# The Recent iOS 0-Click, CVE-2021-30860, Sounds Familiar. An Unreleased Write-up: One Year Later

blog.zecops.com/research/the-recent-ios-0-click-cve-2021-30860-sounds-familiar-an-unreleased-write-up-one-year-later/

By ZecOps Research Team September 14, 2021


## TLDR;

ZecOps identified and reproduced an Out-Of-Bounds Write vulnerability that can be triggered by opening a malformed PDF. This vulnerability reminded us of the FORCEDENTRY vulnerability exploited by NSO/Pegasus according to the CitizenLabs blog.

As a brief background: ZecOps have analyzed several devices of Al-Jazeera journalists in the summer 2020 and automatically and successfully found compromised devices without relying on any IOC. These attacks were later attributed to NSO / Pegasus.
ZecOps Mobile EDR and Mobile XDR are available here.

Noteworthy, although these two vulnerabilities are different – they are close enough and worth a deeper read.

**Timeline:**

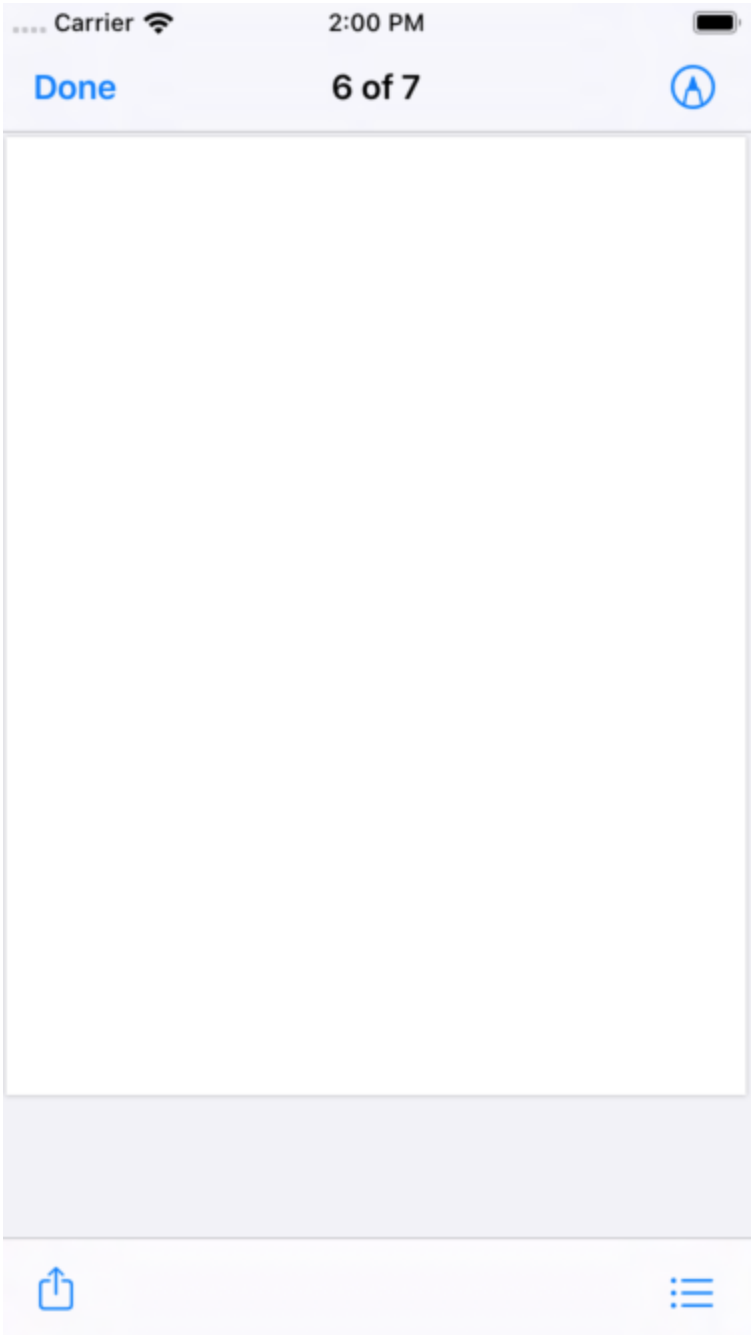
- We reported this vulnerability on **September 1st, 2020** – iOS 14 beta was vulnerable at the time.
- The vulnerability was patched on **September 14th, 2020** – iOS 14 beta release.
- Apple contacted us on **October 20, 2020** – claiming that the bug was already fixed – ("*We were unable to reproduce this issue using any current version of iOS 14. Are you able to reproduce this issue using any version of iOS 14? If so, we would appreciate any additional information you can provide us, such as an updated proof-of-concept.*"). No CVE was assigned.

It is possible that NSO noticed this incremental bug fix, and dived deeper into CoreGraphics.

## The Background

Earlier last year, we obtained a PDF file that cannot be previewed on iOS. The PDF sample crashes previewUI with segmentation fault, meaning that a memory corruption was triggered by the PDF.

Open the PDF previewUI flashes and shows nothing:




The important question is: how do we find out the source of the memory corruption?

The MacOS preview works fine, no crash. Meaning that it’s the iOS library that might have an issue. We confirmed the assumption with the iPhone Simulator, since the crash happened on the iPhone Simulator.

```
Process:               com.apple.quicklook.extension.previewUI [31594]
Path:                  /Library/Developer/CoreSimulator/Profiles/Runtimes/
iOS 13.0.simruntime/Contents/Resources/RuntimeRoot/System/Library/
Frameworks/QuickLook.framework/PlugIns/
com.apple.quicklook.extension.previewUI.appex/
com.apple.quicklook.extension.previewUI
Identifier:            com.apple.quicklook.extension.previewUI
Version:               1.0 (1)
Code Type:             X86-64 (Native)
Parent Process:        launchd_sim [31384]
Responsible:           SimulatorTrampoline [31272]
User ID:               501

Date/Time:             2020-12-03 13:57:24.425 +0800
OS Version:            Mac OS X 10.15.7 (19H15)
Report Version:        12
Anonymous UUID:        DFFA60BB-5564-CEEF-C2AF-07F69D2D662D


Time Awake Since Boot: 44000 seconds

System Integrity Protection: disabled

Crashed Thread:        0  Dispatch queue: com.apple.main-thread

Exception Type:        EXC_BAD_INSTRUCTION (SIGILL)
Exception Codes:       0x0000000000000001, 0x0000000000000000
Exception Note:        EXC_CORPSE_NOTIFY

Termination Signal:    Illegal instruction: 4
Termination Reason:    Namespace SIGNAL, Code 0x4
Terminating Process:   exc handler [31594]

Application Specific Information:
CoreSimulator 732.18.0.2 - Device: iPhone8_13.0 (F6EEA0E3-F88E-4327-
A308-0FD1F63C2C16) - Runtime: iOS 13.0 (17A577) - DeviceType: iPhone 8
```

It's great news since Simulator on MacOS provides better debug tools than iOS. However, having debug capability is not enough since the process crashes only when the corrupted memory is being used, which is AFTER the actual memory corruption.

We need to find a way to trigger the crash right at the point the memory corruption happens.

The idea is to leverage Guard Malloc or Valgrind, making the process crash right at the memory corruption occurs.

> "Guard Malloc is a special version of the malloc library that replaces the standard library during debugging. Guard Malloc uses several techniques to try and crash your application at the specific point where a memory error occurs. For example, it places separate memory allocations on different virtual memory pages and then deletes the entire page when the memory is freed. Subsequent attempts to access the deallocated memory cause an immediate memory exception rather than a blind access into memory that might now hold other data."

## Environment Variables Injection

In this case we cannot simply add an environment variable with the command line since the previewUI launches on clicking the PDF which does not launch from the terminal, we need to inject libgmalloc before the launch.

The process “launchd_sim” launches Simulator XPC services with a trampoline process called “xpcproxy_sim”. The “xpcproxy_sim” launches target processes with a posix_spawn system call, which gives us an opportunity to inject environment variables into the target process, in this case “com.apple.quicklook.extension.previewUI”.

The following lldb command “process attach –name xpcproxy_sim –waitfor” allows us to attach xpcproxy_sim then set a breakpoint on posix_spawn once it’s launched.

```
(lldb) process attach --name xpcproxy_sim --waitfor
Process 44868 stopped
* thread #1, stop reason = signal SIGSTOP
    frame #0: 0x000000011708314a dyld`__mmap + 10
dyld`__mmap:
->  0x11708314a <+10>: jae    0x117083154               ; <+20>
    0x11708314c <+12>: movq   %rax, %rdi
    0x11708314f <+15>: jmp    0x117081408               ; cerror_nocancel
    0x117083154 <+20>: retq
Target 0: (xpcproxy_sim) stopped.

Executable module set to "/Library/Developer/CoreSimulator/Profiles/Runtimes/iOS 13.0.simruntime/
Contents/Resources/RuntimeRoot/usr/libexec/xpcproxy_sim".
Architecture set to: x86_64-apple-ios-simulator.
(lldb) b posix_spawn
Breakpoint 1: no locations (pending).
WARNING:  Unable to resolve breakpoint to any actual locations.
(lldb) c
Process 44868 resuming
1 location added to breakpoint 1
Process 44868 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x00000010c6ae131 libsystem_kernel.dylib`posix_spawn
libsystem_kernel.dylib`posix_spawn:
->  0x10c6ae131 <+0>: pushq  %rbp
    0x10c6ae132 <+1>: movq   %rsp, %rbp
    0x10c6ae135 <+4>: pushq  %r15
    0x10c6ae137 <+6>: pushq  %r14
Target 0: (xpcproxy_sim) stopped.
```

Once the posix_spawn breakpoint is hit, we are able to read the original environment variables by reading the address stored in the $r9 register.

By a few simple lldb expressions, we are able to overwrite one of the environment variables into “DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib”, injection complete.

```
(lldb) x/50gx $r9
0x7f8692d00710: 0x00007f869380179c 0x00007f869380180f
0x7f8692d00720: 0x00007f8693801896 0x00007f86938018f8
0x7f8692d00730: 0x00007f8693801983 0x00007f86938019ff
0x7f8692d00740: 0x00007f8693801a27 0x00007f8693801ac8
0x7f8692d00750: 0x00007f8693801ae9 0x00007f8693801b87
0x7f8692d00760: 0x00007f8693801c38 0x00007f8693801c9b
0x7f8692d00770: 0x00007f8693801cc2 0x00007f8693801d37
0x7f8692d00780: 0x00007f8693801f62 0x00007f8693801f86
0x7f8692d00790: 0x00007f8693801ffb 0x00007f869380207f
0x7f8692d007a0: 0x00007f869380213f 0x00007f869380215e
0x7f8692d007b0: 0x00007f869380222e 0x00007f86938022ea
0x7f8692d007c0: 0x00007f869380230f 0x00007f8693802335
0x7f8692d007d0: 0x00007f8693802354 0x00007f869380237a
0x7f8692d007e0: 0x00007f8693802396 0x00007f86938023b5
0x7f8692d007f0: 0x00007f86938023cc 0x00007f8693802477
0x7f8692d00800: 0x00007f869380249a 0x00007f869380255f
0x7f8692d00810: 0x00007f8693802583 0x00007f86938025b7
0x7f8692d00820: 0x00007f8693802659 0x00007f8693802712
0x7f8692d00830: 0x00007f86938027be 0x00007f8693802870
0x7f8692d00840: 0x00007f86938028a9 0x00007f86938029ba
0x7f8692d00850: 0x00007f8692d008a8 0x0000000000000000
0x7f8692d00860: 0x00007f8693802aaa 0x00007f8692d008a8
0x7f8692d00870: 0x0000000000000000 0x0000000000000000
0x7f8692d00880: 0x0000000000000000 0x0000000000000000
0x7f8692d00890: 0x0000000000000000 0x0000000000000000
(lldb) expression (char*)malloc(50)
(char *) $1 = 0x00007f8692e00000 <no value available>
(lldb) expression (void)strcpy($1,"DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib")
(lldb) memory write 0x7f8692d00850 -s 8 0x00007f8692e00000
(lldb)
```

Continuing execution, the process crashed almost right away.

```
(lldb) c
Process 44868 resuming
1 location added to breakpoint 1
Process 44868 stopped
* thread #8, queue = 'PDFKit.PDFPageBackgroundManager', stop reason = EXC_BAD_ACCESS (code=1, add
ress=0x6000571b1000)
    frame #0: 0x00007fff51417867 libsystem_platform.dylib`_platform_memmove$VARIANT$Nehalem + 71
libsystem_platform.dylib`_platform_memmove$VARIANT$Nehalem:
->  0x7fff51417867 <+71>: movq   %rcx, (%rdi)
    0x7fff5141786a <+74>: addq   $0x8, %rdi
    0x7fff5141786e <+78>: subq   $0x8, %rdx
    0x7fff51417872 <+82>: jae    0x7fff51417860            ; <+64>
Target 0: (com.apple.quicklook.extension.previewUI) stopped.
```

```
(lldb) x/32gx 0x6000571b0f80
0x6000571b0f80: 0x0000000000000000 0x0000000000000000
0x6000571b0f90: 0x0000000000000000 0x0000000000000000
0x6000571b0fa0: 0x0000000000000000 0x0000000000000000
0x6000571b0fb0: 0x0000000000000000 0x0000000000000000
0x6000571b0fc0: 0x0000000000000000 0x0000000000000000
0x6000571b0fd0: 0x0000000000000000 0x0000000000000000
0x6000571b0fe0: 0x0000000000000000 0xdeadbeefdeadbeef
0x6000571b0ff0: 0x0000000000000000 0x0000000000000000
0x6000571b1000: 0x0000000000000000 0x0000000000000000
0x6000571b1010: 0x0000000000000000 0x0000000000000000
0x6000571b1020: 0x0000000000000000 0x0000000000000000
0x6000571b1030: 0x0000000000000000 0x0000000000000000
0x6000571b1040: 0x0000000000000000 0x0000000000000000
0x6000571b1050: 0x0000000000000000 0x0000000000000000
0x6000571b1060: 0x0000000000000000 0x0000000000000000
0x6000571b1070: 0x0000000000000000 0x0000000000000000
```

## Analyzing the Crash

Finally we got the Malloc Guard working as expected, the previewUI crashes right at the memmove function that triggers the memory corruption.

After libgmalloc injection we have the following backtrace that shows an **Out-Of-Bounds write** occurs in "**CGDataProviderDirectGetBytesAtPositionInternal**".

```
Thread 3 Crashed:: Dispatch queue: PDFKit.PDFTilePool.workQueue
0   libsystem_platform.dylib            0x0000000106afc867
_platform_memmove$VARIANT$Nehalem + 71
1   com.apple.CoreGraphics              0x0000000101b44a98
CGDataProviderDirectGetBytesAtPositionInternal + 179
2   com.apple.CoreGraphics              0x0000000101d125ab
provider_for_destination_get_bytes_at_position_inner + 562
3   com.apple.CoreGraphics              0x0000000101b44b09
CGDataProviderDirectGetBytesAtPositionInternal + 292
4   com.apple.CoreGraphics              0x0000000101c6c60c
provider_with_softmask_get_bytes_at_position_inner + 611
5   com.apple.CoreGraphics              0x0000000101b44b09
CGDataProviderDirectGetBytesAtPositionInternal + 292
6   com.apple.CoreGraphics              0x0000000101dad19a get_chunks_direct + 242
7   com.apple.CoreGraphics              0x0000000101c58875 img_raw_read + 1470
8   com.apple.CoreGraphics              0x0000000101c65611 img_data_lock + 10985
9   com.apple.CoreGraphics              0x0000000101c6102f CGSImageDataLock + 1674
10  com.apple.CoreGraphics              0x0000000101a2479e ripc_AcquireRIPImageData +
875
11  com.apple.CoreGraphics              0x0000000101c8399d ripc_DrawImage + 2237
12  com.apple.CoreGraphics              0x0000000101c68d6f
CGContextDrawImageWithOptions + 1112
13  com.apple.CoreGraphics              0x0000000101ab7c94
CGPDFDrawingContextDrawImage + 752
```

With the same method, we can take one step further, with the **MallocStackLogging** flag libgmalloc provides, we can track the function call stack at the time of each allocation.

After setting the “MallocStackLoggingNoCompact=1”, we got the following backtrace showing that the allocation was inside **CGDataProviderCreateWithSoftMaskAndMatte**.

```
ALLOC 0x6000ec9f9ff0-0x6000ec9f9fff [size=16]:
0x7fff51c07b77 (libsystem_pthread.dylib) start_wqthread |
0x7fff51c08a3d (libsystem_pthread.dylib) _pthread_wqthread |
0x7fff519f40c4 (libdispatch.dylib) _dispatch_workloop_worker_thread |
0x7fff519ea044 (libdispatch.dylib) _dispatch_lane_invoke |
0x7fff519e9753 (libdispatch.dylib) _dispatch_lane_serial_drain |
0x7fff519e38cb (libdispatch.dylib) _dispatch_client_callout |
0x7fff519e2951 (libdispatch.dylib) _dispatch_call_block_and_release |
0x7fff2a9df04d (com.apple.PDFKit) __71-[PDFPageBackgroundManager
forceUpdateActivePageIndex:withMaxDuration:]_block_invoke |
0x7fff2a9dfe76 (com.apple.PDFKit) -[PDFPageBackgroundManager
_drawPageImage:forQuality:] |
0x7fff2aa23b85 (com.apple.PDFKit) -[PDFPage imageOfSize:forBox:withOptions:] |
0x7fff2aa23e1e (com.apple.PDFKit) -[PDFPage
_newCGImageWithBox:bitmapSize:scale:offset:backgroundColor:withRotation:withAntialiasi
 |
0x7fff2aa22a40 (com.apple.PDFKit) -[PDFPage
_drawWithBox:inContext:withRotation:isThumbnail:withAnnotations:withBookmark:withDeleg
 |
0x7fff240bdfe0 (com.apple.CoreGraphics) CGContextDrawPDFPage |
0x7fff240bdac4 (com.apple.CoreGraphics) CGContextDrawPDFPageWithDrawingCallbacks |
0x7fff244bb0b1 (com.apple.CoreGraphics) CGPDFScannerScan | 0x7fff244bab02
(com.apple.CoreGraphics) pdf_scanner_handle_xname |
0x7fff2421e73c (com.apple.CoreGraphics) op_Do |
0x7fff2414dc94 (com.apple.CoreGraphics) CGPDFDrawingContextDrawImage |
0x7fff242fed6f (com.apple.CoreGraphics) CGContextDrawImageWithOptions |
0x7fff2431999d (com.apple.CoreGraphics) ripc_DrawImage |
0x7fff240ba79e (com.apple.CoreGraphics) ripc_AcquireRIPImageData |
0x7fff242f6fe8 (com.apple.CoreGraphics) CGSImageDataLock |
0x7fff242f758b (com.apple.CoreGraphics) img_image |
0x7fff24301fe2 (com.apple.CoreGraphics) CGDataProviderCreateWithSoftMaskAndMatte |
0x7fff51bddad8 (libsystem_malloc.dylib) calloc |
0x7fff51bdd426 (libsystem_malloc.dylib) malloc_zone_calloc
```

## The Vulnerability

The OOB-Write vulnerability happens in the function "**CGDataProviderDirectGetBytesAtPositionInternal**" of **CoreGraphics** library, the allocation of the target memory was inside the function "**CGDataProviderCreateWithSoftMaskAndMatte**".

```
size_t __usercall CGDataProviderDirectGetBytesAtPositionInternal@<rax>(_QWORD *a1@<rdi>, char *dst@<rsi>, signed __int64 a3@<rdx>, size_t a
{
  size_t len; // r13
  size_t size; // rbx
  signed __int64 pos; // r12
  __int64 v14; // rax
  __int64 (__fastcall *v15)(_QWORD, char *, size_t, __int64); // r8
  signed __int64 v16; // rax
  __int64 v17; // rbx
  unsigned __int64 v18; // rcx
  unsigned __int64 v19; // rax
  __int64 v20; // r8
  __int64 v21; // r9
  __m128 v22; // xmm4
  __m128 v23; // xmm5
  char v25; // [rsp-8h] [rbp-30h]

  v25 = a5;
  if ( a3 < 0 )
    __assert_rtn(
      "CGDataProviderDirectGetBytesAtPositionInternal",
      "/BuildRoot/Library/Caches/com.apple.xbs/Sources/CoreGraphics_Sim/CoreGraphics-1265.9/CoreGraphics/DataManagers/CGDataProvider.c",
      601,
      "pos >= 0");
  len = a4;
  if ( a4 )
  {
    if ( !a1 || (size = a1[4], size == -1LL) )
      __assert_rtn(
        "CGDataProviderDirectGetBytesAtPositionInternal",
        "/BuildRoot/Library/Caches/com.apple.xbs/Sources/CoreGraphics_Sim/CoreGraphics-1265.9/CoreGraphics/DataManagers/CGDataProvider.c",
        605,
        "size != (-1)");
    pos = a3;
    v14 = CGDataProviderRetainBytePtr((__int64)a1);
    if ( v14 )
    {
      if ( len + pos <= size || (len = size - pos, (signed __int64)size > pos) )// size is bigger than actual allocated memory
      {
        memcpy(dst, (const void *)(pos + v14), len);// OOB here
LABEL_25:
        CGDataProviderReleaseBytePtr((__int64)a1);
        return len;
      }
    }
    else
```

It allocates 16 bytes of memory if the “**bits_per_pixel**” equals or less than 1 byte, which is less than copy length.

```
1032    bits_per_pixel = (unsigned __int64)(v186
1033                                        * CGBitmapPixelInfoGetBitsPerPixel(
1034                                            (unsigned __int64)&v192,
1035                                            (signed int)v228 + 304,
1036                                            v181,
1037                                            0,
1038                                            v182,
1039                                            v183,
1040                                            v189,
1041                                            v190,
1042                                            v191)
1043                                        + 7) >> 3;
1044  }
1045  if ( v184 > bits_per_pixel )
1046    bits_per_pixel = v184;
1047  alloc_size = (bits_per_pixel + 15) & 0xFFFFFFFFFFFFFFF0LL;// alloc_size is 16 if (bits_per_pixel <= 1)
1048  *((_QWORD *)v166 + 153) = alloc_size;
1049  v188 = calloc(1uLL, alloc_size);
1050  *((_QWORD *)v166 + 152) = v188;
```

We came out with a minimum PoC and reported to Apple on September 1st 2020, the issue was fixed on the iOS 14 release. **We will release this POC soon.**

```
1010  if ( !bits_per_pixel )
1011  {
1012    v178 = v171[8];
1013    qmemcpy(&retaddr, v204, 0x130uLL);
1014    bits_per_pixel = (unsigned __int64)(v178 * CGBitmapPixelInfoGetBitsPerPixel(&
1015  }
1016  if ( v176 > bits_per_pixel )
1017    bits_per_pixel = v176;
1018  alloc_size = (delta + bits_per_pixel + 15) & 0xFFFFFFFFFFFFFFF0LL;// the patch
1019  v171[153] = alloc_size;
1020  v180 = calloc(1uLL, alloc_size);
1021  v171[152] = v180;
1022  if ( !v180 || !v171[150] )
```

ZecOps Mobile EDR & Mobile XDR customers are protected against NSO and are well equipped to discover other sophisticated attacks including 0-days attacks.