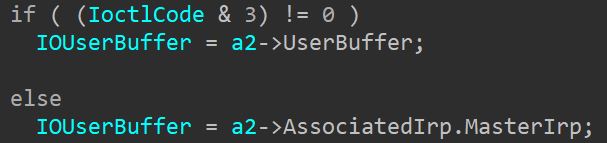
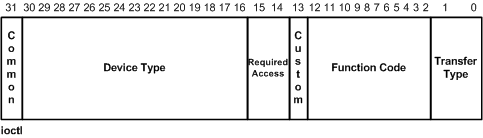
This driver is an ETW driver that’s also ready to get many IOCTL requests from users as it signs DeviceIoControl function and initializes a symbolic link to the driver.

The dispatch function makes sure that the I/O buffer is not NULL before continuing to one of the actual dispatch functions per IOCTL code. It also checks the following check before checking if the I/O buffer is NULL:

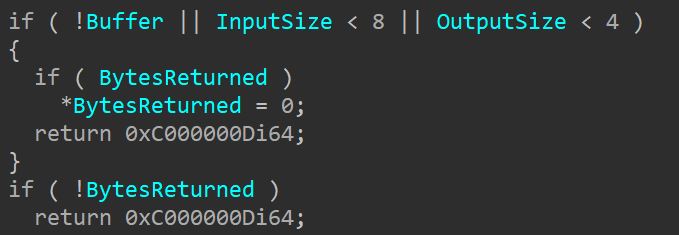


This area is just to take the right I/O buffer based on the passing method of the information that is specified in the last 2 bits of the 32 bit IOCTL code value

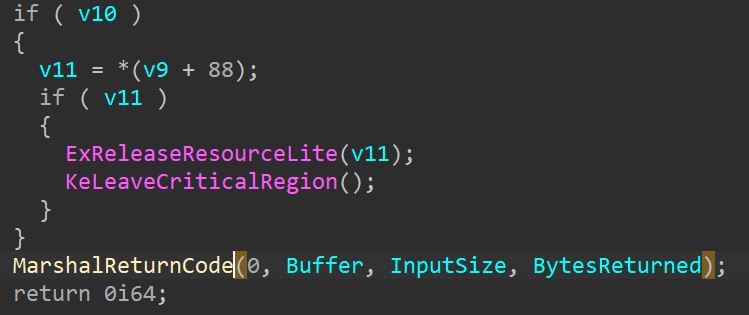


IOCTL 0xA689802C:

This operation expects the input buffer to not be NULL and BytesReturned to not be NULL, both cannot be NULL as previous checks are done before this function. It also wants the input size to be at least 8 and the output size to be at least 4.



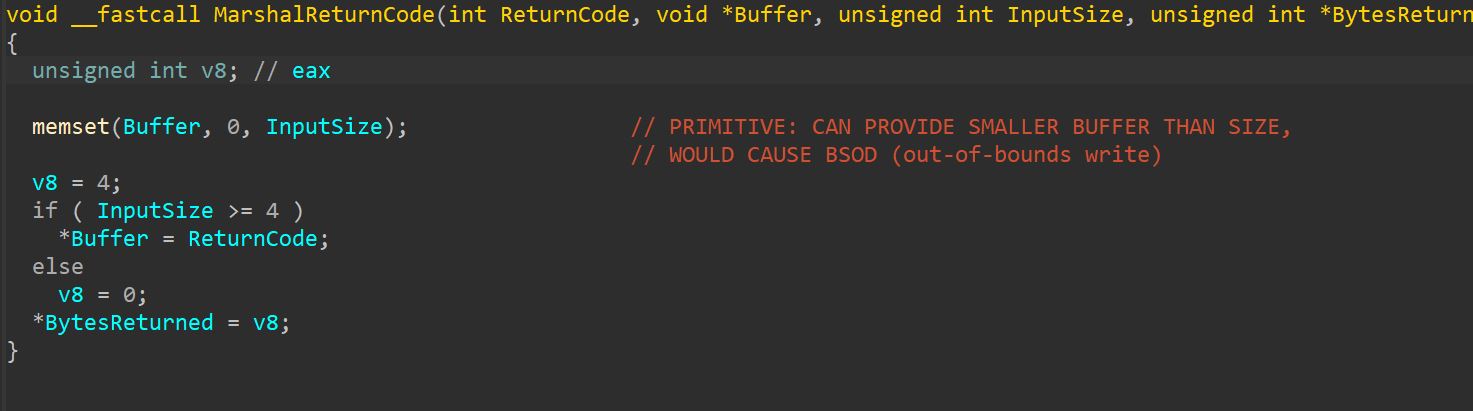
Then it also checks if the value in the first 4 bytes of the input are equal to the input size – which means that the first 4 bytes of input should be the total input buffer size. After this check some unclear operations are done on the information inside the device extension of the driver, and after that MarshalReturnCode is called with the following parameters:



Because I can control the actual content and content size of Buffer + interpreted size of buffer I can control everything provided to MarshalReturnCode. This function is also called at the end of every other IOCTL triggered function with the same caller-controlled parameters, so if there is any vulnerability inside this function It could be triggered by any IOCTL that would lead to the function’s execution

The function MarshalReturnCode:

This function is a short one and it looks like so:



This function zeroes out Buffer by the provided size of InputSize, returns the status code of the operation in the first 4 bytes of the buffer if the buffer size is >= 4 bytes and puts the BytesReturned value in the provided pointer (this is provided to the function as a pointer to a local variable, so I cannot make this address invalid somehow to fail the operation). Because this driver does not use ProbeForRead/Write to verify that the system allocated buffer for the provided parameters is actually the provided size, if I would provide a buffer the size of 4 bytes and would specify the size 8 as the buffer size, I could cause an out-of-bounds write that will DOS the whole system (in this case will cause a PAGE\_FAULT\_IN\_NON\_PAGED\_AREA BSoD)

Exploitation attempt:

For exploitation I need to make sure that 3 things happen:

1. System allocated buffer != NULL (should be possible in most cases if memory is available and if I provide an actual buffer)
2. In this IOCTL, InputSize should be >= 8 (which should work if I provide 8 as the input size but will actually send a buffer that is 4 bytes)
3. First 4 bytes of buffer should contain the value of InputSize (in this case -> 8, I will provide a ULONG variable that holds the value 8 as the actual buffer for this exploitation attempt)

This exploitation attempt failed because the dispatcher probably still allocates the system memory for the parameters by InputSize so i cannot cause a oob-write because i thought i could somehow provide a larger size

All other IOCTLs seemed to be based on the same structure that did not look like it would be vulnerable by the parameters I provide, so I stopped analyzing this driver