

BLUEHAT

IL 2022

A Brief History of iMessage Exploitation

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



iMessage Exploitation ~ 2019

iMessage Exploit Flow ~ 2019

Attack Surface?

Attack Surface: Deserialization

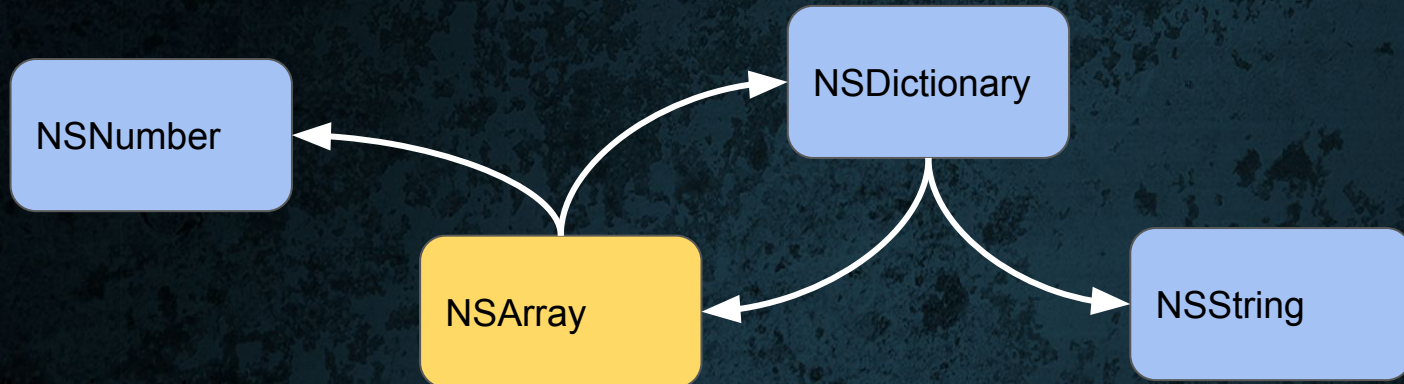
```
{
    ati = [ NSCoder Archiver ];
    gid = "27EDB72A-DFC1-43DD-B8AE-8DBD2CE70068";
    gv = 8;
    p = (
        "mailto:sender@foo.bar",
        "mailto:receiver@foo.bar"
    );
    pv = 0;
    r = "E417E766-0B85-4427-AF49-9246AA76C803";
    t = "Hello BlueHat!";
    v = 1;
    x = "<html><body>Hello BlueHat!</body></html>";
}
```

Attack Surface: Deserialization

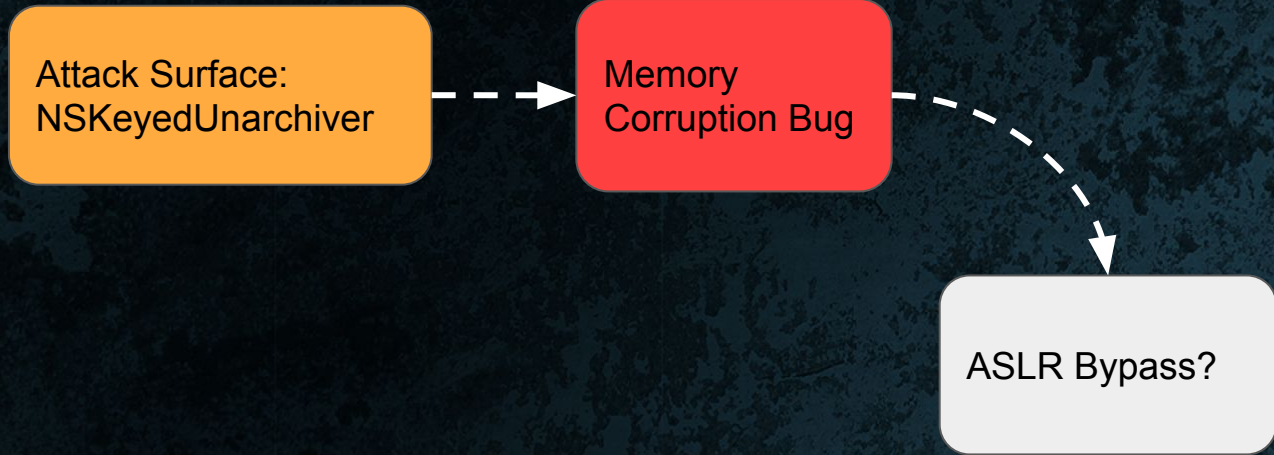
```
{
    ati = [ NSKeyedArchiver Archive ];
    gid = "27EDB72A-DFC1-43DD-B8AE-8DBD2CE70068";
    gv = 8;
    p = (
        "mailto:sender@foo.bar",
        "mailto:receiver@foo.bar"
    );
    pv = 0;
    r = "E417E766-0B85-4427-AF49-9246AA76C803";
    t = "Hello BlueHat!";
    v = 1;
    x = "<html><body>Hello BlueHat!</body></html>";
}
```

Attack Surface: Deserialization

- “NSKeyedArchiver [...] provides a way to encode objects (and scalar values) into an architecture-independent format suitable for storage in a file.”
- Can (de)serialize pretty complex object hierarchies (even circles!)
- This is our attack surface!
- One key is deserialized in Springboard process, which is *unsandboxed*

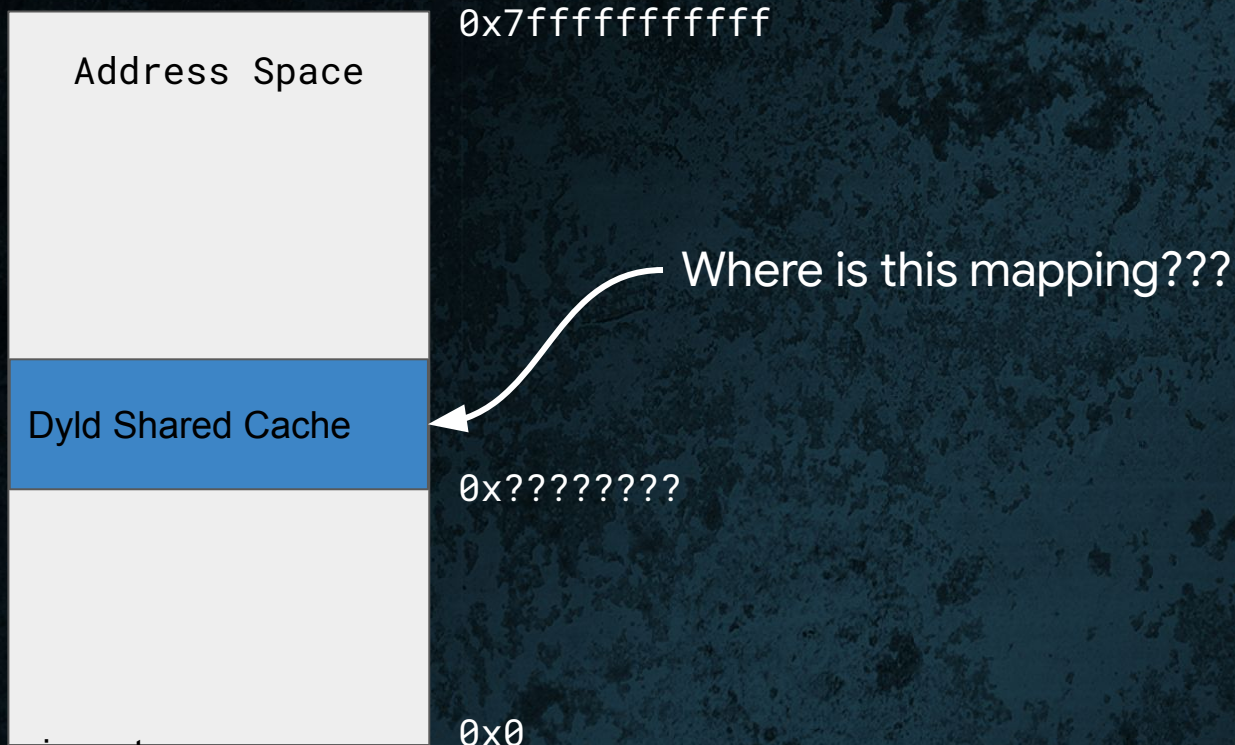


iMessage Exploit Flow ~ 2019



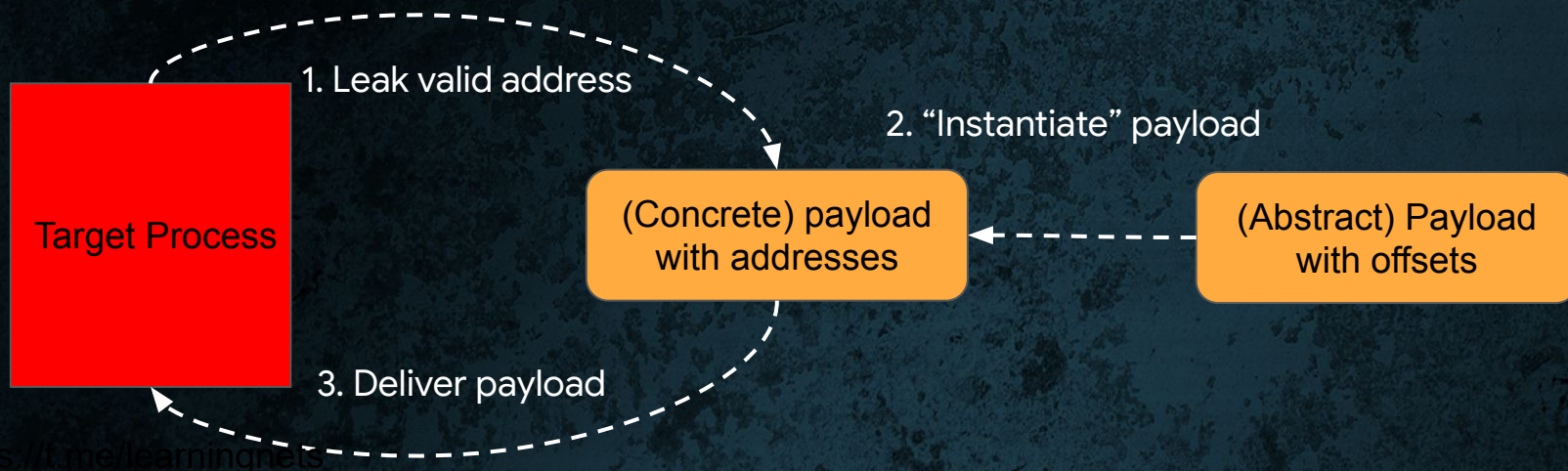
Unsandboxed
Code Execution

Exploitation (~ 2019): Defeating ASLR

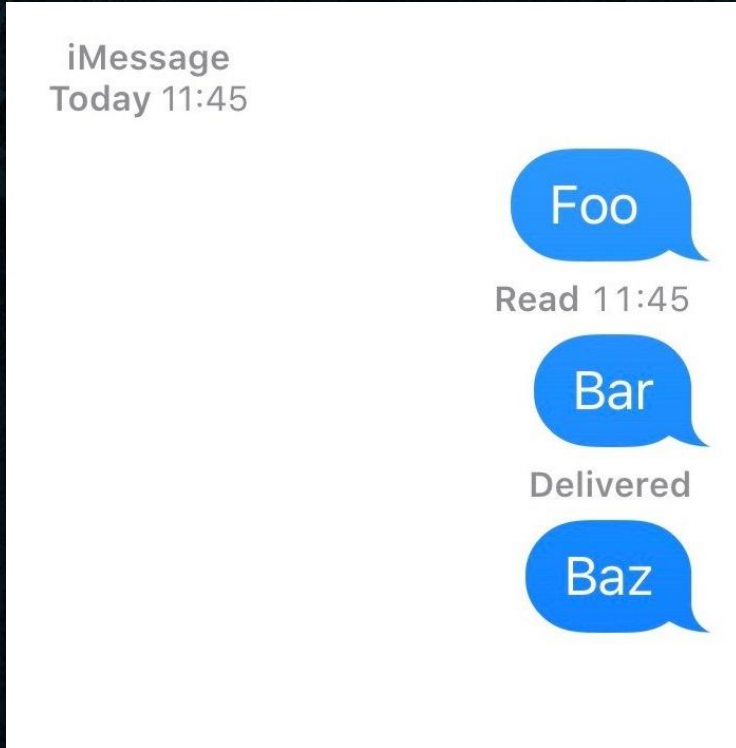


Why is ASLR a Problem?

- **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...

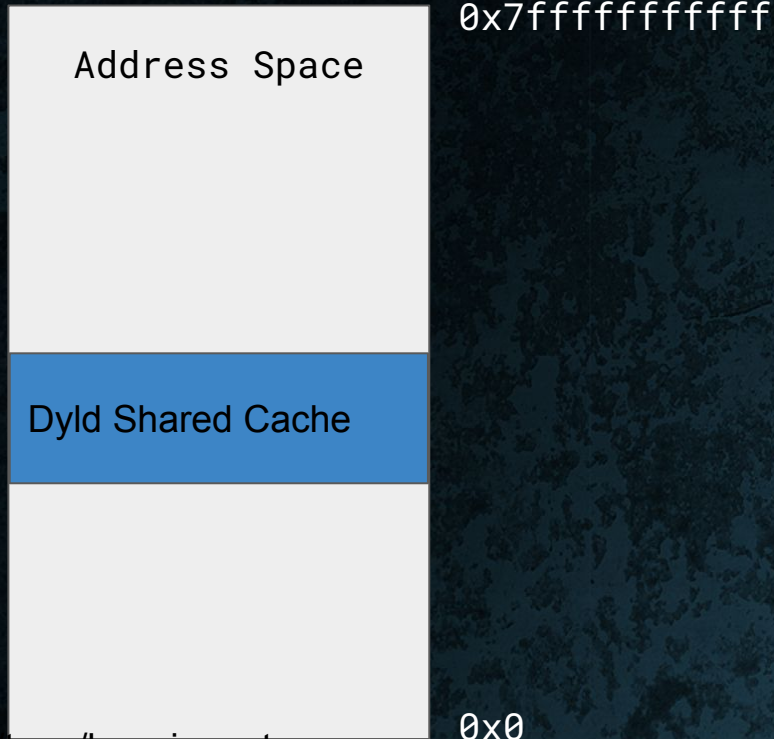


Delivery Receipts as Communication Channel



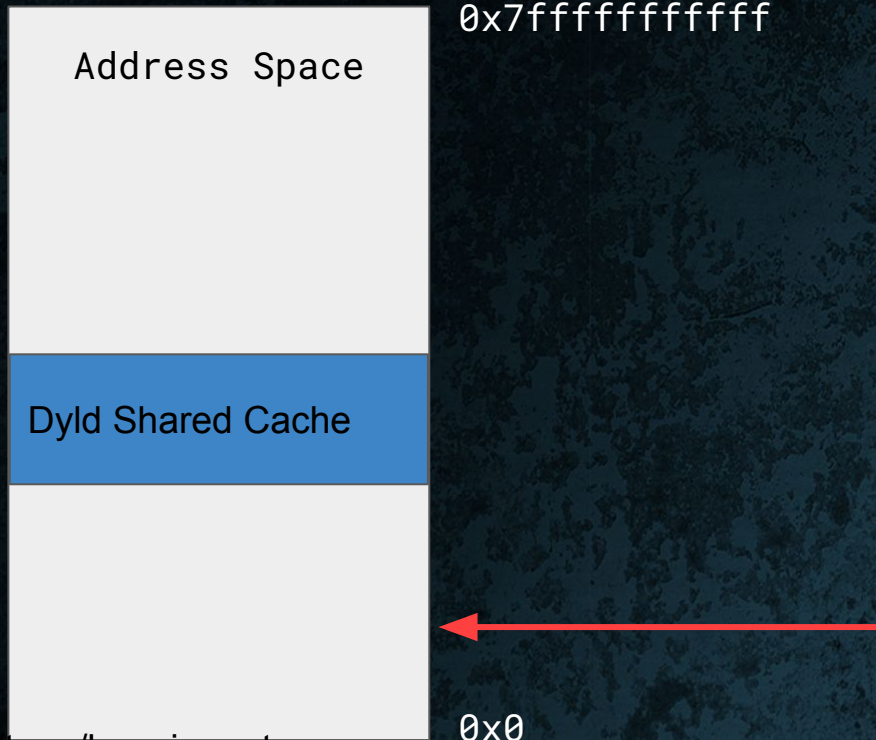
- When iMessage process receives a message, it sends a *delivery receipt* to the sender
- If process crashes before sending the receipt, the delivery receipt message is never sent
- => **1-bit communication channel:**
crashed or didn't crash

Crash Oracle + Binary Search = ASLR defeat



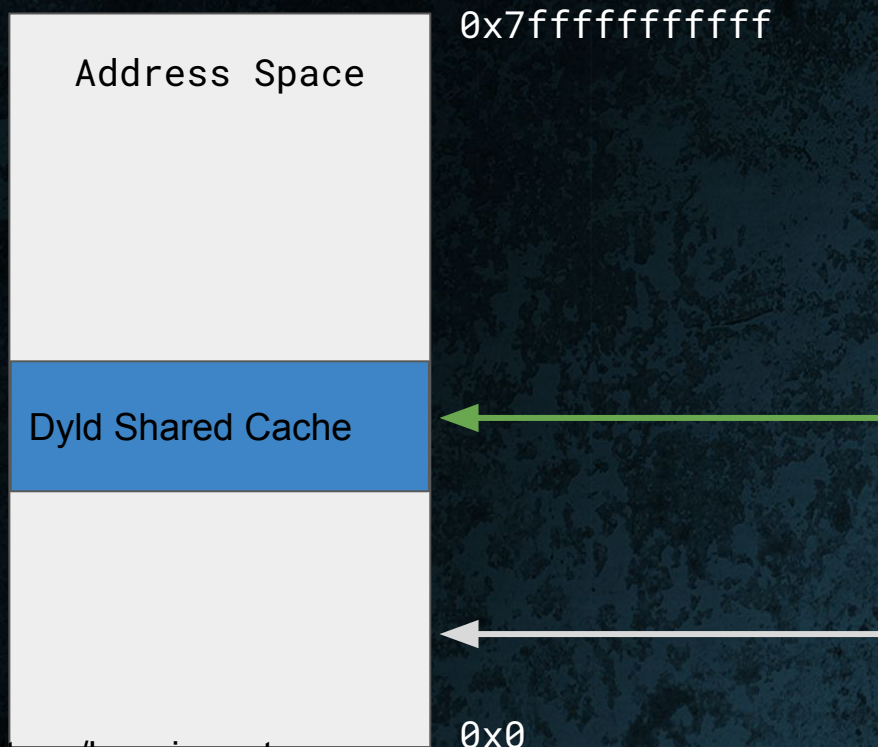
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



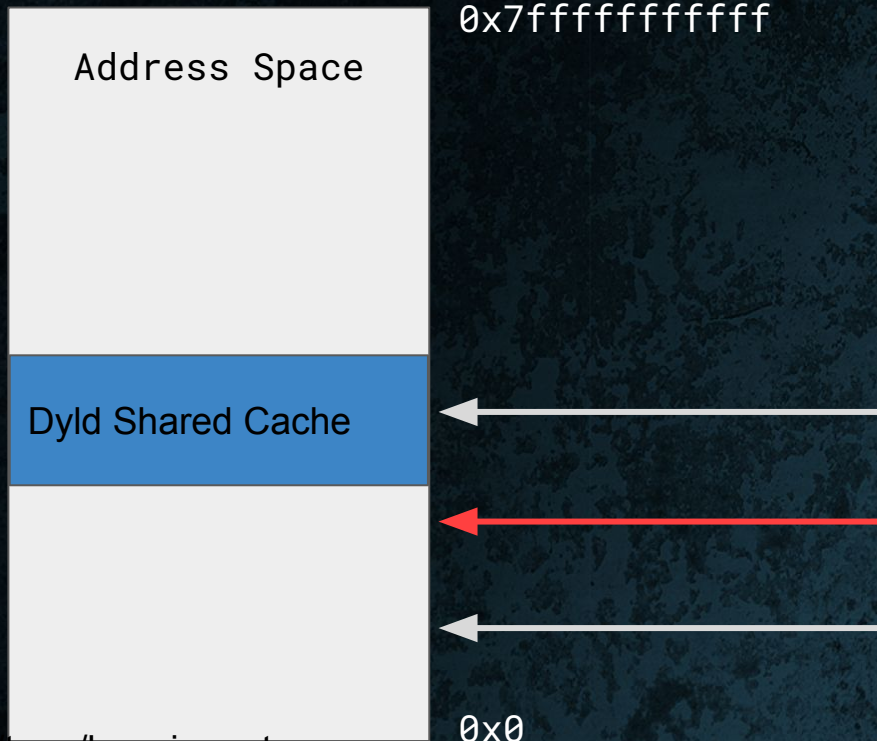
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



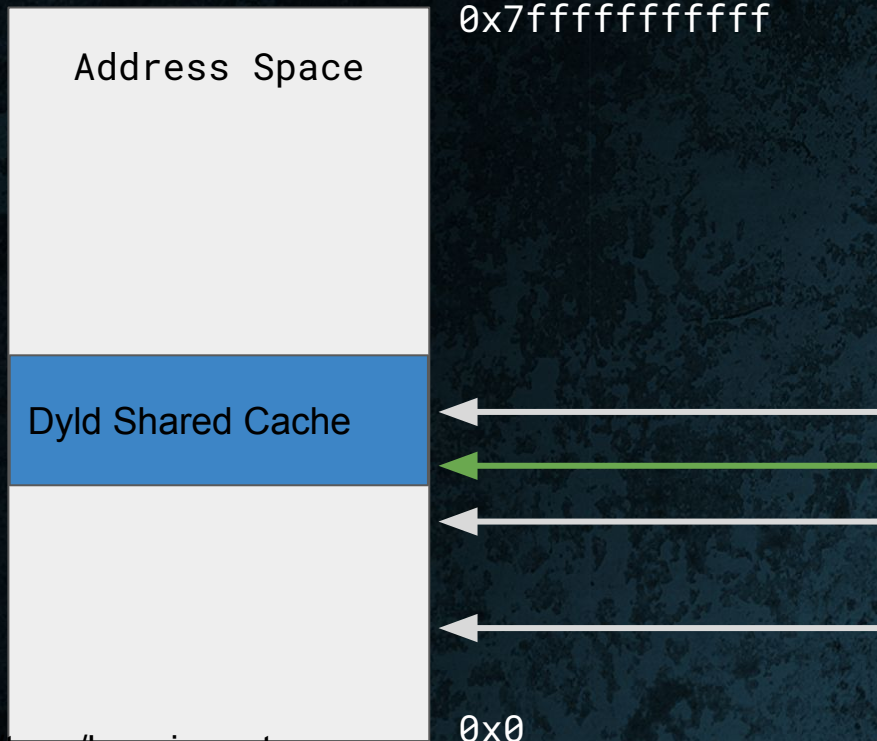
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



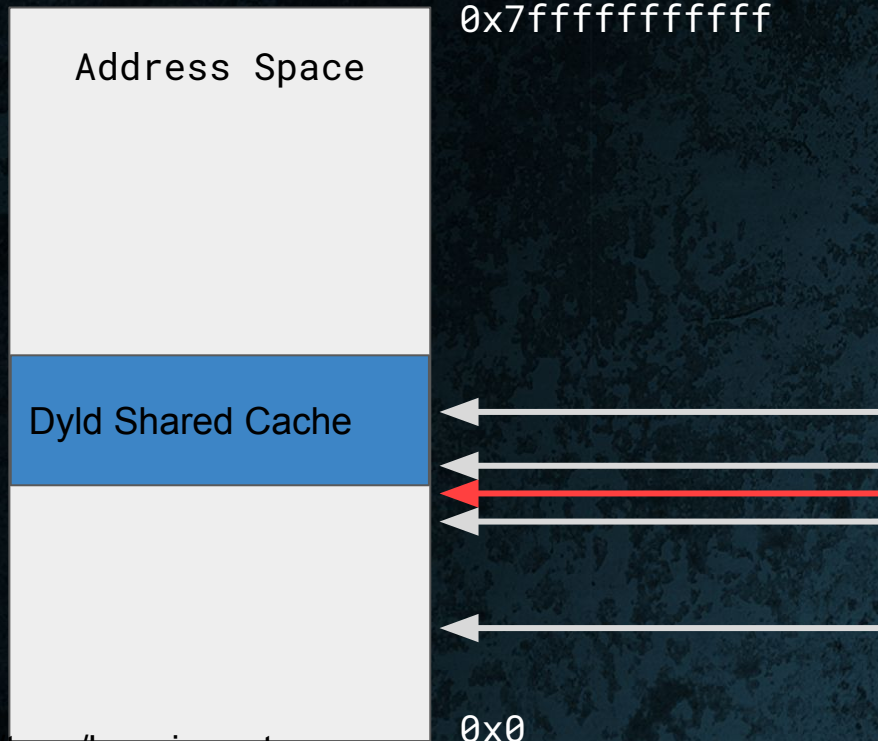
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



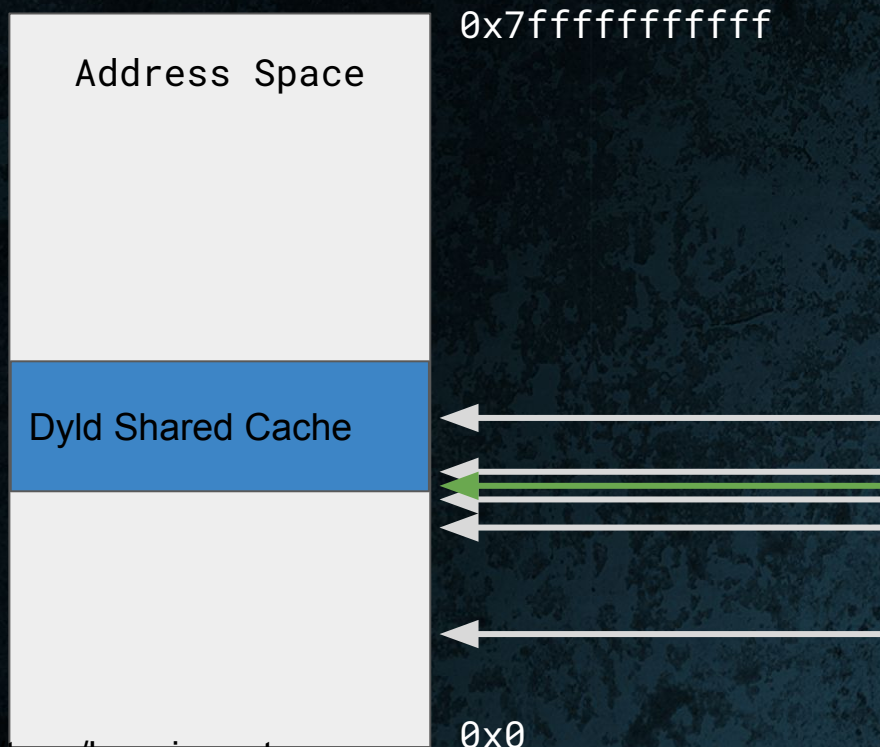
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

Crash Oracle + Binary Search = ASLR defeat



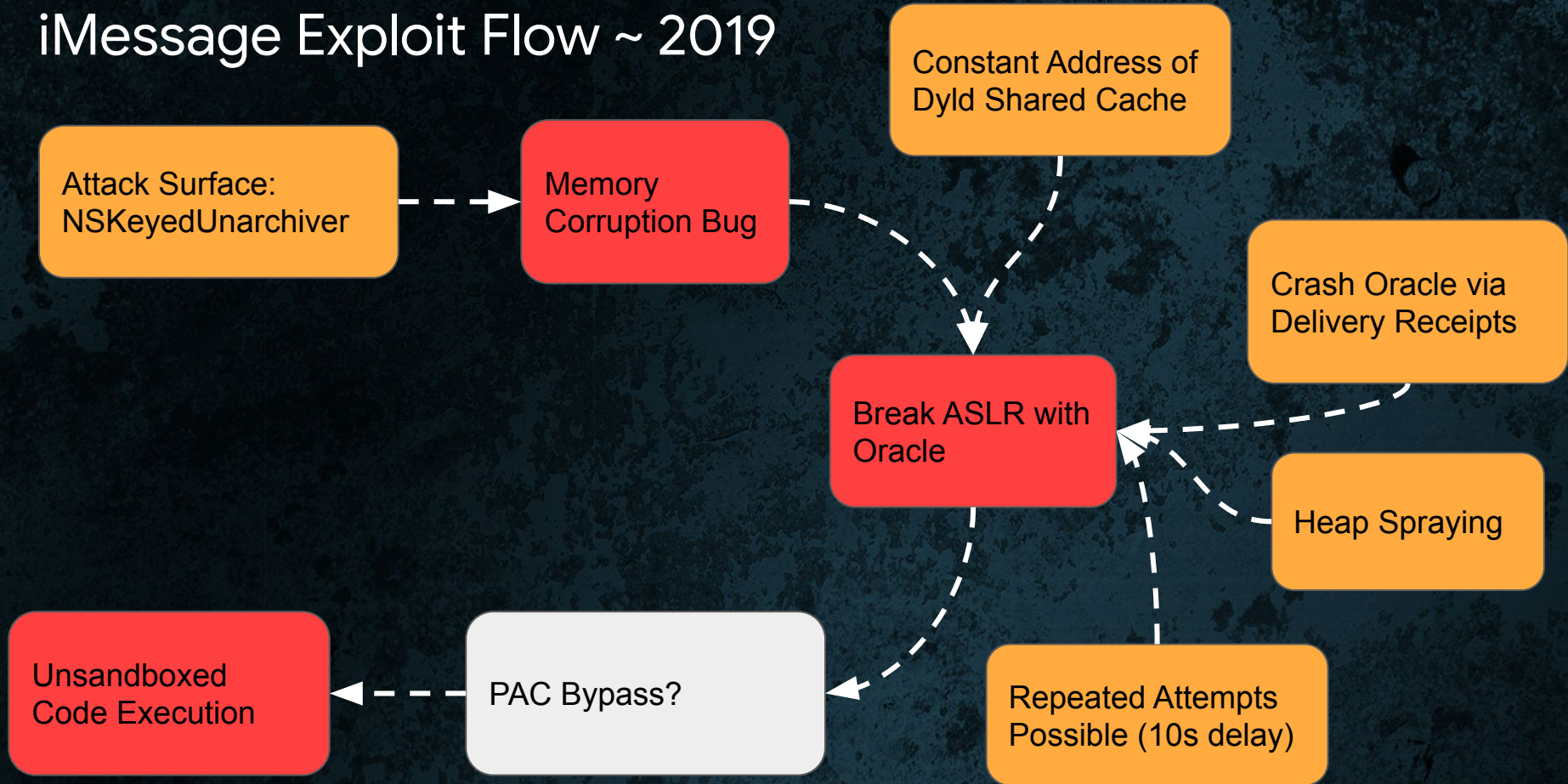
- Construct payload to dereference a given address
- Send payload over iMessage
- Got a delivery receipt? If yes: address is valid, otherwise not
- Do this as binary search to find base address with 20-30 messages

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 code pointers => breaks ROP, JOP, ...

0000002012345678


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

a827152012345678

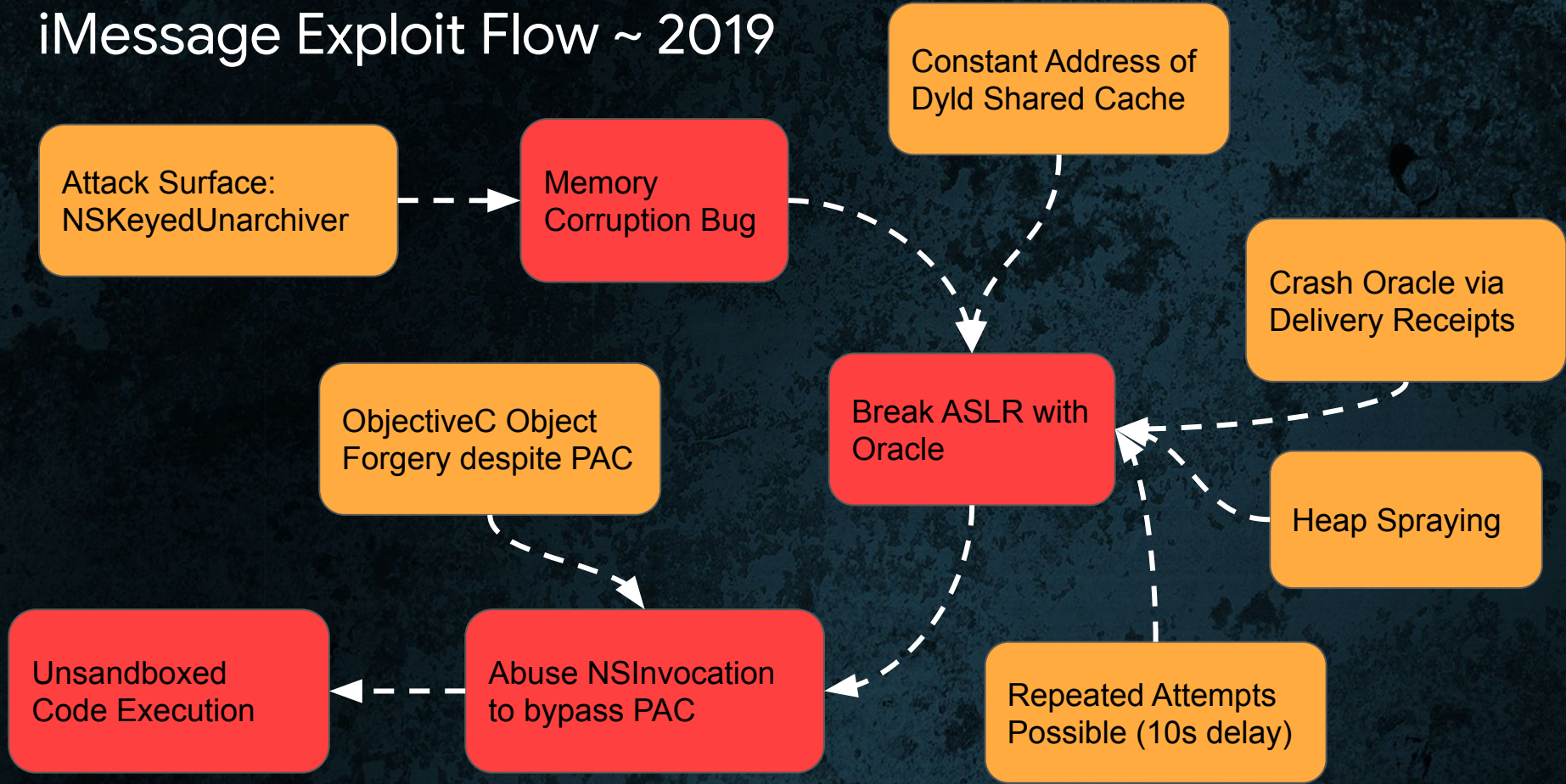

; Authenticate function pointer in X3
; and call it. Clobbers X3 if signature
; is invalid, leading to crash
AUTIZA X3
BL X3

Defeating PAC (Pointer Authentication)

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

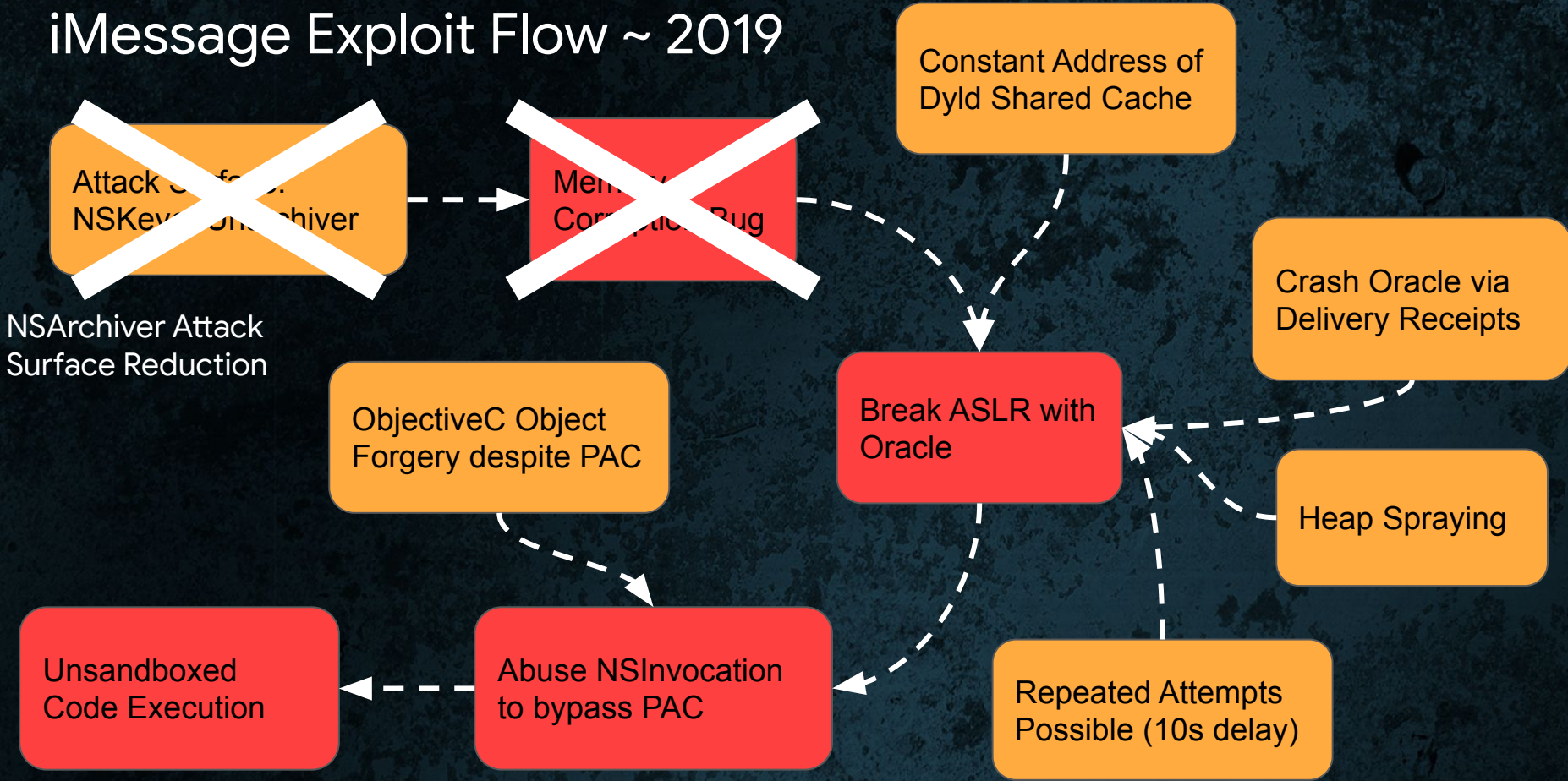
```
NSInvocation* invocation = [NSInvocation invocationWithMethodSignature: sig];  
[invocation setTarget: foo];  
[invocation setSelector:@selector(bar)];  
[invocation invoke];  
// [Foo bar] called
```

iMessage Exploit Flow ~ 2019



iMessage Hardening ~ 2019-2020

iMessage Exploit Flow ~ 2019



NSArchiver Attack
Surface Reduction

ObjectiveC Object
Forgery despite PAC

Unsandboxed
Code Execution

Abuse NSInvocation
to bypass PAC

Break ASLR with
Oracle

Constant Address of
Dyld Shared Cache

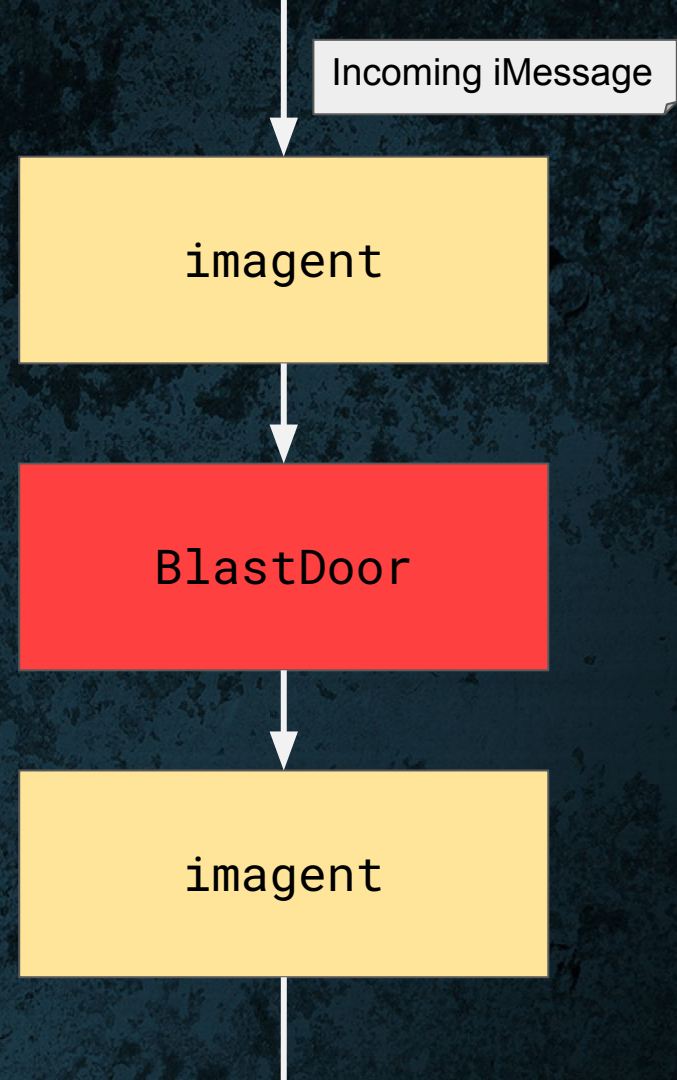
Crash Oracle via
Delivery Receipts

Heap Spraying

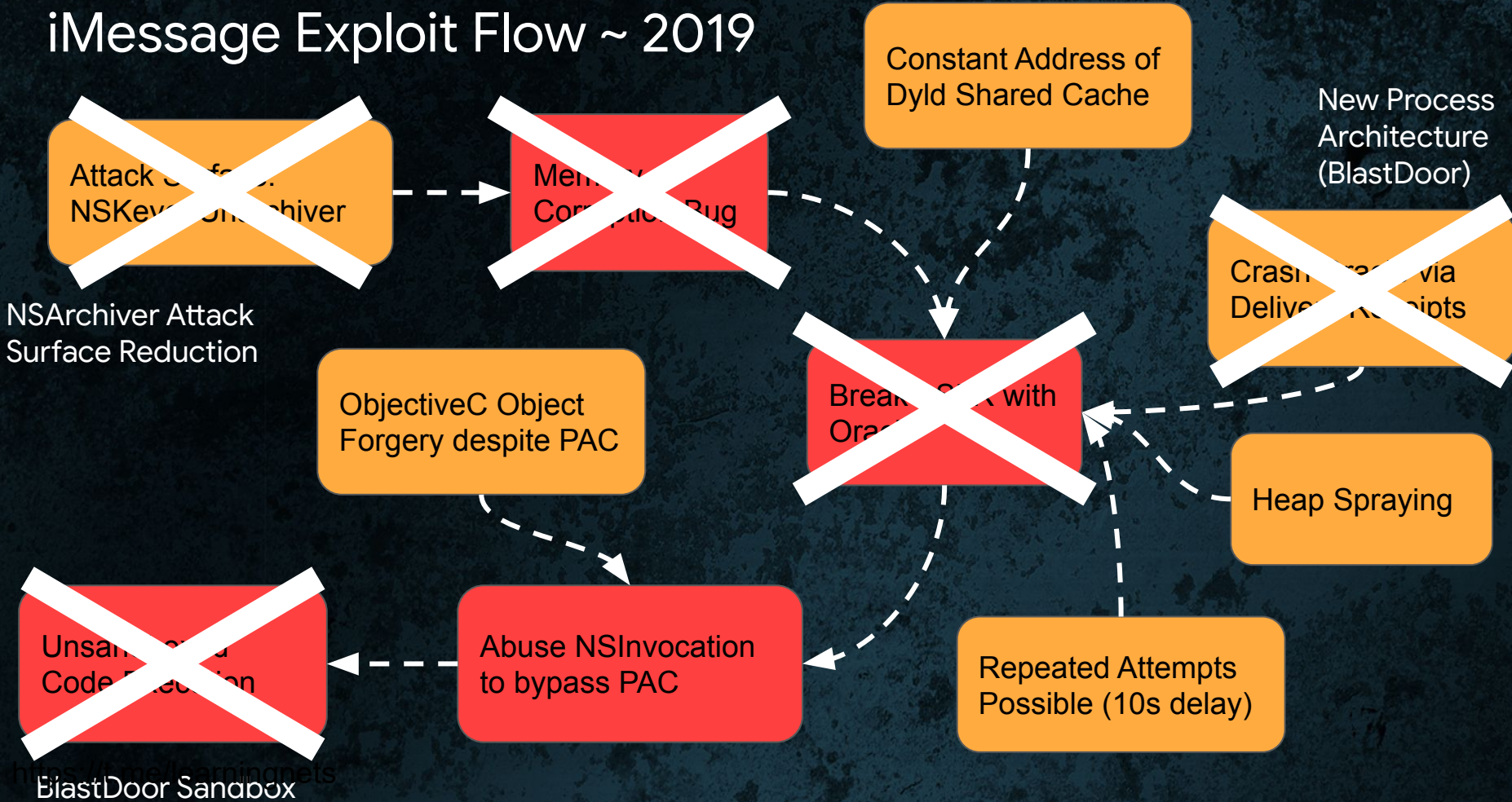
Repeated Attempts
Possible (10s delay)

Blastdoor (iOS 14, ~ mid 2020)

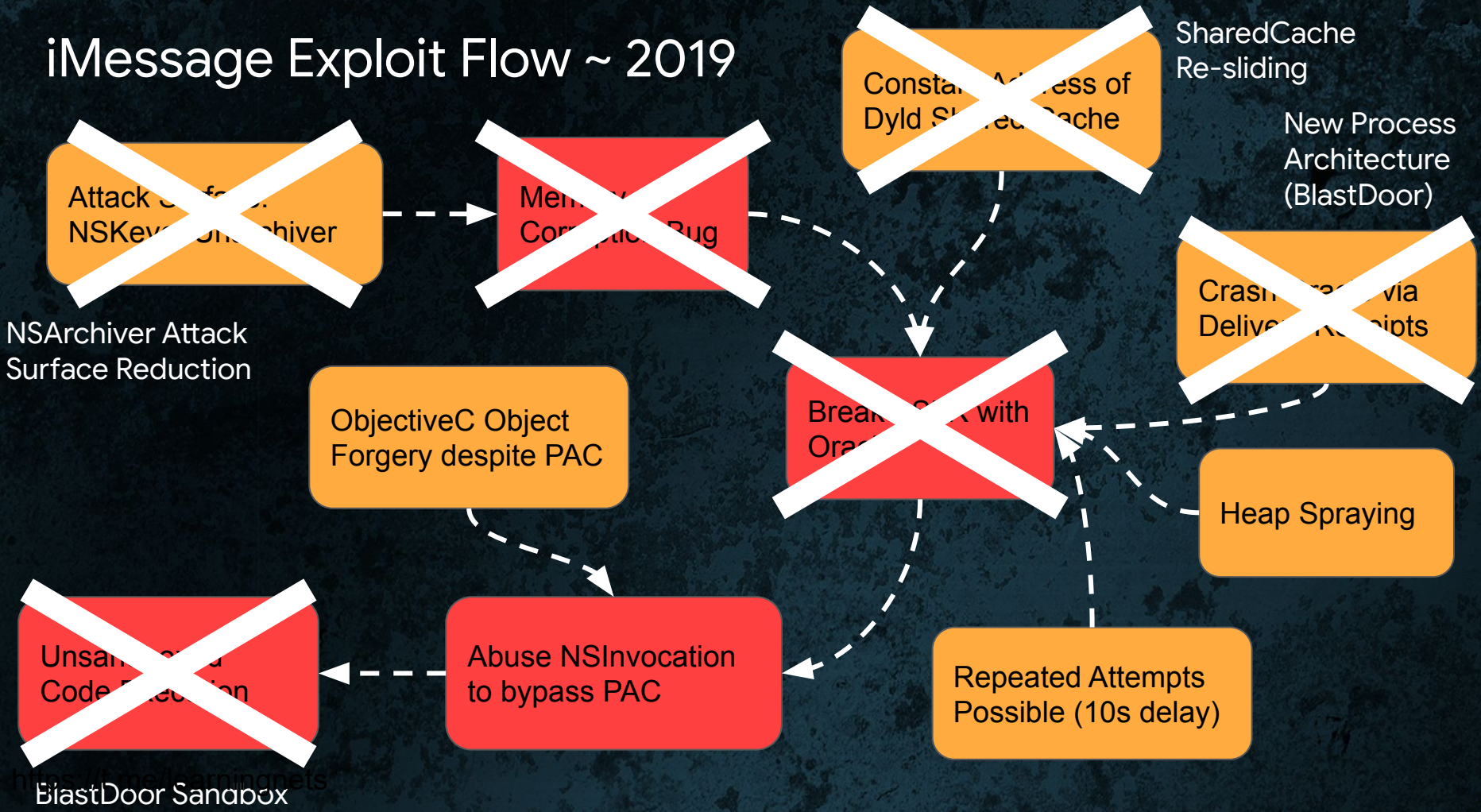
- Re-architected iMessage processing
- Idea: complex parsing now happens in a tightly sandboxed process: MessagesBlastDoorService
- High-level logic implemented in Swift
- Also breaks crash oracle: crashing process (BlastDoor) is not the process sending the delivery receipt (imagent)



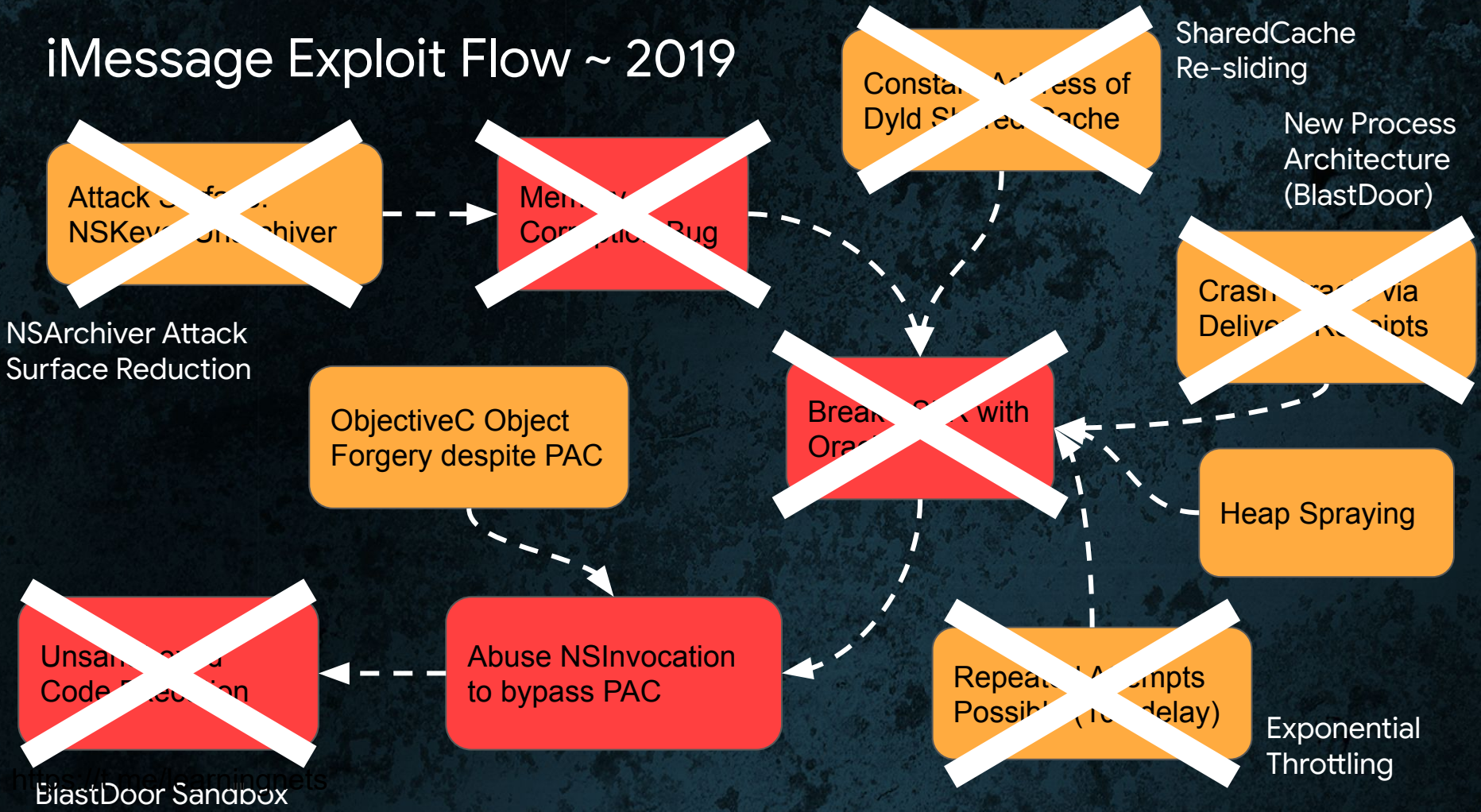
iMessage Exploit Flow ~ 2019



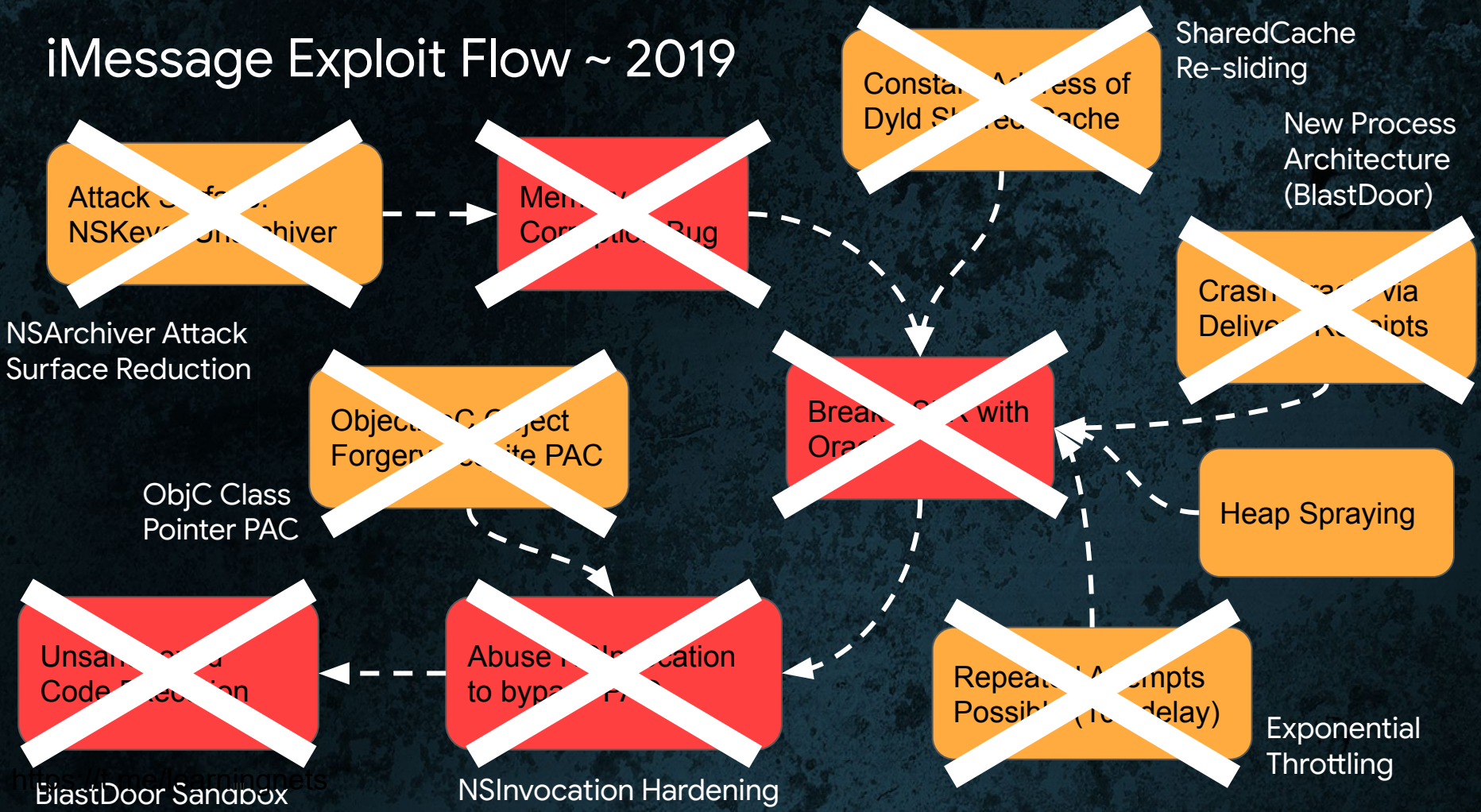
iMessage Exploit Flow ~ 2019



iMessage Exploit Flow ~ 2019



iMessage Exploit Flow ~ 2019



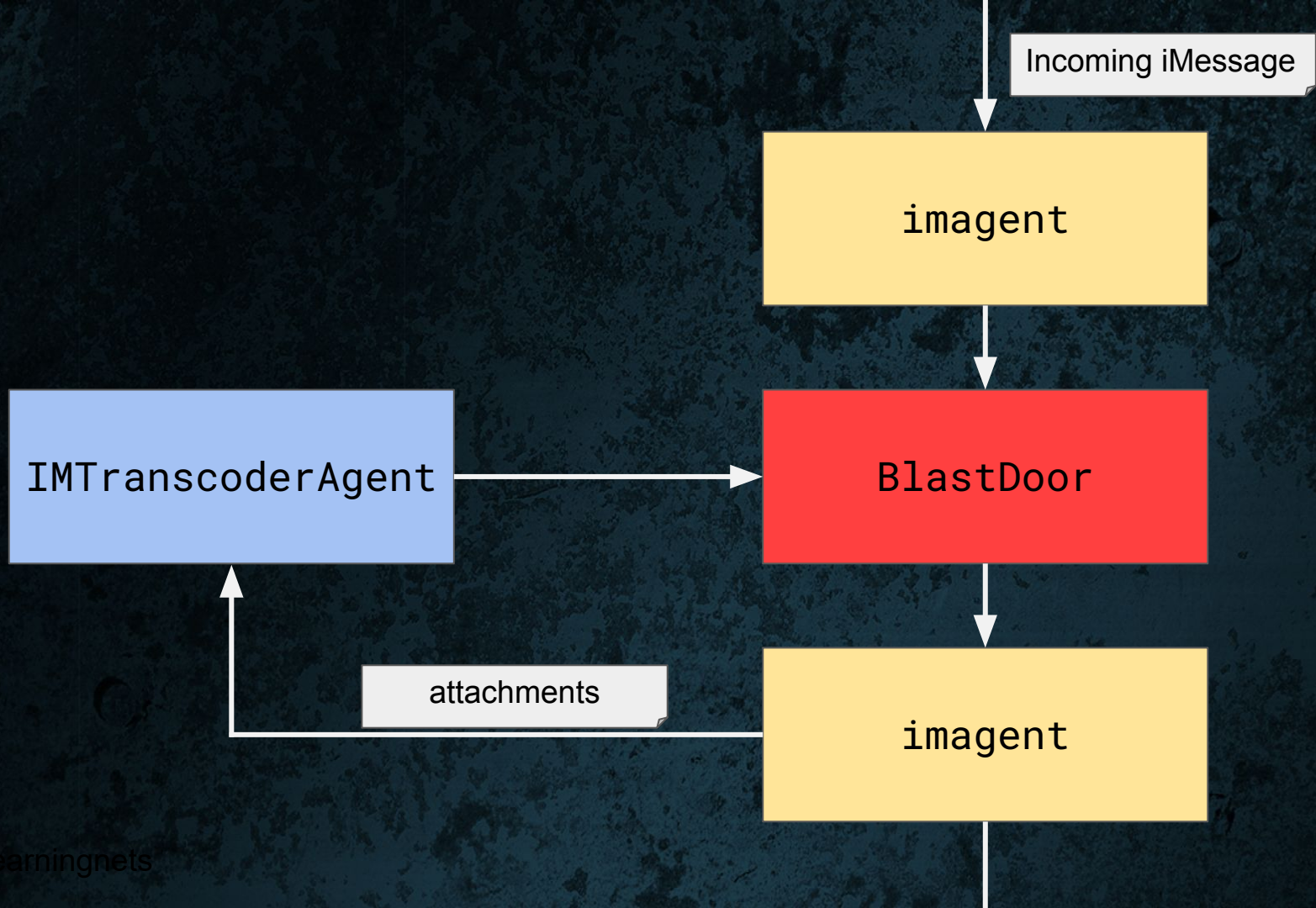
ForcedEntry ~ 2021

<https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html>

<https://t.me/learningnets>

iMessage Exploit Flow ~ 2021

Attack Surface?



one_loop.gif



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 01 00 00 |2.0.....|

```

```

...
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.....|

```

Implementation of infinite loop GIF edit in iMessage:

```
[IMGIFUtils copyGifFromPath:toDestinationPath:error]

objc_msgSend(a1,
              sel_readFileProperties_fromImageSource_0LL_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 IIIOImageSource::copyProperties
15: ImageIO IIIOImageSource::getProperties
14: ImageIO IIIO_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

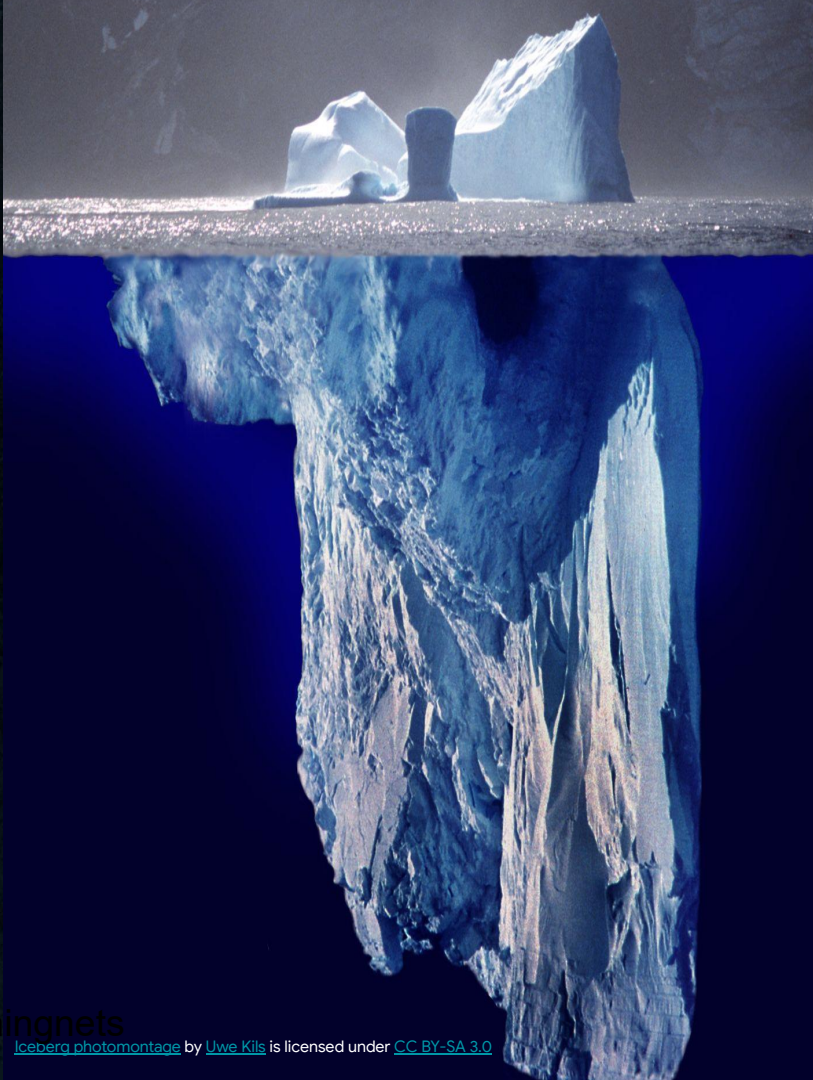
CoreGraphics

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

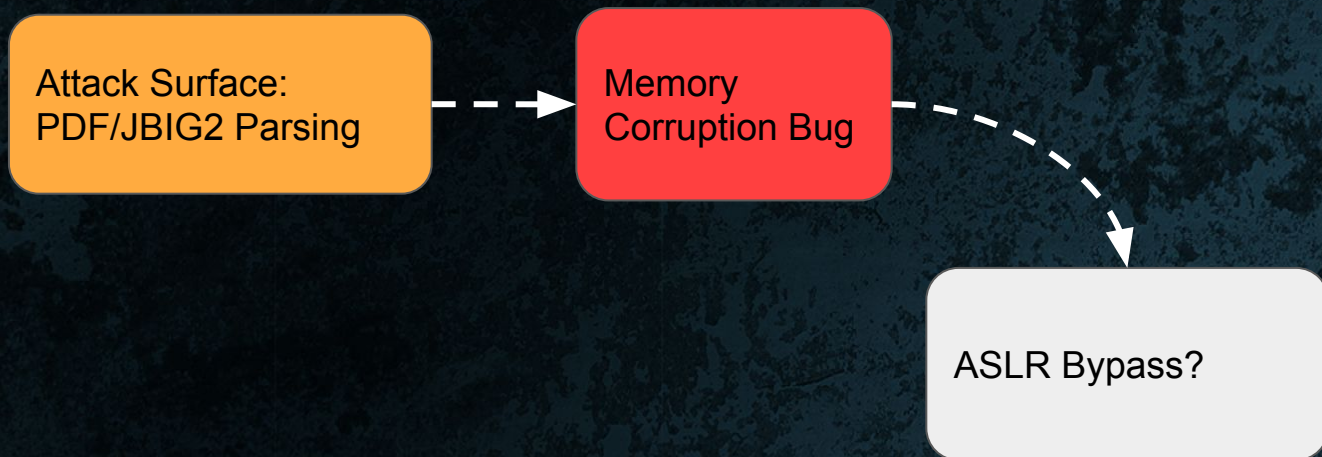
process
arbitrary JBIG2

.ai .bc .cur .exr .psd
.astc .bmp .mpo .pvr
.raw .heif .gif .pbm .tiff
.atx .icns .ktx .png .rad
.jpeg .ico .jp2 .pdf .tga

A JBIG2 heap overflow

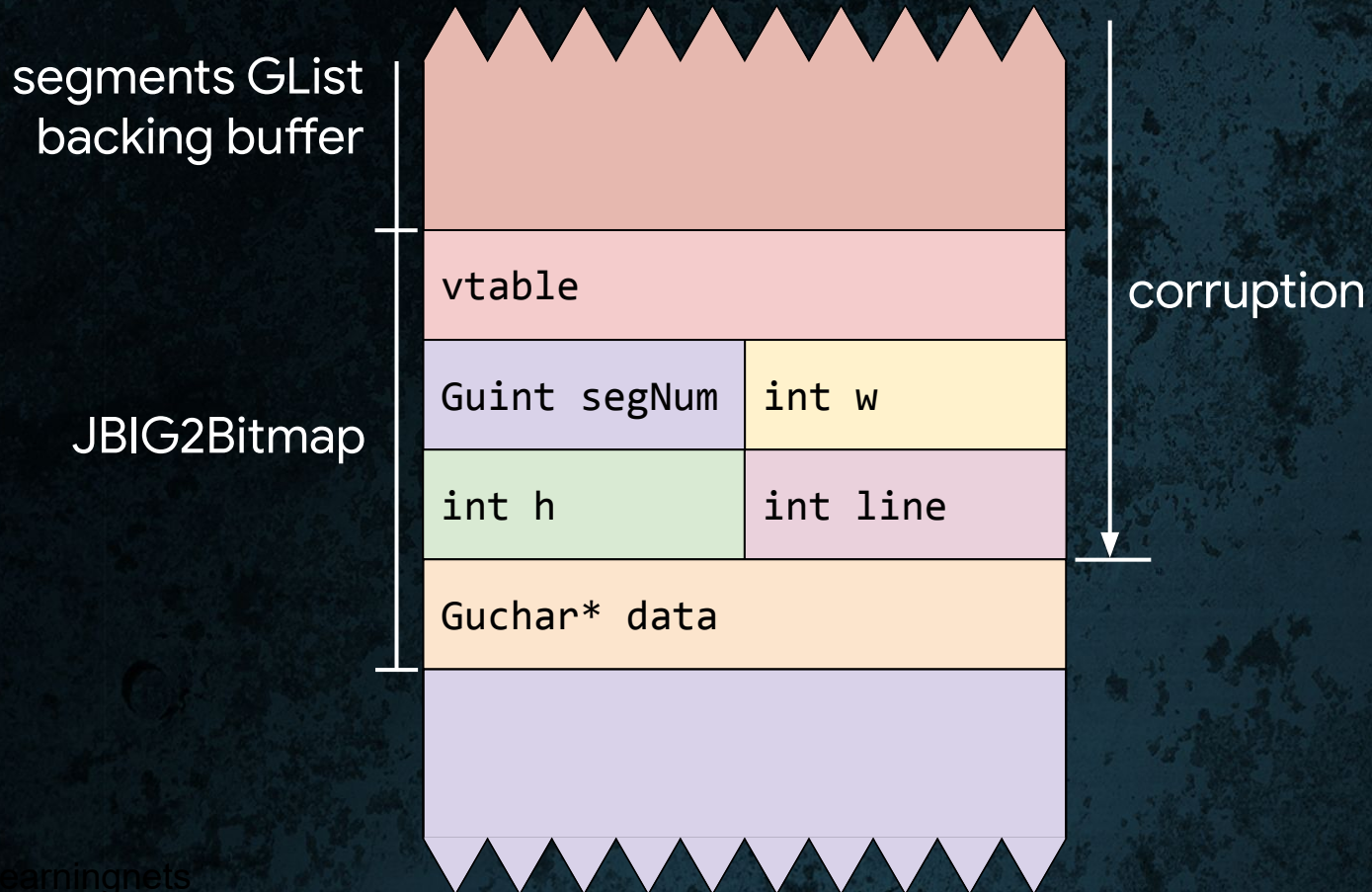
```
uint numSyms;
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            numSyms += ((JBIG2SymbolDict *)seg)->getSize();
        }
        // ...
    }
    // ...
}
syms = (JBIG2Bitmap **)gmallocn(numSyms,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

iMessage Exploit Flow ~ 2021

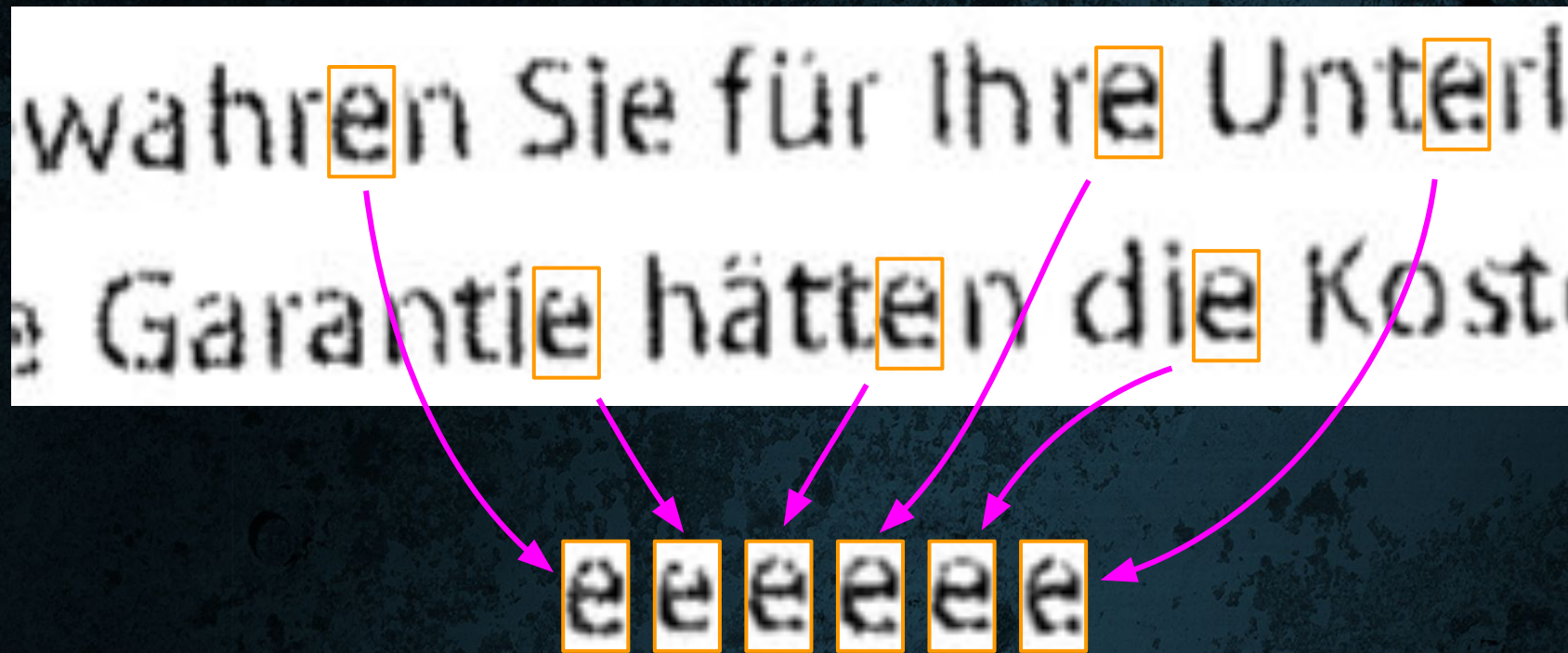


Code Execution in
IMTranscoderAgent

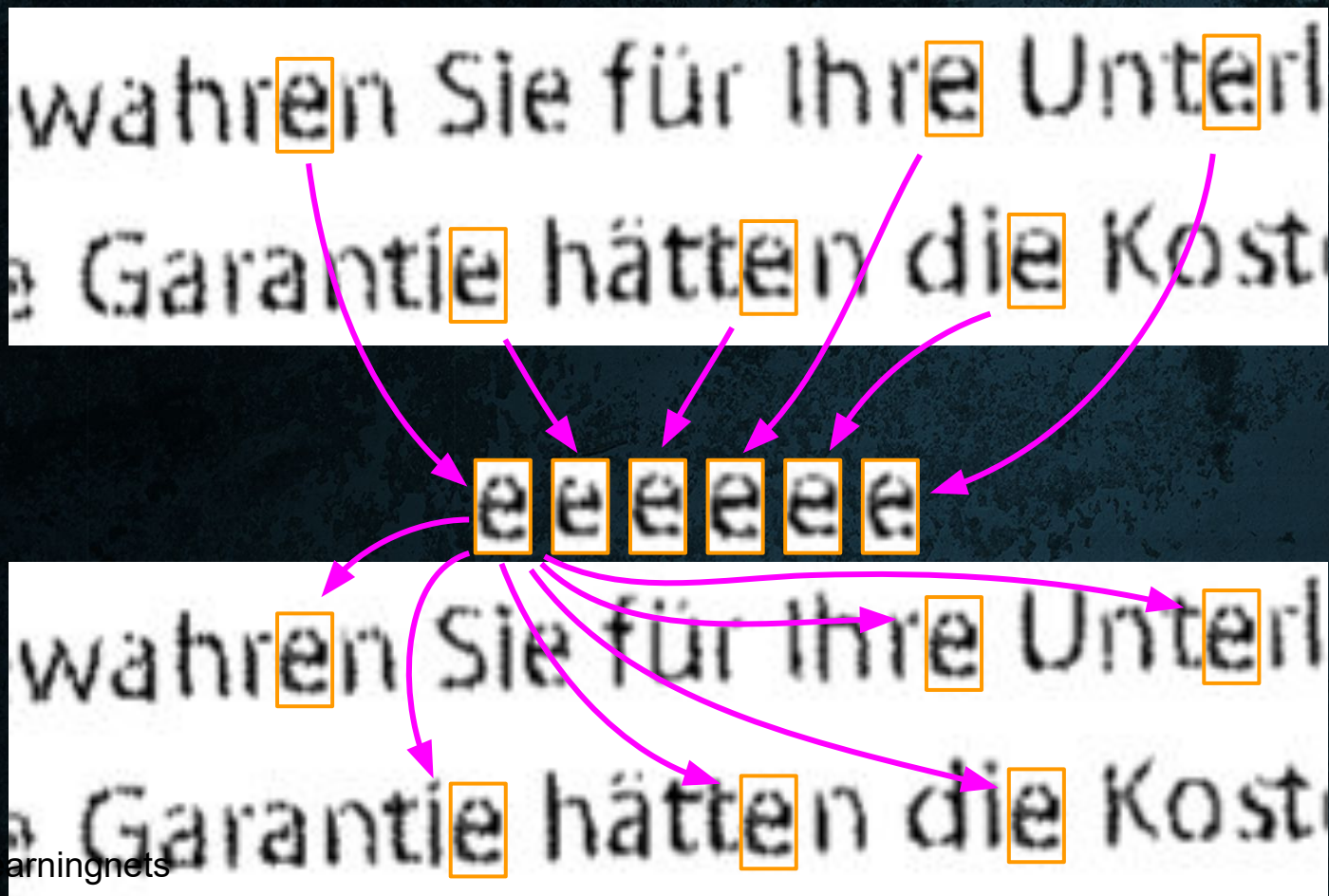
Unbounding JBIG2 canvas with a heap overflow



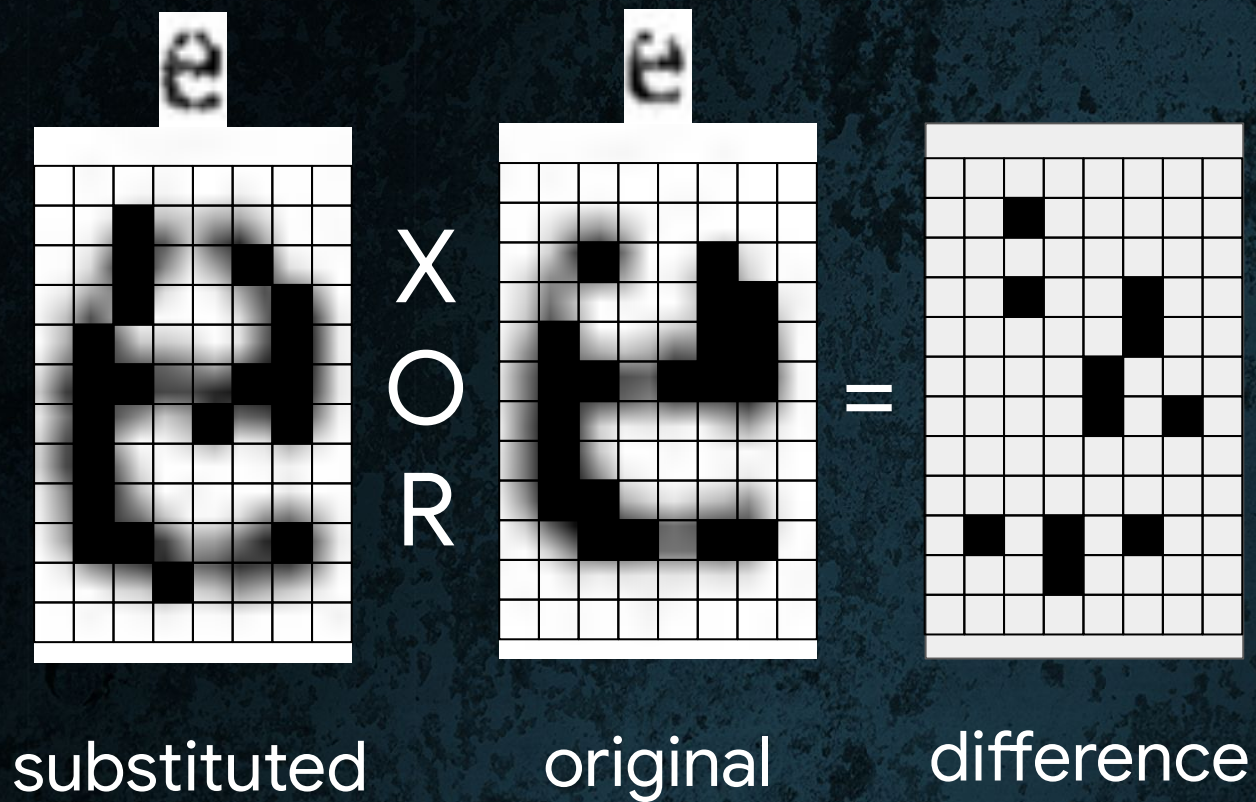
JBIG2 compression



JBIG2 compression



JBIG2 refinement operations



JBIG2 refinement operations: logic gates



JBIG2 refinement operations: NAND



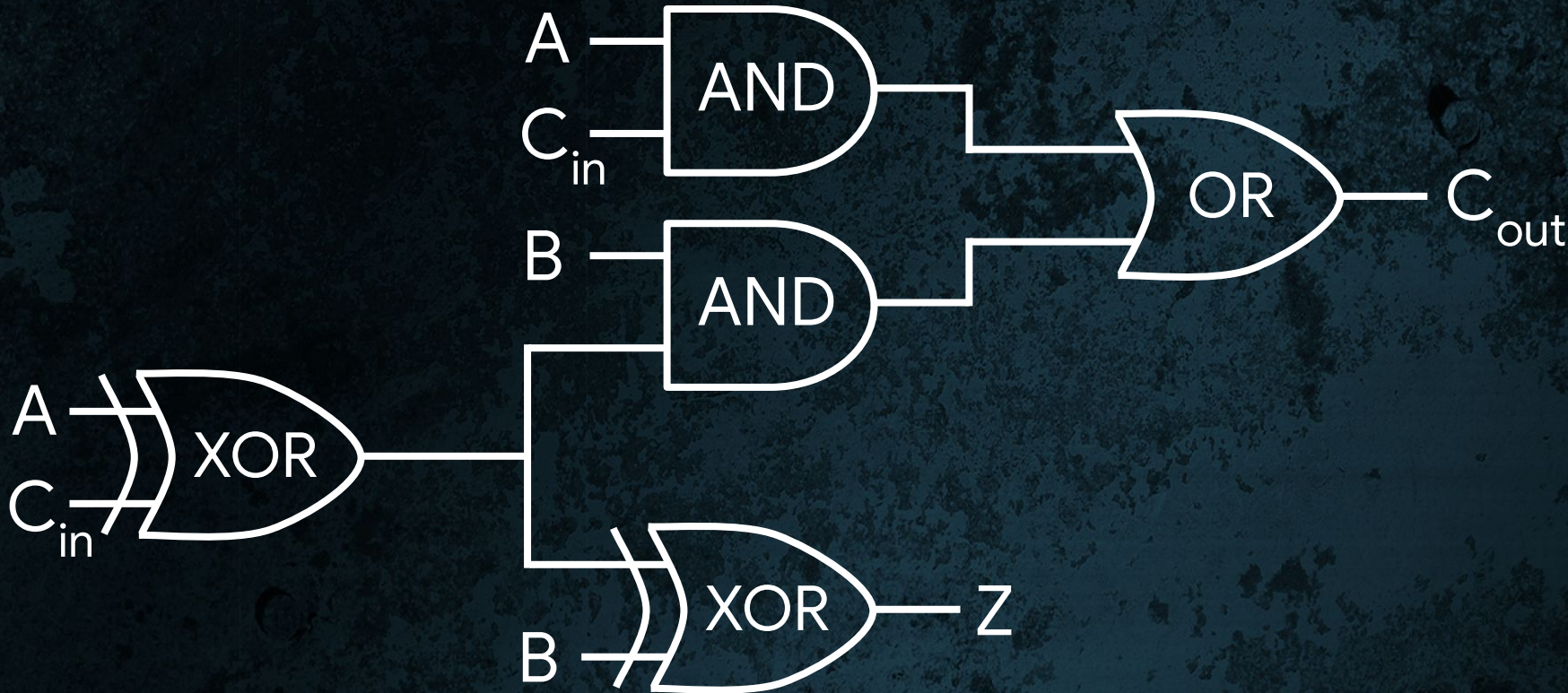
JBIG2 refinement operations: NAND

The screenshot shows the website nand2tetris.org in an Incognito browser window. The page features a navigation menu on the left with the following items: Home, Projects, Book, Software, Demos, License, Papers, Cool Stuff, Team, Stay in Touch, and Q&A. The main content area includes the title "From Nand to Tetris" and the subtitle "Building a Modern Computer From First Principles". Below this, it states "The official website of Nand to Tetris courses" and mentions the book "The Elements of Computing Systems" by Noam Nisan and Shimon Schocken (MIT Press). A video player is embedded, showing a hand-drawn diagram of a circuit with components labeled NAND, OR, AND, and MUX. The video player has a play button in the center and navigation controls at the bottom.

JBIG2 refinement NAND

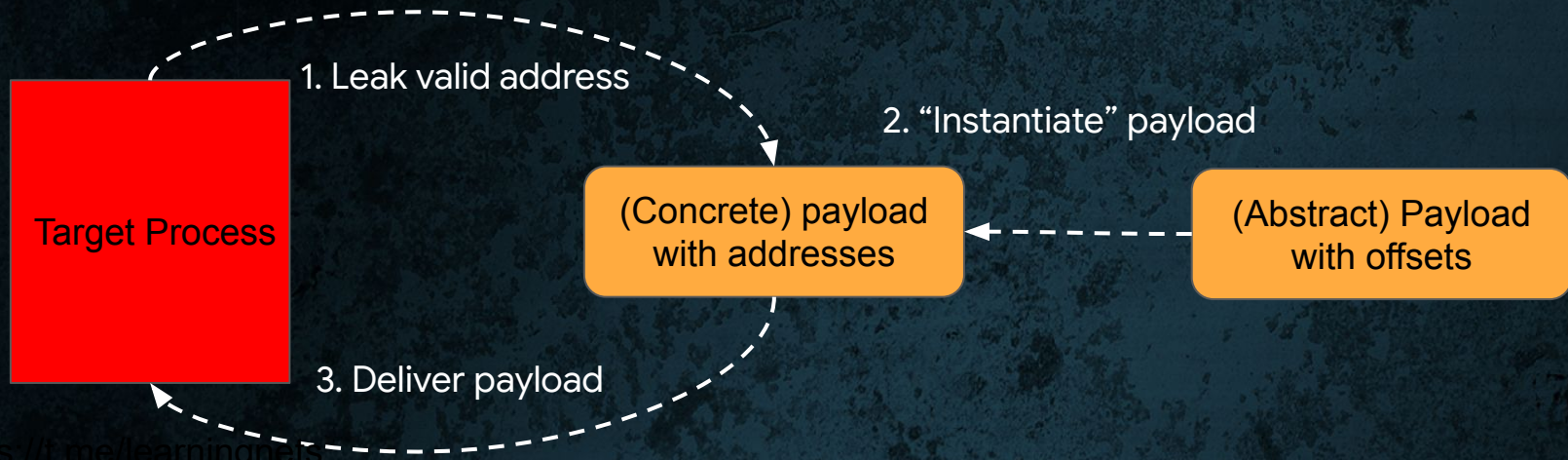


JBIG2 refinement operations: ripple carry adder

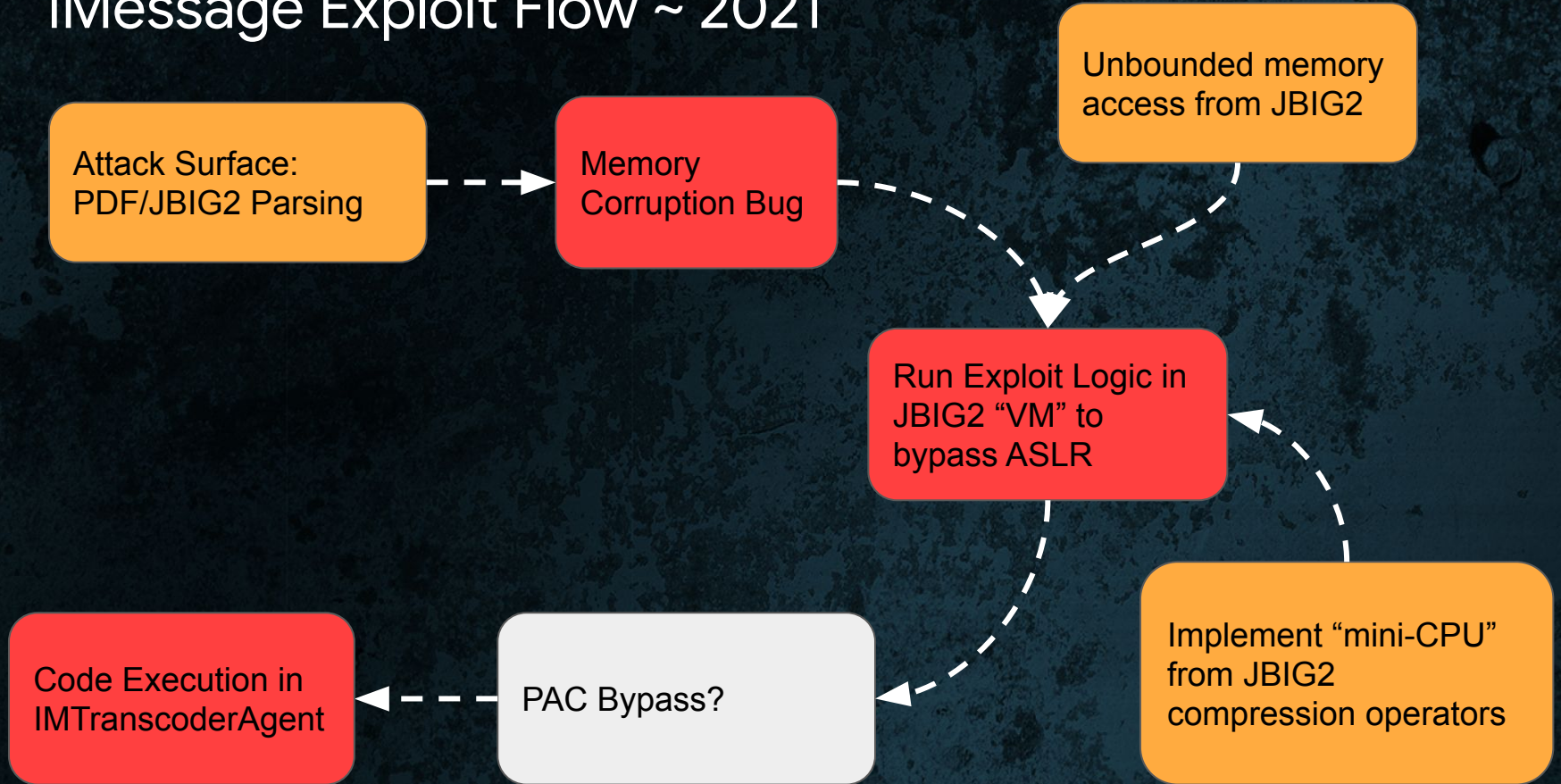


Why is ASLR a Problem?

- Need communication channel between target process and exploit logic
- **Now: Exploit logic implemented in JBIG2 VM => runs inside target process**
- Explicit communication channel with attacker machine not necessary



iMessage Exploit Flow ~ 2021



Class

NSEExpression

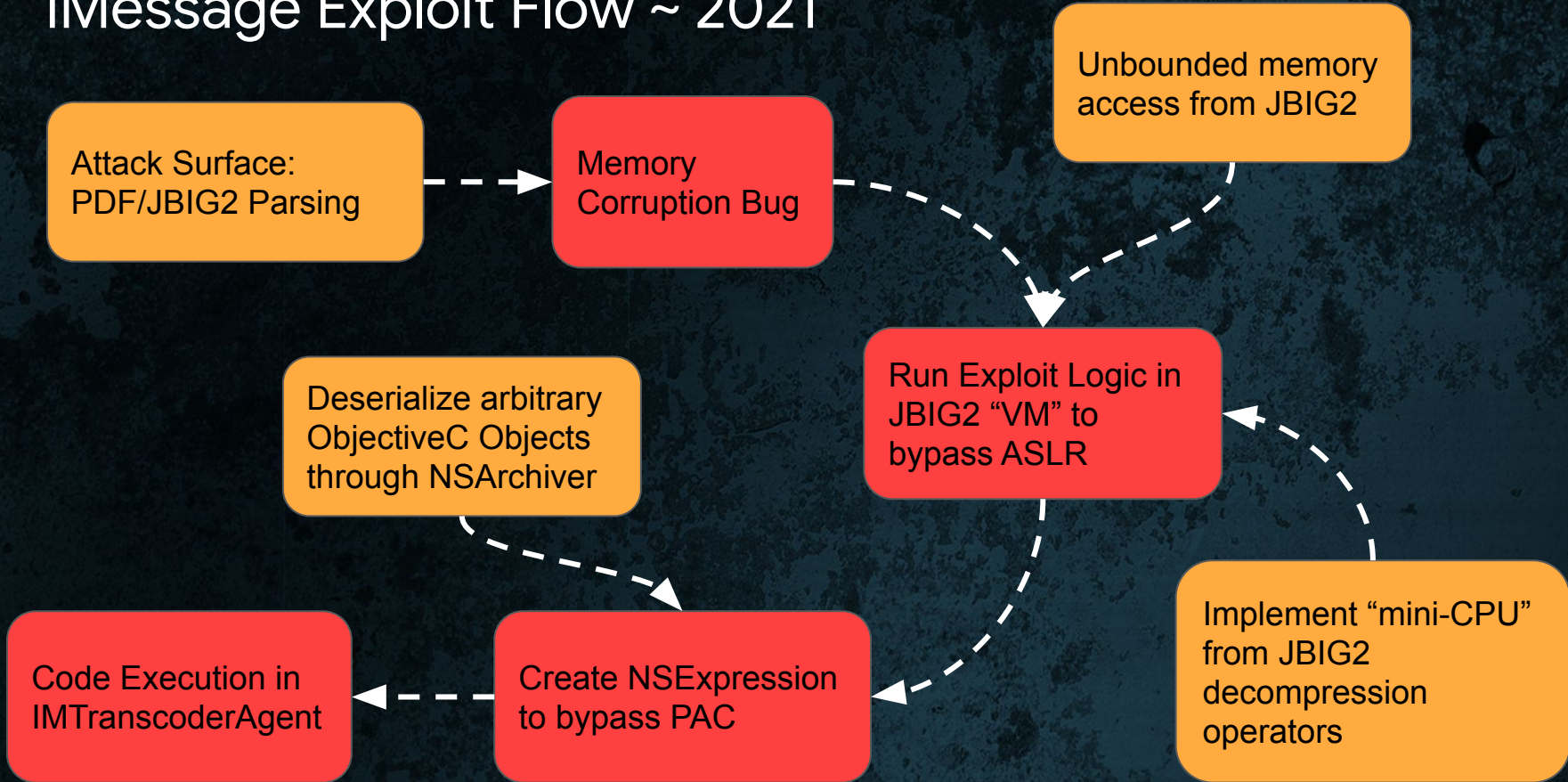
...

Function Expressions

In macOS 10.5 and later, function expressions also support arbitrary method invocations. To implement this extended functionality, use the syntax `FUNCTION(receiver, selector Name, arguments, ...)`, as in the following example:

```
FUNCTION(@"/Developer/Tools/otest", @"lastPathComponent") => @"otest"
```

iMessage Exploit Flow ~ 2021



Conclusion

- The right mitigations/hardenings can make a big difference
- Still: should assume memory corruption bugs to be exploitable unless proven otherwise (this is hard...)
- Sometimes not trivial to reason about where code executes
- Look out for hidden attack surface