

Matthew Sutton
12/4/2016
IASC 4540
Semester Project – Staff Study

SUBJECT: CVE-2016-4631 Tagged Image File Format (TIFF) Exploit for Apple Operating Systems Leading to Remote Code Execution and Denial of Service (DoS)

1. PROBLEM: To prevent DoS, sensitive information exposure, and unauthorized information modification, all parties must update Apple iOS, OS X, tvOS, and watchOS to their latest secure release.

2. FACTORS BEARING ON THE PROBLEM

a. Facts.

CVE-2016-4631 affects Apple iOS before 9.3.3, OS X before 10.11.6, tvOS before 9.2.2, and watchOS before 2.2.2. (“Apple Max OSX”, 2016). The security patches for this vulnerability were released by Apple on July 18th, 2016 (“Apple security updates”, 2016). The operating systems are used in the following devices: iOS for Apple phones, OS X for Mac computers/laptops, tvOS for Apple TVs, and watchOS for the Apple Watch series.

This exploit is triggered when a crafted tiled TIFF file is opened in applications using the Apple Image I/O Application Program Interface (API). This includes malicious web pages, Multimedia Messaging Service (MMS) messages, iMessages, and other means of delivering TIFF files (Bohan, 2016). Basically, all that is necessary for an attacker to perpetrate this attack is to have their victim view their crafted TIFF file in the correct setting. The vulnerability executes when a user interacts with the attack mechanism because TIFF files are automatically processed by Apple’s Image I/O API (Paganini, 2016). For example, the receiver of a malicious TIFF file via iOS MMS cannot prevent exploitation because the MMS software will automatically download and view the TIFF file using Apple Image I/O (Paganini, 2016). Here is another example for more context: those browsing the Internet using Safari on vulnerable OS X versions are vulnerable to malicious web pages hosting TIFFs. When a user browses to the malicious page hosting the TIFF file, Safari will use the Apple Image I/O API to display the TIFF subsequently triggering the exploit. The attack can be done silently without alerting a user they have been exploited. Denial of service can be achieved with the TIFF exploit through a process called heap corruption leading to the device crashing.

iOS is generally protected from this exploit code because an attacker needs further iOS jailbreak, or a root exploit, to take complete control of the mobile device (Paganini, 2016). OS X doesn’t use a sandbox protection and is vulnerable to the attack (Paganini, 2016).

National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) Common Vulnerability Scoring System version 3 ratings for this vulnerability (“Vulnerability Summary”, 2016):

- Base Score: 8.8 High
- Impact Score: 5.9
- Exploitability Score: 2.8
- Attack Vector: Network
- Attack Complexity: Low
- User Privileges Required: None
- User Interaction: Required
- Loss of Confidentiality (C): High
- Loss of Integrity (I): High
- Loss of Availability (A): High

b. Assumptions.

It is assumed some parties may not have updated their Apple devices on schedule and will still be vulnerable to this exploit. It is common for users to fail to update their software. The security policy should require mandatory security updates for all devices after thorough testing of the updates has been completed to ensure the issue has been fixed.

It is assumed most users do not have their Apple devices jailbroken. This should generally protect iOS, tvOS, and watchOS devices from the worst parts of this exploit. Jailbreaking should be banned in the security policy as is it a major security issue. Root access is given to jailbroken devices which simplifies the malicious attack vector.

It is assumed highly sensitive data and critical software applications are stored/run on Apple devices (specifically iOS and OS X) leading to grave damage if exposed to the TIFF vulnerability. Denial of service, via TIFF heap corruption, can lead to loss of availability and lost revenue from downtime costs. Sensitive information exposure, via remote code execution, could lead to regulatory fines and loss of confidentiality. Unauthorized information modification, via remote code execution, could lead to loss of data integrity.

c. Criteria.

The solution must eliminate the TIFF exploit.

The solution must result in a 100% device update rate to the latest secure version of Apple iOS, OS X, tvOS, and watchOS to eliminate TIFF exploit.

The solution must check for a 100% compliance against jailbroken devices. This is an extra step that is not required as a solution to the TIFF exploit, but is necessary to prevent any future exploit code from running with easy root access. Updating an Apple device to its latest secure version will require the device is not jailbroken. This criterion will automatically take place with the execution of criterion two.

d. Definitions.

Apple Image I/O Application Program Interface – Apple’s programming interface framework which allows applications to read and write most image file formats, manage color, and access image metadata (“API Reference Framework”, n.d.).

Denial of Service – An issue preventing users from using a service or obtaining pertinent information (usually by malicious means).

Exploit – The method used to gain unauthorized access to the device.

Heap Corruption – When the heap (an important primary memory structure used to store data) has its data overwritten.

iMessage – Apple’s custom instant messaging service for Apple devices.

Jailbreak – To remove a restricted mode of operation from a device often using third-party software. In the case of iOS devices, jailbreaking provides root access.

Loss of Availability – Loss of the ability to keep critical assets accessible, and the loss of the ability to provide essential information to those who need it (“Confidentiality, Integrity & Availability”, 2009).

Loss of Confidentiality – Loss of the ability to control or restrict access so that only authorized individuals can view sensitive information (“Confidentiality, Integrity & Availability”, 2009).

Loss of Data Integrity – Loss of the ability to prove information is accurate, reliable, and has not been subtly changed or tampered with by an unauthorized party over the course of the information’s lifespan (“Confidentiality, Integrity & Availability”, 2009).

Multimedia Messaging Service (MMS) – The technology that allows phones and devices to communicate text, pictures, video, and audio between one another.

Operating System – The underlying software on a device that allows for the device’s basic function.

Patch – An update for a product that fixes an issue or improves the product.

Regulatory Fines – Businesses are subject to following IT regulations as placed forth by state and federal government. Failure to secure data, such as in the case of Health Insurance Portability and Accountability Act (HIPPA) records, can lead to noncompliance penalties ranging from \$100 to \$50,000 per violation (or per record). Other regulations that may apply: The Sarbanes Oxley Act, Federal Information Security Management Act of 2002 (FISMA), Family Educational Rights and Privacy

Act (FERPA), Payment Card Industry Data Security Standard (PCI-DSS), and the Gramm Leach Bliley Act (GLBA) (Vanderburg, 2012).

Remote Code Execution – The ability for an attacker to execute their own malicious code on their target.

Root Access (Privilege) – Having the same level of access as the root user on a computer system (i.e. has access to all files, commands, and privileges for the system).

Root Exploit – An exploit that ends in the attacker gaining root access.

Safari –The default Apple web browser.

Sandbox Protection – A special protection when running a program that places its resources into a “walled sandbox” to control accesses in or out of its assigned space.

Tagged Image File Format (TIFF) – A type of Bitmap image storage format designed to be used in any almost operating environment. It is found on most platforms requiring image data storage (“TIFF: Summary from”, 2013). Images consist of data describing the image in the header then tags throughout the image file describing how and where the data should be displayed (Bohan, 2016).

Vulnerability – Software, hardware, or procedural weakness that could be used by an attacker to gain unauthorized access to information or information resources (Rudolph, 2014).

3. DISCUSSION.

The solution is to mandate all that iOS, OS X, tvOS, and watchOS devices be upgraded to their latest secure release. By updating all Apple devices, the TIFF exploit will be completely mitigated because this exploit was patched via Apple OS updates on July 18th, 2016 (“Apple Security Updates”, 2016). Upgrading to the latest OS version will also yield additional patches for other Apple vulnerabilities not mentioned in this staff study. This ultimately satisfies, and goes beyond, the first criterion.

Effectively mandating the upgrade process will achieve the 100% update rate as posed by the second criterion. This Apple OS upgrade order should first get approval from the Chief Information Security Officer (CISO). Updates should then be executed by a chosen task-force in which they have the proper authority, and tools, to complete the upgrade process. If there is network encompassing update management software in place (with the capability to push OS updates) it should be used to complete most of the upgrade process. The update management software approach is easiest because of the metrics the management system will provide. It is a likely possibility that some devices are not capable of being managed through update management software (such as iPhones and Apple Watches). Therefore, the task-force should also communicate with each Information System Security Officer (ISSO) to obtain a full-

encompassing inventory of Apple devices and determine which devices are unable to be pushed updates from the update management software. The task-force will then manually apply the Apple OS updates until 100% of Apple devices have been updated. Additionally, this task-force will check for, and revert, jailbroken Apple devices to their official OS. Jailbroken devices will not be able to receive OS updates because they use custom software, so criterion three will automatically be handled during the process of criterion two.

Overall, this solution covers all the criteria. The solution also leaves the organization less vulnerable to malicious attack through Apple devices. This is because we have updated all Apple products to their most secure OS version to eliminate vulnerabilities, and have removed all jailbroken Apple devices from the internal network.

4. CONCLUSION:

Get approval from the CISO to upgrade Apple devices to their latest secure version to eliminate the TIFF vulnerability present in: Apple iOS before 9.3.3, OS X before 10.11.6, tvOS before 9.2.2, and watchOS before 2.2.2. Updating to the latest version will also patch additional vulnerabilities not mentioned in this staff study.

Latest Apple OS versions as of December 4th, 2016 (“Apple Security Updates”, 2016):

- Update iOS to version 10.1.1.
- Update OS X (macOS) to 10.12.1.
- Update tvOS to 10.0.1.
- Update watchOS to 3.1.

The CISO will assign a task-force to:

- Use update management software to push OS updates to connected Apple devices.
- Communicate with all ISSOs to obtain a full-encompassing inventory of Apple devices not connected to the update management software.
- Determine which Apple devices from the inventory need the latest OS updates.
- Manually apply OS updates to these devices until the 100% device update rate has been met.
- Check Apple devices for jailbroken OS while doing manual update.

5. ACTION RECOMMENDED:

Patch the tiled TIFF exploit (CVE-2016-4631) by requesting approval/action from the CISO to create a task-force that will update all Apple devices to their latest secure OS version. Specifically: iOS to version 10.1.1, OS X (macOS) to 10.12.1, tvOS to 10.0.1, and watchOS to 3.1.

Approval: _____

DR. DWIGHT HAWORTH , Cybersecurity CISO
Professor, University of Nebraska at Omaha

\Attachments:

1. Analysis of CVE-2016-4631
2. Deep Technical Analysis of CVE-2016-4631

Attachment 1 - Analysis of CVE-2016-4631:

The discovery of CVE-2016-4631 belongs to Talos security researcher Tyler Bohan. This attachment primarily summarizes Bohan's (2016) vulnerability report due to lack of additional resources for this vulnerability. The vulnerability was tested on OSX El Capitan 10.11.4 and iOS 9.3.1 (Bohan, 2016).

Introduction to TIFFs - Strips, Tiles, Tags, and the Exploit:

TIFF bitmap image data uses two forms: strips and tiles. A strip is an individual collection of one or more contiguous rows of bitmapped image data ("TIFF: Summary from", 2013). Strips can be thought of as one-dimensional objects that only have length ("TIFF: Summary from", 2013). TIFF 6.0 introduced the concept of tiled bitmap data. Tiles can be thought of as two-dimensional strips that have both width and length ("TIFF: Summary from", 2013). Each tile in an image can be imagined as a small bitmap containing a piece of a larger bitmap ("TIFF: Summary from", 2013). Fit the tiles together in their proper locations to get the full TIFF image ("TIFF: Summary from", 2013). Tags throughout the TIFF file describe how and where this tiled/stripped data should be displayed (Bohan, 2016). Each TIFF parameter, shown below in Fig. 1., is derived from a tag with a specific identifier in the TIFF file (Bohan, 2016).

The Apple Image I/O API does correctly handle specially crafted tiled TIFFs. This can lead to an out of bounds write or heap corruption (Bohan, 2016). To begin the technical analysis, a TIFF analyzer is used to observe the parameters of Bohan's (2016) malicious TIFF file.

Fig 1. TIFF Analyzer Output for Malicious Tiled TIFF File (Bohan, 2016)

```
Key:
Notable information

. . .
TIFFFetchNormalTag: Warning, Nonstandard tile width 255, convert file.
TIFFFetchNormalTag: Warning, IO error during reading of "DocumentName"; tag ignored.
TIFFFetchNormalTag: Warning, Incompatible type for "ResolutionUnit"; tag ignored.
TIFFReadDirectory: Warning, Incorrect count for "ColorMap"; tag ignored.
TIFFReadDirectory: Warning, TIFF directory is missing required "StripByteCounts" field
, calculating from imagelength.
TIFF Directory at offset 0xa0 (160)
Image Width: 2 Image Length: 1
Tile Width: 255 Tile Length: 1024
Resolution: 72, 72
Bits/Sample: 8
Compression Scheme: None
Photometric Interpretation: separated
FillOrder: msb-to-lsb
Orientation: row 0 bottom, col 0 rhs
Samples/Pixel: 4
Rows/Strip: 1024
Planar Configuration: separate image planes
Page Number: 0-1
. . .
```

The highlighted parameters in Fig. 1. are used to conduct the exploit:

Definitions for Highlighted TIFF Parameters in Fig. 1.:

Image Width – The number of columns in the image, i.e., the number of pixels per row (Damme, n.d.).

Image Length – The number of rows of pixels in the image (Damme, n.d.).

Tile Width – The tile width in pixels. This is the number of columns in each tile. Tile Width is supposed to be a multiple of 16 (Damme, n.d.).

Tile Length – The tile length (height) in pixels. This is the number of rows in each tile. Tile Length is supposed to be a multiple of 16 (Damme, n.d.).

Samples/Pixel – An image is defined to be a rectangular array of pixels, each of which consists of one or more samples. With monochromatic data, we have one sample per pixel, and sample and pixel can be used interchangeably. Color data usually contains three samples per pixel as in an RGB scheme (Davenport & Vellon, 1987). Basically, the amount of different samples placed within a single pixel to finalize its color.

Crashing a Program with a Malicious Tiled TIFF:

When viewing the TIFF in Apple Preview, the default image and PDF viewer for OS X, we receive the following crash information:

Fig. 2. Apple Preview Software Viewing a Malicious Tiled TIFF: Crash Output (Bohan, 2016)

```
Key:
Notable information

...
Exception Type:      EXC_CRASH (SIGABRT)
Exception Codes:     0x0000000000000000, 0x0000000000000000
Exception Note:      EXC_CORPSE_NOTIFY
Application Specific Information:
abort() called
*** error for object 0x7f8a89f4e6b8: incorrect checksum for freed object - object was probably modified after being freed.
0  libsystem_kernel.dylib      0x00007fff9242df06  _pthread_kill + 10
1  libsystem_thread.dylib     0x00007fff911794ec  pthread_kill + 90
2  libsystem_c.dylib          0x00007fff95b436e7  abort + 129
3  libsystem_malloc.dylib     0x00007fff97f53396  szone_error + 626
4  libsystem_malloc.dylib     0x00007fff97f46c03  tiny_malloc_from_free_list + 1351
5  libsystem_malloc.dylib     0x00007fff97f45705  szone_malloc_should_clear + 292
6  libsystem_malloc.dylib     0x00007fff97f455a1  malloc_zone_malloc + 71
7  libsystem_malloc.dylib     0x00007fff97f440cc  malloc + 42
8  libsystem_c.dylib          0x00007fff95b29e7b  _vasprintf + 354
9  libsystem_c.dylib          0x00007fff95b211a8  asprintf + 186
10 com.apple.ImageIO.framework 0x00007fff8d394b05  ImageIOLogger + 97
11 com.apple.ImageIO.framework 0x00007fff8d33d43e  myErrorHandler + 9
12 libTIFF.dylib              0x00007fff9d4667aa  TIFFErrorExt + 179
13 libTIFF.dylib              0x00007fff9d480214  TIFFReadRawTile1 + 336
14 libTIFF.dylib              0x00007fff9d47fe78  TIFFFillTile + 384
15 libTIFF.dylib              0x00007fff9d47fc8a  TIFFReadEncodedTile + 95
16 com.apple.ImageIO.framework 0x00007fff8d33d9b8  copyImageBlockSetTiledTIFF + 1372
17 com.apple.ImageIO.framework 0x00007fff8d2f40f4  ImageProviderCopyImageBlockSetCallback + 651
18 com.apple.CoreGraphics     0x00007fff93f15cb4  CGImageProviderCopyImageBlockSetWithOptions + 132
19 com.apple.CoreGraphics     0x00007fff93f1739c  CGImageProviderCopyImageBlockSet + 205
20 com.apple.CoreGraphics     0x00007fff93f4e7fd  img_blocks_create + 517
...
```


This crash is an example of a heap overflow. Note the free heap block header at 0x7f8a89f4e6b8 has been corrupted (Bohan, 2016). The crash occurs because the heap was modified after being freed (bad checksum).

The Vulnerability:

This overflow is caused by an issue with a function in the Apple Image I/O API titled ImageIO`copyImageBlockSetTiledTIFF. Essentially, the TIFF parameters from Fig. 1. are used by the function to calloc a vulnerable buffer with an incorrect size.

Variables/Buffers Used in ImageIO`copyImageBlockSetTiledTIFF:

<i>Malicious TIFF Variables</i>	<i>Algorithm Variables/Buffers</i>
image_width = 2 image_length = 1 tile_width = 255 tile_length = 1024 num_samples = 4	tile_size = calls TIFFTileSize(tiff) = 255 width = smallest width of image_width & tile_width = 2 length = smallest length image_length & tile_length = 1 samples_x_width = num_samples * width = 8 calloc_size_1 = The size for vuln_buffer to be calloced vuln_buffer[] = Set by calloc_size_1 bytes_read[] = the amount of bytes read using TIFFReadTile() and vuln_buffer

Algorithm for ImageIO`copyImageBlockSetTiledTIFF Using Data from the Malicious TIFF:

Most important fragments outlined in yellow.

1. Call `TIFFTileSize(tiff)` and assign the return value to `tile_size`.
`TIFFTileSize()` returns what function `TIFFReadTile()` calculates as the number of bytes decoded for the TIFF. For this example, we get a `tile_size` of 255
2. Compare `tile_width (255)` and the `image_width (2)` -- default `tile_width` and `width to the smaller value: 2`
3. Compare `tile_length (1024)` and the `image_length (1)` -- default `tile_length` and `length to the smaller value: 1`
4. Calculate and `set samples_x_width by multiplying: num_samples * width.`
5. Calculate `calloc_size_1` for the vulnerable buffer by multiplying:
`samples_x_width * length`
6. `If (samples_x_width * length) is less than tile_size set calloc_size_1 equal to tile_size`, otherwise do not update the value of `calloc_size_1`

In this example, `(samples_X_width * length)` is less than `tile_size` so `calloc_size_1` is set to `tile_size = 255`

7. Allocate: `calloc()` `vuln_buffer` to size of `calloc_size_1 = 255 bytes`
8. Skipping further ahead in the function to where the overwrite happens.
9. Set `cur_sample = 0`
10. While `num_samples` does not equal `cur_sample`
 - a. Read in tile data from the TIFF image -- one tile per sample:
`bytes_read = TIFFReadTile(tuff_Strucutre, vuln_buffer, x, y, 0LL, cur_sample)`
 - b. **VULNERABLE:** Move the `vuln_buffer` forward the size of one tile (`bytes_read`)

Explanation:

The vulnerability comes in when looking back at Step 2. `width` was defaulted to a bad `image_width` of 2 using the maliciously crafted TIFF. Following the sequence outlined by the highlighted text eventually leads to `vuln_buffer` being allocated with only 255 bytes (`calloc_size_1`), enough for only one sample. Every subsequent sample (`cur_sample > 0` in the while loop) writes out of bounds and further corrupts memory. The number of samples is easily controlled in the crafted TIFF image allowing for further heap corruption (Bohan, 2016). In this example, `vuln_buffer` will write $(\text{num_samples} + 1 - 1) * 255 = 1020$ bytes of data out of bounds.

- c. Get the next sample `cur_sample++`

Recommendations:

This vulnerability can be fixed in Step 6:

If `(samples_X_width * length)` is less than `tile_size` set `calloc_size_1` equal to `tile_size`, otherwise do not update the value of `calloc_size_1`

Change:

“set `calloc_size_1` equal to `tile_size`” to:

“set `calloc_size_1` equal to `(tile_size * num_samples)`”

`(tile_size * num_samples)` will make `vuln_buffer` `calloc` the correct number of bytes.

Additional Crash Information:

Fig. 3. Testing TIFF Vulnerability Against Apple QuickLook: Crash Output (qlmanage) (Bohan, 2016)

```
Key:
Notable information

...
Testing Quick Look preview with files:
  bunny.tif
May  6 16:00:54  qlmanage[55235] <Warning>: ImageIO: readAndCreateASCIIString Unable to read ASCII TIFF Tag #
269 with reported length (8)

May  6 16:00:54  qlmanage[55235] <Warning>: ImageIO: readAndCreateASCIIString Unable to read ASCII TIFF Tag #
269 with reported length (8)

May  6 16:00:54  qlmanage[55235] <Warning>: ImageIO: readAndCreateASCIIString Unable to read ASCII TIFF Tag #
269 with reported length (8)

Crashed thread log =
: Dispatch queue: com.apple.root.default-qos
0  com.apple.ColorSync          0x00007fff9357070f CollectFlattenedConversion(CMMConvNode*, CMMMemMgr*, b
ool, __CFArray*) + 49
1  com.apple.ColorSync          0x00007fff935707d4 DoFlattenFullConversion + 71
2  com.apple.ColorSync          0x00007fff93571b69 AppleCMMCreateTransformProperty + 174
3  com.apple.CoreGraphics       0x00007fff93efc8bd __get_full_conversion_code_fragment_block_invoke + 97
4  libdispatch.dylib            0x00007fff9c52b40b _dispatch_client_callout + 8
5  libdispatch.dylib            0x00007fff9c52b303 dispatch_once_f + 67
6  com.apple.CoreGraphics       0x00007fff93efc55a convert_icc + 2557
7  com.apple.CoreGraphics       0x00007fff93efbb4e CGCMSConverterConvertData + 91
8  com.apple.CoreGraphics       0x00007fff93f28b88 CGColorTransformConvertData + 381
9  com.apple.CoreGraphics       0x00007fff93f1d9ca img_colormatch_read + 582
10 com.apple.CoreGraphics       0x00007fff93f1b837 img_data_lock + 8852
11 com.apple.CoreGraphics       0x00007fff93f186c7 CGSImageDataLock + 151
12 libRIP.A.dylib               0x00007fff933ee1d4 ripc_AcquireImage + 972
13 libRIP.A.dylib               0x00007fff933ecc7e ripc_DrawImage + 1011
14 com.apple.CoreGraphics       0x00007fff93f17c48 CGContextDrawImageWithOptions + 571
15 c
```

In Fig. 3., Bohan (2016) runs an alternative test case using Apple QuickLook to scan/view a malicious TIFF. qlmanage, the Quick Look Server debug and management tool, does not appear to succumb to TIFF heap overflow. Instead, a warning is pushed stating Image I/O is unable to read the ASCII TIFF Tag #269 with reported length of (8). qlmanage then crashes and thread log is dumped.

Attachment 2 – Deep Technical Analysis of CVE-2016-4631:

The discovery of CVE-2016-4631 belongs to Talos security researcher Tyler Bohan. This attachment primarily summarizes Bohan's (2016) vulnerability report due to lack of additional resources for this vulnerability. The vulnerability was tested on OSX El Capitan 10.11.4 and iOS 9.3.1 (Bohan, 2016).

Crashing a Program via Malicious Tiled TIFF with Guard Malloc Enabled:

To gain more info about this exploit, guard malloc can be used while debugging to crash Apple Preview at the specific point where the TIFF's heap memory error occurs.

Fig. 1. Apple Preview Software Viewing a Malicious Tiled TIFF: Crash Output with Guard Malloc Enabled (Bohan, 2016)

Key:

Bohan's debugging input

Notable information

```
. . .
* thread #1: tid = 0x918fde, 0x00007fff8bd3f01c libsystem_platform.dylib`_platform_memmove$VARIANT$Haswell + 25
2, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x12d9cc000)
frame #0: 0x00007fff8bd3f01c libsystem_platform.dylib`_platform_memmove$VARIANT$Haswell + 252
libsystem_platform.dylib`_platform_memmove$VARIANT$Haswell:
-> 0x7fff8bd3f01c <+252>: vmovups %ymm0, (%rax)
0x7fff8bd3f020 <+256>: vmovups 0x20(%rsi), %ymm2
0x7fff8bd3f025 <+261>: addq $0x40, %rsi
0x7fff8bd3f029 <+265>: subq $0x80, %rdx
(lldb) register read Shows the general purpose registers for the current thread.
General Purpose Registers:
rax = 0x000000012d9cbfff
rbx = 0x000000012c62bb30
rcx = 0x0000000000000001
rdx = 0x00000000000000fe
rdi = 0x000000012c64c000
rsi = 0x000000012c659c01
rbp = 0x00007fff5fbf9d90
rsp = 0x00007fff5fbf9d90
r8 = 0x00000000000000ff
r9 = 0x00000000000000ff
r10 = 0x00000000fffff01
r11 = 0xfffffffffff23ff
r12 = 0x000000012c64bfff
r13 = 0x0000000000000001
r14 = 0x00000000000000ff
r15 = 0x00000000000000ff
rip = 0x00007fff8bd3f01c
rflags = 0x0000000000010202
(lldb) bt Shows the stack backtrace for the current thread.
frame #0: 0x00007fff8bd3f01c libsystem_platform.dylib`_platform_memmove$VARIANT$Haswell + 252
frame #1: 0x00007fff9d459283 libTIFF.dylib`TIFFmemcpy + 9
frame #2: 0x00007fff9d461870 libTIFF.dylib`DumpModeDecode + 92
frame #3: 0x00007fff9d47fc4f libTIFF.dylib`TIFFReadEncodedTile + 132
frame #4: 0x00007fff8d33d9b8 ImageIO`copyImageBlockSetTiledTIFF + 1372
frame #5: 0x00007fff8d2f40f4 ImageIO`ImageProviderCopyImageBlockSetCallback + 651
frame #6: 0x00007fff93f15cb4 CoreGraphics`CGImageProviderCopyImageBlockSetWithOptions + 132
frame #7: 0x00007fff93f1739c CoreGraphics`CGImageProviderCopyImageBlockSet + 205
(lldb) malloc_info $rax Gets information about a specific heap allocation for register RAX
0x000000012d9cbfff: malloc( 256) -> 0x12d9cbf00 + 255
(lldb) malloc_info -S $rax Gets more information about a specific heap allocation for register RAX
ColorSync Utility(54051,0x7fff79aef000) malloc: process 54687 no longer exists, stack logs deleted from /tmp/st
ack-logs.54687.100084000.ColorSync Utility.2vS0Sg.index
0x000000012d9cbfff: malloc( 256) -> 0x12d9cbf00 + 255
stack[0]: addr = 0x12d9cbf00, type=malloc, frames:
[0] 0x00007fff97f489ce libsystem_malloc.dylib`malloc_zone_malloc + 118
[1] 0x00007fff97f49462 libsystem_malloc.dylib`calloc + 49
[2] 0x00007fff8d33d815 ImageIO`copyImageBlockSetTiledTIFF + 953
[3] 0x00007fff8d2f40f4 ImageIO`ImageProviderCopyImageBlockSetCallback + 651
. . .
```

Apple Preview crashes when there is an attempt to write to memory at the address held in RAX.

Crashing Instruction:

-> 0x7fff8bd3f01c <+252>: vmovups %ymm0, (%rax)

Assembly Syntax for vmovups:

Move unaligned, packed, single-precision floating-point from ymm2/mem to ymm1.

VMOVUPS ymm1, ymm2/m256

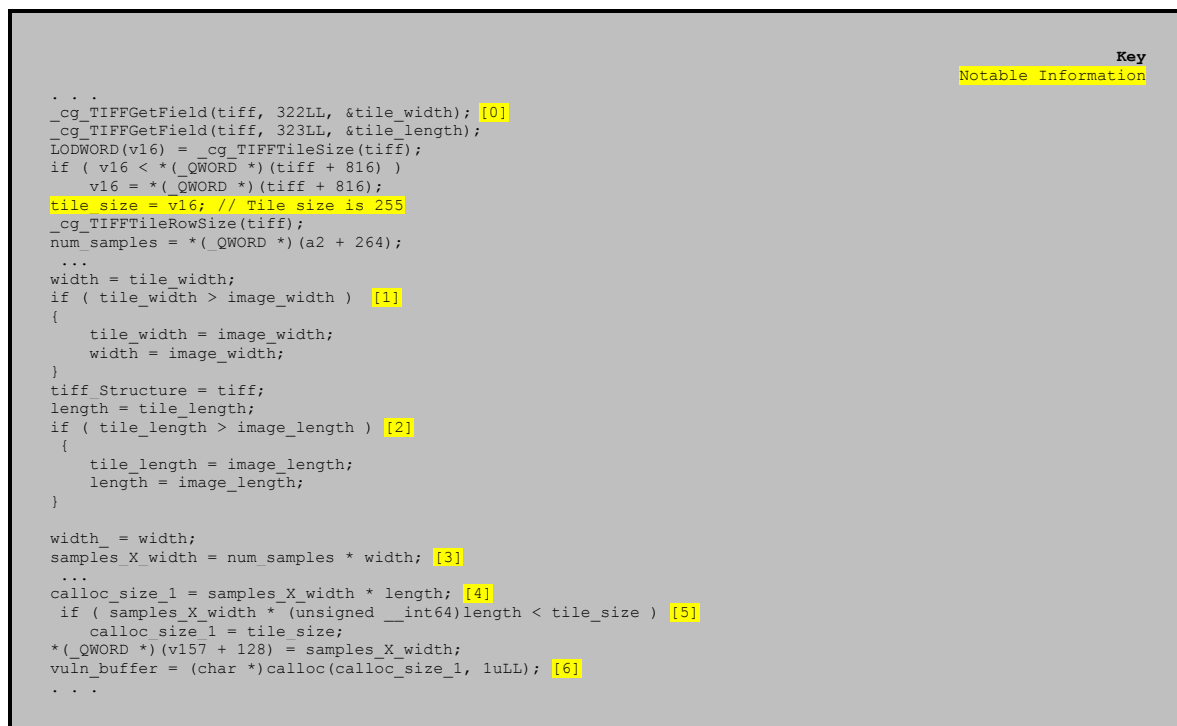
Essentially, this vmovups instruction is attempting to move a floating-point from %ymm0 and write it to the memory location specified at RAX.

Note these facts about the crash:

- RAX is pointing to the very end of a heap block 0x12d9cbfff where the error is received: EXC_BAD_ACCESS (code=1, address=0x12d9cc000) (Bohan, 2016).
- There is still data to be written. Counter register RDX contains 0xFE which is 254 in decimal (Bohan, 2016).
- The malloced buffer for RAX and the TIFF tile width are the same size of 255 (Bohan, 2016).

Decompiled Portion of the Apple Image I/O Function Where the Vulnerability Resides:

Fig. 2.1. Decompiled Code for ImageIO`copyImageBlockSetTiledTIFF: Near Where the Malloc Block is Allocated (Bohan, 2016)

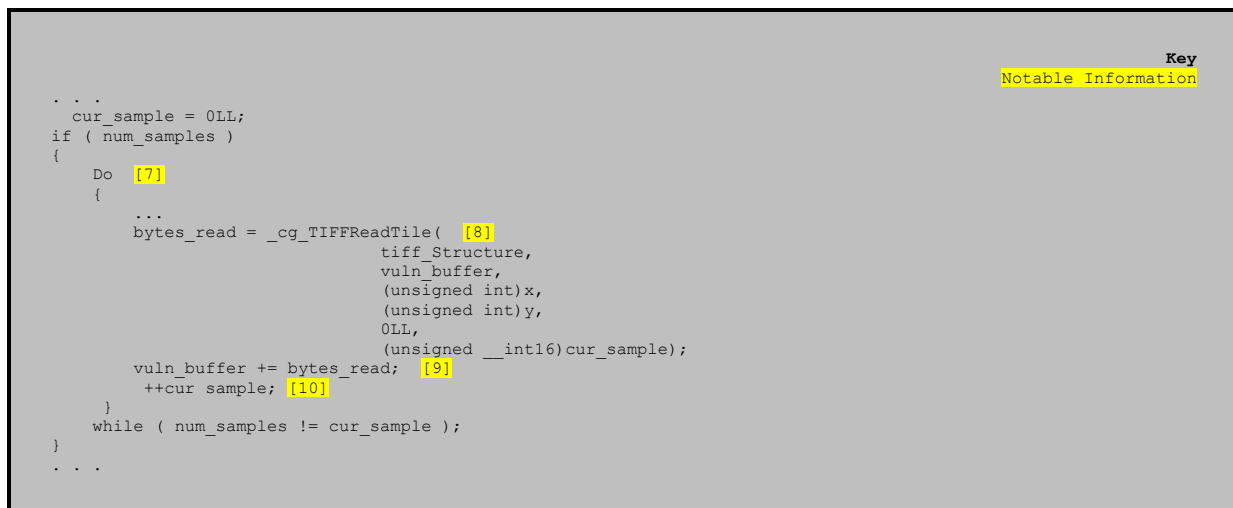


```
...
_cg TIFFGetField(tiff, 322LL, &tile_width); [0]
_cg TIFFGetField(tiff, 323LL, &tile_length);
LODWORD(v16) = _cg TIFFTileSize(tiff);
if ( v16 < *(_QWORD *) (tiff + 816) )
    v16 = *(_QWORD *) (tiff + 816);
tile_size = v16; // Tile size is 255
_cg TIFFTileRowSize(tiff);
num_samples = *(_QWORD *) (a2 + 264);
...
width = tile_width;
if ( tile_width > image_width ) [1]
{
    tile_width = image_width;
    width = image_width;
}
tiff Structure = tiff;
length = tile_length;
if ( tile_length > image_length ) [2]
{
    tile length = image length;
    length = image_length;
}

width_ = width;
samples_X_width = num_samples * width; [3]
...
calloc_size_1 = samples_X_width * length; [4]
if ( samples_X_width * (unsigned __int64)length < tile_size ) [5]
    calloc_size_1 = tile_size;
*(_QWORD *) (v157 + 128) = samples_X_width;
vuln_buffer = (char *)calloc(calloc_size_1, 1uLL); [6]
...
```

Key
Notable Information

Fig. 2.2. Decompiled Code for ImageIO`copyImageBlockSetTiledTIFF: Vulnerable Code (Bohan, 2016)



Breakdown of ImageIO`copyImageBlockSetTiledTIFF's Decompiled Code (Bohan, 2016):

Synopsis uses the malicious TIFF data from Attachment 1, Fig. 1.:

```
image_width = 2
image_length = 1
tile_width = 255
tile_length = 1024
num_samples = 4
```

[0] Information is read from the TIFF image to preform initial calculations.

Sets tile_size to 255

[1] Compare the tile width and the image width -- default to the smaller value.

[2] Compare the tile length and the image length -- default to the smaller value.

[3] samples_X_width is calculated to be used at point [4].

[4] calloc_size_1 is calculated using samples_X_width * length

[5] If (samples_X_width * length) < tile_size

a. Set calloc_size_1 = tile_size;

[6] calloc vuln_buffer size calloc_size_1

[7] do-while loop: num_samples != cur_sample

[8] Read in tile data from the TIFF image: one tile_size per sample = 255 bytes.

[9] **Vulnerable Buffer:** Moves the buffer forward size of one tile although the buffer is only allocated 255 bytes (points [5-6]).

Overflow Begins: Second iteration of loop at point [7] cur_sample > 0.

Loop Iterations Showing Memory Corruption:

cur_sample = 0 : writes to vuln_buffer[0-254]

cur_sample = 1 : writes out of bounds 255 bytes

cur_sample = 2 : writes out of bounds 510 bytes

cur_sample = 3 : writes out of bounds 765 bytes

cur_sample = 4 : writes out of bounds 1020 bytes

[10] Increment cur_sample and return to point [7]

References

- API Reference Framework Image I/O. (n.d.). Retrieved December 3, 2016, from <https://developer.apple.com/reference/imageio>
- Apple Mac OS X/watchOS/iOS/tvOS Multiple Security Vulnerabilities. (2016, July 18). Retrieved from <http://www.securityfocus.com/bid/91834>
- Apple Security Updates. (2016, November 2). Retrieved from <https://support.apple.com/en-us/HT201222>
- Bohan, T. (2016, July 18). Apple Image I/O API Tiled TIFF Remote Code Execution Vulnerability. Retrieved from <http://www.talosintelligence.com/reports/TALOS-2016-0171>
- Confidentiality, Integrity & Availability. (2009). Retrieved from <http://ishandbook.bsewall.com/risk/Methodology/CIA.html>
- Damme, J. V. (n.d.). TIFF Tag Reference, Baseline TIFF Tags. Retrieved December 3, 2016, from <http://www.awaresystems.be/imaging/tiff/tifftags/baseline.html>
- Davenport, T., & Vellon, M. (2008, November 24). Tag Image File Format Rev 4.0. Retrieved from <http://cool.conservation-us.org/bytopic/imaging/std/tiff4.html>
- Paganini, P. (2016, July 20). Hacking Apple devices with just a Message Exploiting the CVE-2016-4631. Retrieved from <http://securityaffairs.co/wordpress/49542/hacking/hacking-apple-cve-2016-4631.html>
- Rouse, M. (2007, May). Multimedia Messaging Service (MMS). Retrieved from <http://searchmobilecomputing.techtarget.com/definition/Multimedia-Messaging-Service>
- Rudolph K. (2014). Monitoring and Control Systems. In Bosworth S., Kabay M. E., & Whyne, E. (Eds.), *Customized Computer Security Handbook Select Chapters University of Nebraska-Omaha* (pp. 657). Hoboken, NJ: John Wiley & Sons.
- TIFF: Summary from the Encyclopedia of Graphics File Formats. (2013, February 17). Retrieved from <http://www.fileformat.info/format/tiff/egff.htm>
- Proffitt, B. (2013, September 19). What APIs Are And Why They're Important. Retrieved from <http://readwrite.com/2013/09/19/api-defined/>
- Vanderburg, E. (2012). Information Security Compliance: Which regulations relate to me? Retrieved from <http://jurinnov.com/information-security-compliance-which-regulations/>
- Vulnerability Summary for CVE-2016-4631*. (2016, November 28). Retrieved from <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-4631>