0: CoreGraphics
```



iMessage

ImageIO

CoreGraphics

XPdf

alter loop-count property of an animated GIF

process arbitrary JBIG2

.ai

.bc

.cur

.exr

.psd

.pvr

.bmp

.mpo

.astc

.tiff

.gif

.pbm

.raw

.heif

.ktx

.icns

.atx

.rad

.png

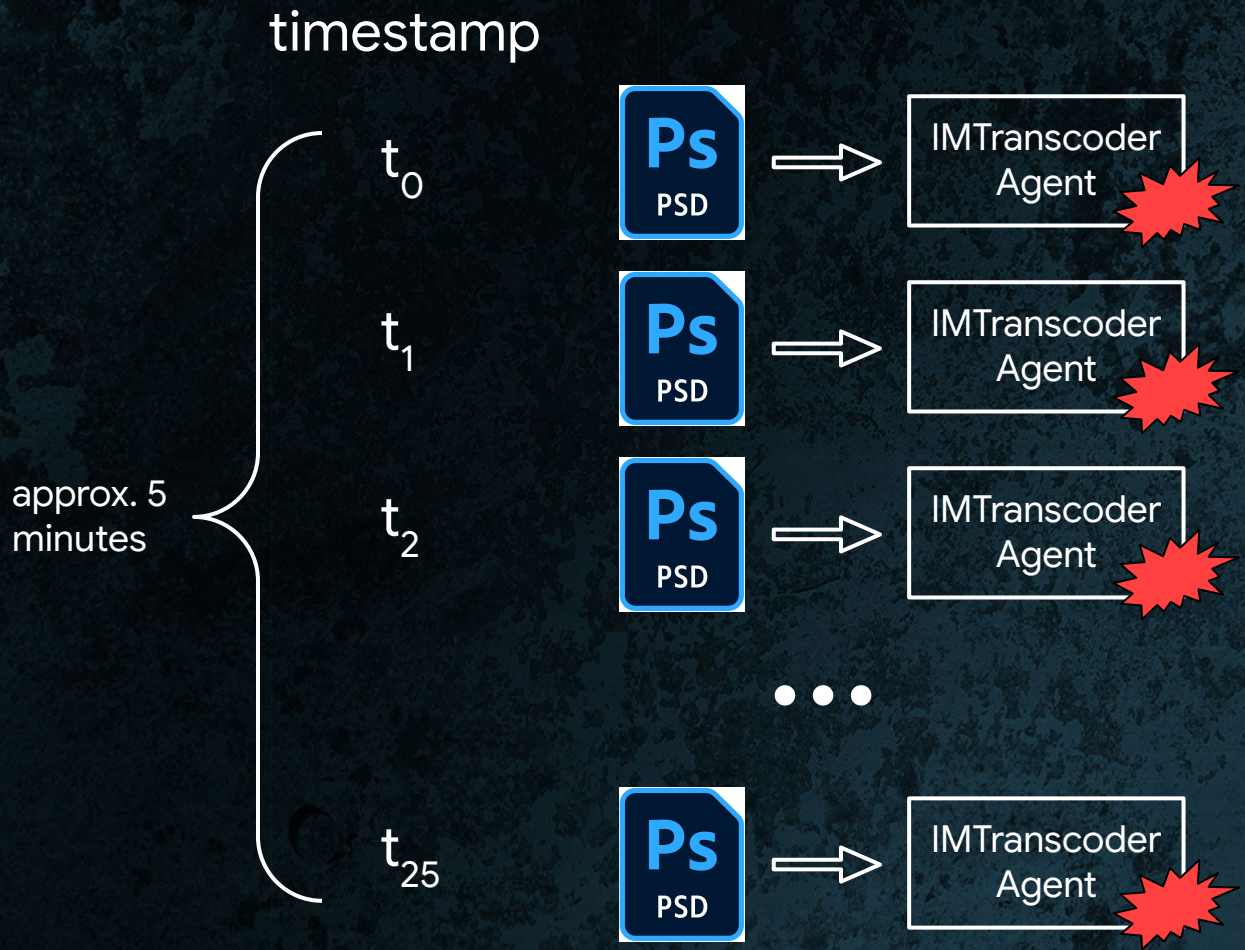
.ico

.jpeg

.jp2

.tga

.pdf



Hypothesis: another ASLR crash oracle?

```m
// clang -o render render.m -framework Foundation -framework CoreGraphics -framework AppKit

#include <Foundation/Foundation.h>
#import <ImageIO/ImageIO.h>
#import <AppKit/AppKit.h>
#import <CoreGraphics/CoreGraphics.h>

int main(int argc, const char* argv[]) {
    NSString* file = [NSString stringWithUTF8String:argv[1]];
    NSData* content = [NSData dataWithContentsOfFile:file];
    NSImage* img = [[NSImage alloc] initWithData:content];
    CGImageRef cgImg = [img CGImageForProposedRect:nil context:nil hints:nil];
    size_t width = CGImageGetWidth(cgImg);
    size_t height = CGImageGetHeight(cgImg);
    CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
    CGContextRef ctx = CGBitmapContextCreate(0, width, height, 8, 0, colorspace, 1);
    CGRect rect = CGRectMake(0, 0, width, height);
    CGContextDrawImage(ctx, rect, cgImg);
    return 0;
}
```

MacOS for analysis vs iOS for exploitation

MacOS: large allocation (gigabytes) succeeds

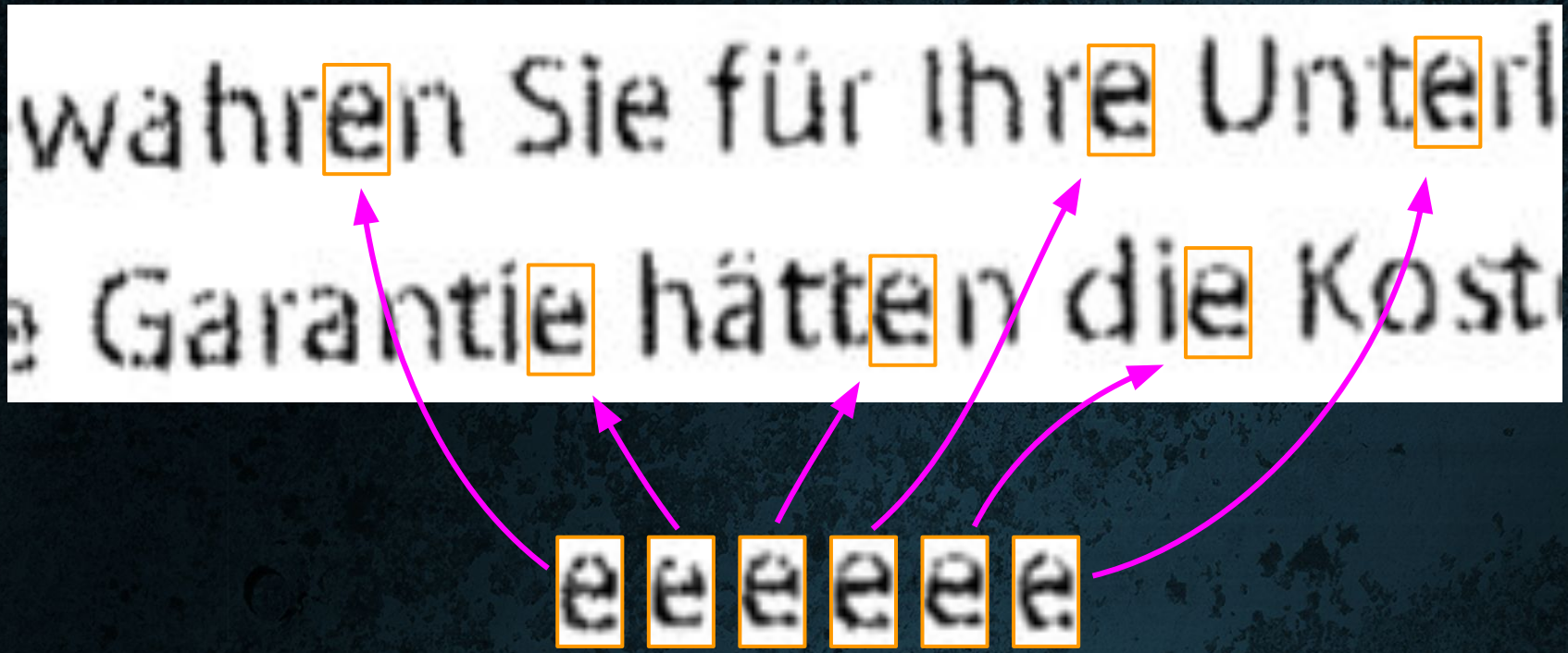
iOS: large allocation fails -> NULL pointer dereference

Hypothesis: maybe an offset from NULL read/write?

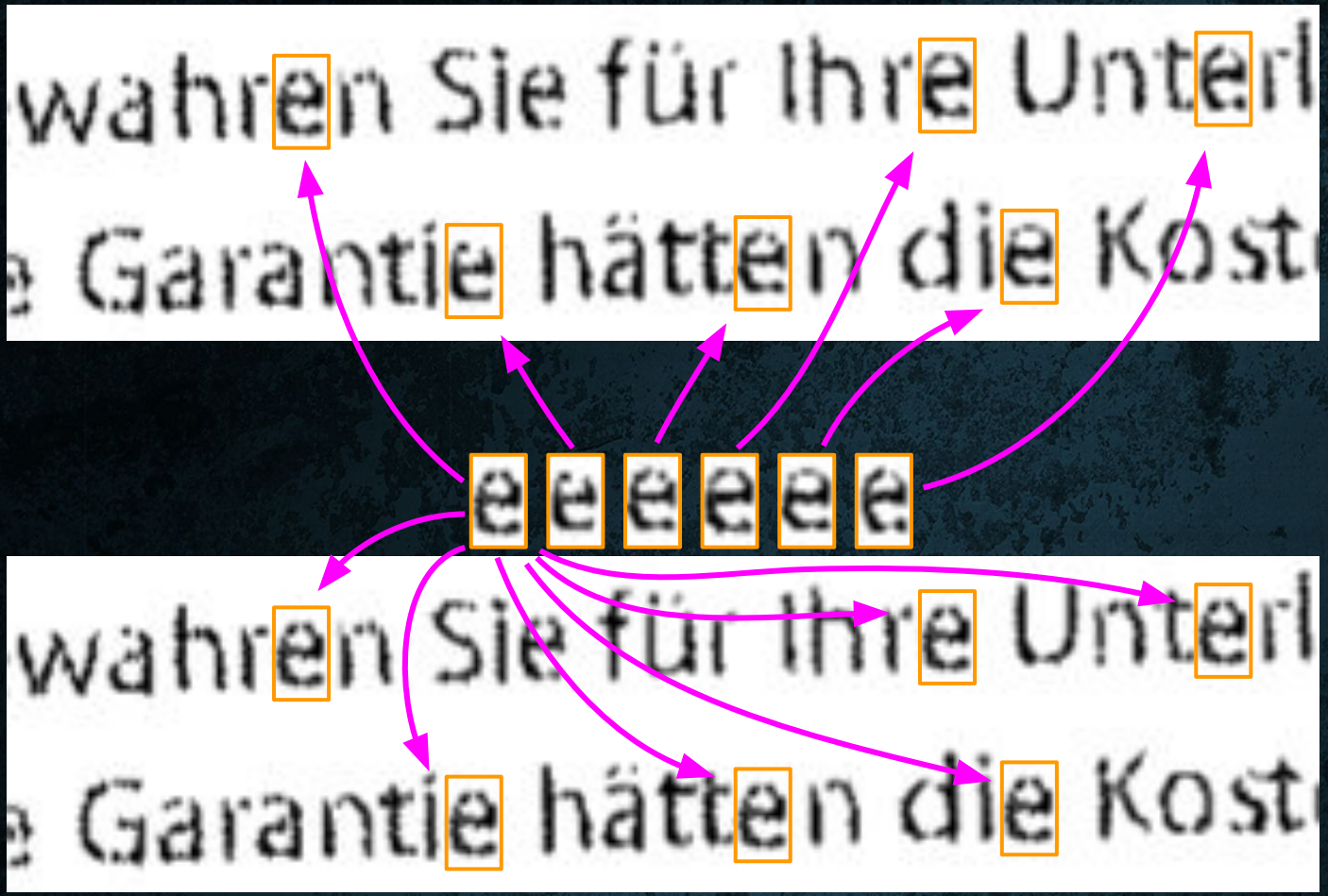
Reversing the PSD parser: it's just a crasher, no way to avoid near-NULL access

```
default 13:37:00.000000 +0000   ReportCrash Saved type 'XXX'
  report (2 of max 25) at
  /var/mobile/Library/Logs/CrashReporter/ZZZ-2022-02-02-133700.ips
```

JBIG2 compression



JBIG2 compression

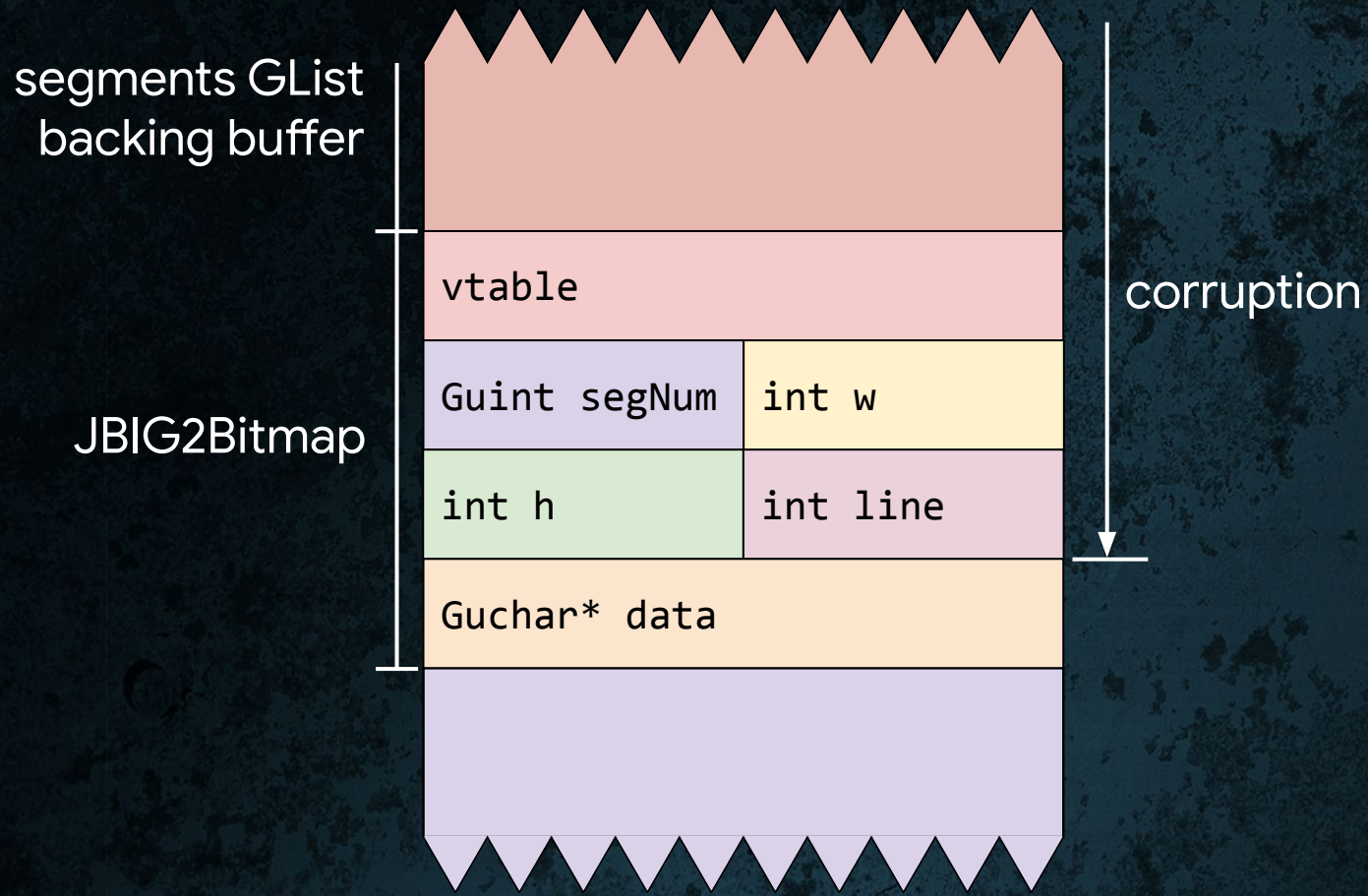


Unbounding JBIG2 canvas with a heap overflow:

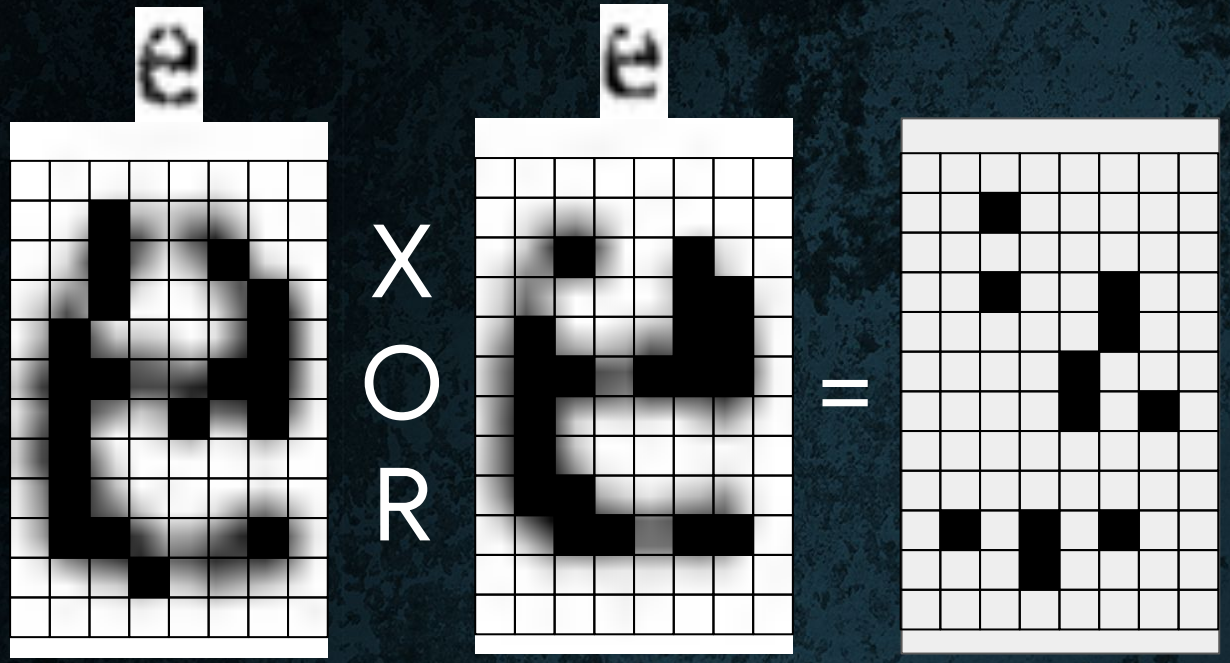
```
Guint numSyms;

numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
 if ((seg = findSegment(refSegs[i]))) {
   if (seg->getType() == jbig2SegSymbolDict) {
     numSyms += ((JBIG2SymbolDict *)seg)->getSize();
   }
   // ...
}
// ...
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));
```

Unbounding JBIG2 canvas with a heap overflow:

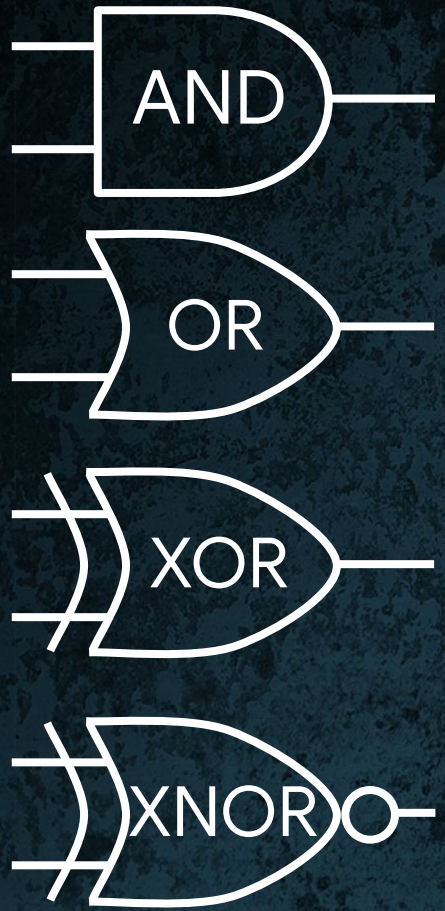


JBIG2 refinement operations



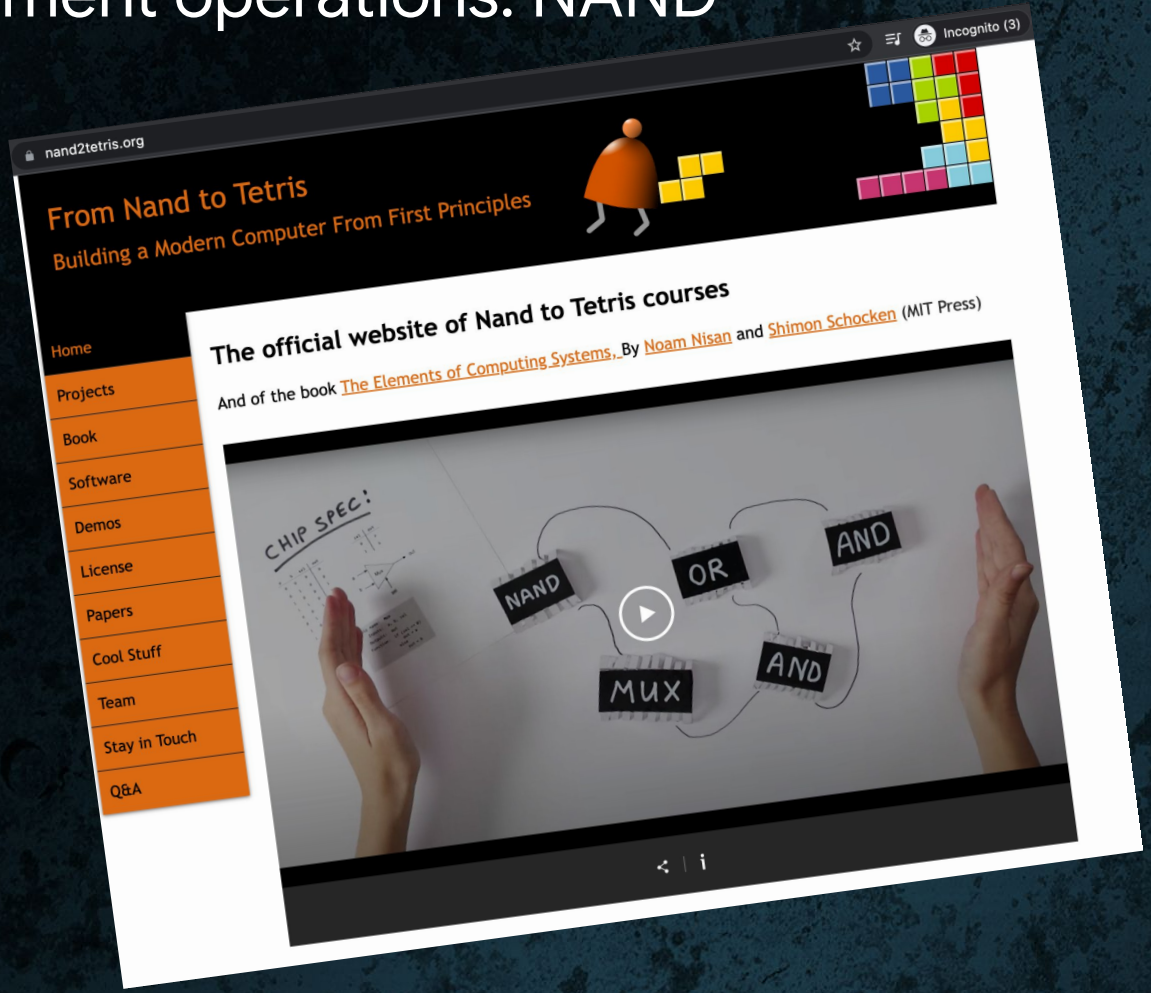
substituted original difference

JBIG2 refinement operations: logic gates

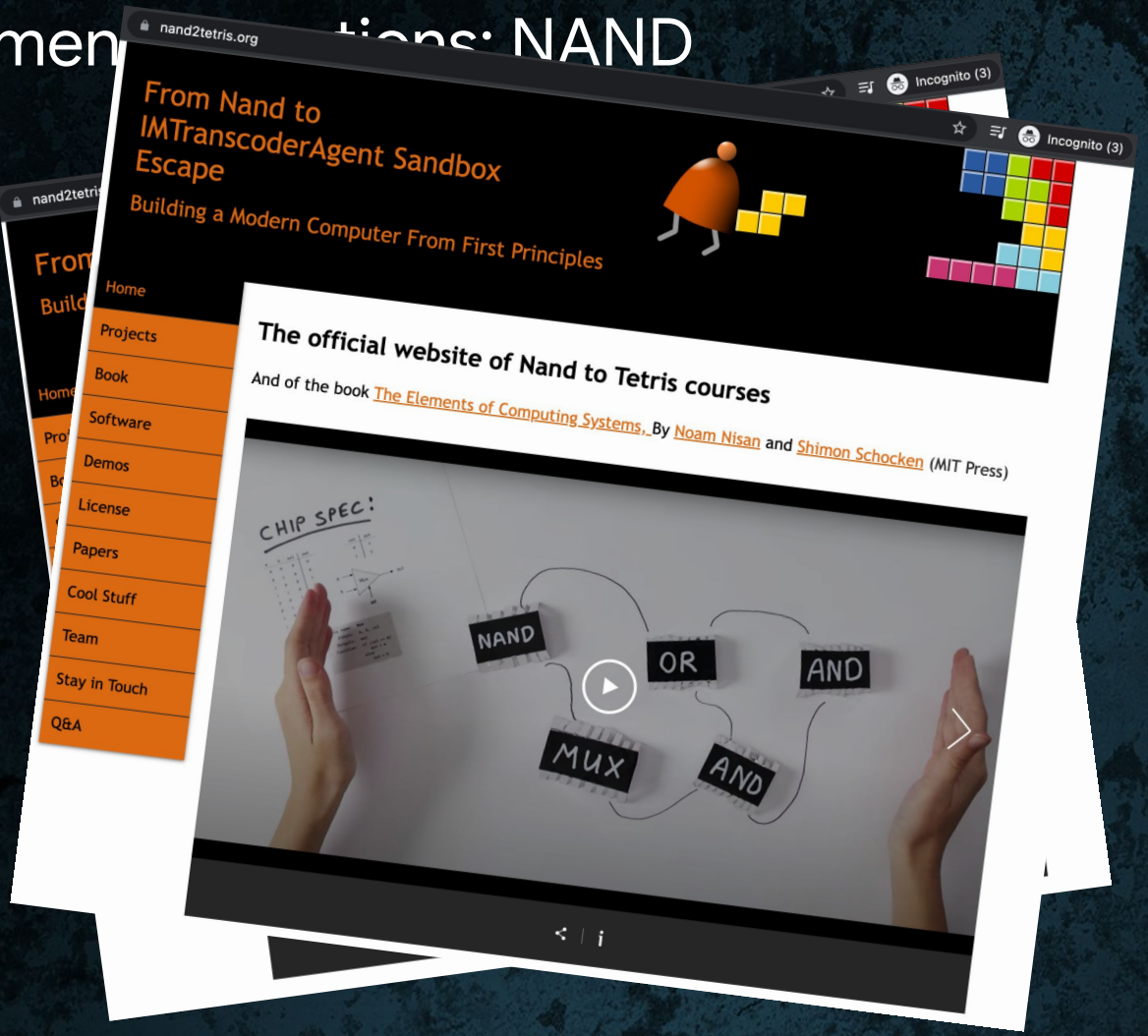


JBIG2 refinement operations: NAND

JBIG2 refinement operations: NAND



JBIG2 refinemen tions: NAND



Conclusion

- The right mitigations/hardenings can make a big difference
  - Shift something something
- Still: assume memory corruption bugs to be exploitable unless proven otherwise (this is hard)
- TODO