CVE-2022-42864 - Diabolical Cookies

January 19, 2023

iOS 16.2 and macOS Ventura 13.1 released just over a month ago, fixing an interesting vulnerability that I reported in IOHIDFamily. I would like to share the details of that vulnerability today.

The Bug

Apple's comment from the source code when this issue was fixed sums this up nicely:

```
// Find the number of cookies in the data. The data from eleme
```

Let us have a look at the function before the patch (I have tried to label relevant lines):

```
IOReturn IOHIDDevice::postElementTransaction(const void* eleme
{
    IOReturn ret = kIOReturnError;
    uint32_t   cookies_[kMaxLocalCookieArrayLength];
    uint32_t   *cookies = cookies_;
    uint32_t   cookieCount = 0;
    uint32_t   cookieSize = 0;
    uint32_t   dataOffset = 0;
    uint8_t    *data = (uint8_t*)elementData;
    IOMemoryDescriptor *elementDesc = getMemoryWithCurrentElem
    require(_elementArray && elementDesc, fail);

    WORKLOOP_LOCK;

    // Find the number of cookies in the data. Check that all
```

```
while (dataOffset < dataSize) {
    const IOHIDElementValueHeader *headerPtr = (const IOHI
    IOHIDElementPrivate *element = GetElement(headerPtr->c
    if (!element) {
        HIDDeviceLogError("Could not find element for cook
        ret = kIOReturnAborted;
        goto fail;
    }
    cookieCount++;

    require_noerr_action(os_add3_overflow(dataOffset, head
}
// Data isn't as large as expected, don't overrun, just ab
if (dataOffset != dataSize) {   //
    HIDDeviceLogError("Cookie data buffer is smaller than
                      (unsigned int)dataSize, (unsigned int)
    ret = kIOReturnAborted;
    goto fail;
}
dataOffset = 0;

require_noerr_action(os_mul_overflow(cookieCount, sizeof(u
                     fail,
                     HIDDeviceLogError("Overflow calculatir

cookies = (cookieCount <= kMaxLocalCookieArrayLength) ? co

if (cookies == NULL) {
    ret = kIOReturnNoMemory;
    goto fail;
}

// Update the elements, this replaced the shared kernel-us
for (size_t index = 0; dataOffset < dataSize; ++index) {
    const IOHIDElementValueHeader *headerPtr;
    IOHIDElementPrivate *element;
    OSData *elementVal;
```

```
        headerPtr = (const IOHIDElementValueHeader *)(data + d
        element = GetElement(headerPtr->cookie);
        dataOffset += headerPtr->length + sizeof(IOHIDElementV

        elementVal = OSData::withBytesNoCopy((void*)headerPtr-
                                             headerPtr->length)
        require_action(elementVal, fail, ret = kIOReturnNoMemo
        element->setDataBits(elementVal);
        elementVal->release();

        cookies[index] = headerPtr->cookie; //
    }

    // Actually post elements
    ret = postElementValues((IOHIDElementCookie *)cookies, (UI

fail:
    WORKLOOP_UNLOCK;
    if (cookies != &cookies_[0]) {
        IOFreeData(cookies, cookieSize);
    }

    return ret;
}
```

1. The loop at `[1]` counts the number of `IOHIDElementValue`s in the buffer, and stores this count in `cookieCount`.
2. The check at `[2]` (combined with the condition of the while loop) will make sure that the `length` field of each header does not extend out of the bounds of the `elementData` buffer (nor can it fall short of the end of the buffer, although this is less relevant).
3. Once all the elements have been counted and sanity-checked by this loop, a `cookies` buffer is allocated to the heap at `[3]` with a size of `cookieCount * 4` (or a stack buffer is used if `cookieCount` is

sufficiently small).

4. A second loop at `[4]` then makes a second pass through the buffer, parsing the `IOHIDElementValue`s again.
5. `OSData` objects are created to hold each element's value at `[5]`, using the `length` field that was validated in the first loop.
6. At `[6]`, each element's `cookie` is written into the `cookies` array allocated at `[3]`.

So what's the issue? This function behaves entirely correctly when `elementData` is non-volatile, the issue comes when the method is called with shared memory. Enter the `IOHIDInterface::SetElementValues_Impl` DriverKit method:

```
kern_return_t
IMPL(IOHIDInterface, SetElementValues)
{
    IOReturn ret = kIOReturnError;
    UInt8 *values = NULL;
    IOBufferMemoryDescriptor *md = NULL;

    md = OSDynamicCast(IOBufferMemoryDescriptor, elementValues
    require_action(md && count, exit, ret = kIOReturnBadArgume

    values = (UInt8 *)md->getBytesNoCopy();

    // Post the data to the device
    ret = _owner->postElementTransaction(values, (UInt32)md->g
    require_noerr_action(ret, exit, HIDServiceLogError("postEl

exit:
    return ret;
}
```

Here, `postElementTransaction` is called with `md->getBytesNoCopy()`,

memory shared with userspace, violating the assumption that `elementData` is non-volatile. The content of the `elementData` buffer can change after the loop at `[1]`, but before the loop at `[4]`, so what does this mean for an attacker?

There are two ways an attacker can abuse this:

- The first is to swap the `length` of a small `IOHIDElementValueHeader` at the end of the buffer to a much larger value. This means that when the `OSData` at `[5]` is created, it will extend far outside the bounds of the `elementData` buffer, allowing an attacker to read out-of-bounds data using `IOHIDInterface::GetElementValues_Impl`.
- The second is to swap the `length` of a large `IOHIDElementValueHeader` at the beginning of the buffer to a much smaller value. This will make the loop at `[4]` parse many more headers than were originally counted in the loop at `[1]`, so when cookies are written to the `cookies` array at `[6]`, they will overflow out of the array as `index` is never validated against `cookieCount`.

In practice, this allows an attacker to read out-of-bounds kernel heap data of an arbitrary size, and to write arbitrary data (again of an arbitrary size) out-of-bounds to the kernel heap. These are two powerful primitives.

Winning the race

With race conditions, we always look for ways to determine whether the race was won successfully, that way we can keep retrying until we succeed, making our trigger deterministic. Luckily in the case of this race condition, we can do exactly that.

For the OOB read variant, I place one more `IOHIDElementValueHeader` after the header that I'm switching the `length` of, with its `value` set to the recognisable constant of `0xD1AB011CAC1DF00D`. Then, when reading the

value of the element back, I know I have won the race if I see the familiar `0xD1AB011CAC1DF00D` header at the start of the returned data.

For the OOB write variant, I place one `IOHIDElementValueHeader` after the header that I'm switching the `length` of, but before the headers whose `cookie`s will be overflowed, this time with the recognisable `value` of `0xD15EA5ED`. This header will be encapsulated inside the `value` of the larger element in the case where we do not win the race, so the header will only be parsed and the element's value be set to `0xD15EA5ED` if we win the race. By reading back the element's value, I know whether I was successful.

Apple's fix

To fix the issue, Apple chose to add a third loop in between loop `[1]` and loop `[4]`, validating each `length` field, and then caching it in a new `dataLengths` array, while ensuring the number of elements had not changed. The final loop then uses the cached lengths for its calculations, avoiding reading from the buffer another time.

Issues with exploitation

The main obstacle to overcome when exploiting this issue is that the buffer we are overflowing out of belongs to `KHEAP_DATA_BUFFERS`, so exploitation targets are limited. In this proof-of-concept I chose to target kmsg headers, as these are one of very few structures in `KHEAP_DATA_BUFFERS` that contain kernel pointers. The "arbitrary kfree" primitive I obtained using this approach is the same primitive used in the multicast_bytecopy exploit, however the `IOSurfaceClient` array is now PAC'd and forged clients need to have a valid pointer back to the `IOSurfaceRootUserClient` that created them, rendering this no longer a desirable kernel r/w target.

Proof-of-concept

My proof-of-concept exploit for this issue, along with partial build instructions, is available on my GitHub: https://github.com/Muirey03/CVE-2022-42864

Thanks for Reading

Thank you for making it this far! If you have any questions about this bug, feel free to drop me a message on Twitter @Muirey03 :)