A Simple Inline Hook

The hook we're about to perform includes 3 parts:

1. Locating the beginning of the function to hook
2. Calculating the distance between the wanted function and our function (for the relative jump)
3. Replacing the first 5 bytes of the function with our 5 bytes of the jump

Let's start.

We are performing the demonstration with a very simple function – CloseHandle from Kernel32.dll. We chose this function because it takes only 1 parameter.

First we locate the address of the function CloseHandle:

```
hHandleToModule = GetModuleHandleW(pcModuleName);
if (NULL == hHandleToModule)
{
    printf("Error in function setHook: invalid parameter pcModuleName. exiting...");
    return RETURN_ERROR_INVALID_PARAM_MODULENAME;
}

iOldFunctionAddr = GetProcAddress(hHandleToModule, pcFunctionName);
if (NULL == iOldFunctionAddr)
{
    printf("Error in function setHook: Wasn't successful in getting the address of the given function. exiting...");
    return RETURN_ERROR_FAILED_GETTING_FUNCTION_ADDRESS;
}
```

- pcModuleName is the module of the function we want to hook, in our case – "Kernel32.dll"
- pcFunctionName is the name of the function to hook, in our case – "CloseHandle"

After performing this piece of code, the variable iOldFunctionAddr contains the address of CloseHandle function.

For the second step, we calculate the distance between the function to hook and the function we want to hook to:

```
diff = (INT_PTR)pNewFunction - ((INT_PTR)iOldFunctionAddr + 5);
```

- Note that we added the extra 5 bytes to the offset of the old function because the actual jump will take place from the next instruction after the JMP instruction, which is 5 bytes long from the beginning of the function.

Now we will replace the first 5 bytes with our own 5 bytes:

```
iProtection = VirtualProtect(iOldFunctionAddr, 5, PAGE_EXECUTE_READWRITE, &dwOldProtect);
if (0 == iProtection)
{
    printf("Error in function setHook: Wasn't successful in getting write access to given function. exiting...");
    return RETURN_ERROR_FAILED_IN_GETTING_ACCESS_TO_GIVEN_FUNCTION;
}

iOldFunctionAddr[0] = 0xE9;
memcpy(iOldFunctionAddr + 1, &diff, 4);
```

- Note that we had to use the VirtualProtect function in order to get permission to write in the CloseHandle function space.

And that's it! We finished setting the hook, nice and easy.

Don't forget to retrieve the previous permissions:

```
iProtection = VirtualProtect(iOldFunctionAddr, 5, dwOldProtect, &dwOldProtect2);
if (0 == iProtection)
{
    printf("Error in function setHook: Wasn't successful in getting the previous to given function. exiting...");
    return RETURN_ERROR_FAILED_IN_RETURNING_ACCESS_TO_GIVEN_FUNCTION;
}
```

And now we are truly done.

Let's check the results.

We call the function CloseHandle (that was hooked):

```
// Let's simulate the hook action:
hFakeHandle = (HANDLE) 0x1337;
result = CloseHandle(hFakeHandle);
```

And look at the output:

```
HAHA I'm not CloseHandle function and yet you called me!!
```

As expected :)

To sum up:

- We used a simple inline hooking for overriding the beginning of the CloseHandle function in order to make it jump to a function of our own.
- Note that when calling the function, the user run our code but not the real code as well. Obviously, in a real-life scenario we probably want to make the user think the legitimate code was run as well, so it won't be enough just to jump to our function, but we will also want to return to the actual function. More on that on the next time.
- Check the full c program here: https://github.com/Aluma010/Inline-Hooking/blob/master/A_Simple_Inline_Hook.c