How to patch

1. Get and extract the linux source tarball
2. Copy my patch (named: patch-5.14.6-rc1811) in your system
3. `cd` into the extracted tarball
4. enter command: `patch -p1 < ../patch-5.14.6-rc1811` (or path/to/patchfile)
5. patch applied, now get your config and make

How it works

1. The patch modifies `arch/x86/entry/syscalls/syscall_64.tbl` and adds an entry to our syscall in that table. I have named it the same requested in the question (kernel_2d_memcpy). The syscall has number 548. As of linux-5.14.6 there were only 547 syscalls in that table.
2. The patch also modifies `kernel/sys.c` and adds the "definition" of the syscall there.

```
+SYSCALL_DEFINE3(kernel_2d_memcpy, long*, from_matr, int, cells, long*, to_usermat)
+{
+       int itr;
+       long temp[1];
+
+       if(access_ok(from_matr, cells)==0) {
+               return -EFAULT;
+       }
+       if(access_ok(to_usermat, cells)==0) {
+               return -EFAULT;
+       }
+
+       for(itr=0; itr<cells; itr++) {
+               if(__copy_from_user(temp, from_matr++, 8)!=0){
+                       return -EFAULT;
+               }
+               if(__copy_to_user(to_usermat++, temp, 8)!=0){
+                       return -EFAULT;
+               }
+       }
+       return 0;
+}
+
```

As seen above, the syscall expects 3 arguments, the first one being the source, second being the number of cells to copy, and the last being the destination.
Since a user cannot be trusted with ring 0 privileges, I have added `access_ok` to check if a user maliciously wants access (read/write) to some other process or the kernel itself. If access_ok returns 0, it implies that the user shouldn't be able to access that region, hence a pagefault (-EFAULT) is returned.
Iff all access checks pass, we start copying the data from the userspace to the kernel space in temp buffer, then write from that temp buffer in the supplied destination pointer. These operations require `__copy_from_user` and `__copy_to_user`. Both the functions error at non-zero value. Note that I have not copied the whole array into the kernel, this allows me to use less space and also copy LARGE matrices (kmalloc wouldn't allow large memory allocations).
Another important point to note is that I have not copied floating point numbers as floating points, the primary reason being that I was not obligated to do any floating point operations. They are instead copied as `long` because both `double` and `long` store 8 bytes of data and the kernel wouldn't really need to know if it was copying float, int, string, char, some_random_datatype, as long as it can copy the bytes.
That's it.

The testing file

1. Declares a floating point matrix of size 4x3
2. "Mallocates" a pointer to copy_to (the size remains `4*3*sizeof(double)`)
3. syscall 548
4. prints the exit status of the syscall
5. prints the `copy_to` matrix in one-dimension. (The source matrix is actually compressed as 1 dimensional using row major expansion which is again utilized in the syscall definition)
6. Q.E.D. ?