API Hooking Using the Detour Library

In this section, we are going to learn about API Hooking Using the Detour Library. Intercepting the API functions.

Let's see what API Hooking is:

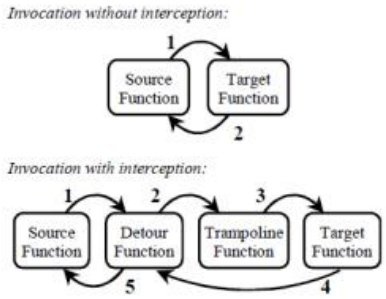
What is API hooking?

- To add functionality into a program where the source code is not available
- Make modifications at runtime without modifying the binary
- In game cheats, api hooking can intercept function which checks for player health and return 100% health
- Debuggers also use API hooking when you set breakpoints

What is Detours?

- A library for extending Win32 functions
- Intercepts Win32 functions by re-writing target function images
- Includes utilities to attach arbitrary DLLs and payloads to any Win32 binary

Calling function with and without interception



Source: Detours: Binary Interception of Win32 Functions, Galen Hunt and Doug Brubacher

Here consider the first case where the source function and the target function both are residing in the same process, then the source function is trying to call the target function, and the target function also returns the result of the execution to the source function.

But here in detours: From a Dll injected into the process.

We have a detour function, which will intercept a call from the Source function, and we are in control of the detour function.

So, now if we want we can forward the call, we can forward it to the target function, or jump to the payload, or do something else, or we can return something to the source function.

So, if we want to forward the call to the target process, we can pass the parameters from the source function through the trampoline function(Also a part of the detour library), which then calls the target process.

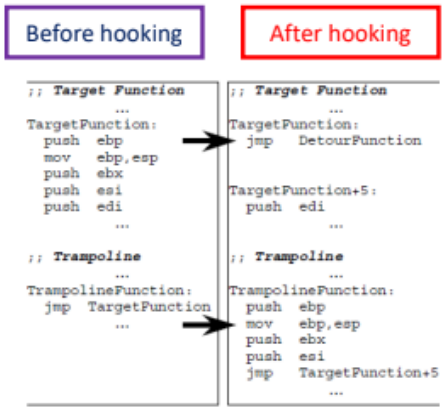
And if we can modify all the variables passed through the detour function, or we can just substitute it with a different function.

So, the function of the Trampoline function is to set the parameters that are passed from the source to the detour function, then forward it to the target process, and the trampoline function also contains additional things that came from the Source function.

After the target process executes the process, it gives the results to the Detour function, which then forwards it to the Source function.

So, from the point of view of the target process, it can't say whether it is dealing with the detour or the source function.

Target and Trampoline Functions, before and after hooking



So, now let's go through the code:

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#pragma comment(lib, "user32.lib")

int main(void){

    printf("Target For Hooker is Starting...\n");

    //-- ref:  https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-messagebox
    MessageBox(NULL, "Hello I am First message", "1st MessageBox", MB_OK | MB_ICONINFORMATION);
    MessageBox(NULL, "Hello I am Second message", "2nd MessageBox", MB_OK | MB_ICONINFORMATION);
    MessageBox(NULL, "Hello I am Third message", "3rd MessageBox", MB_OK | MB_ICONINFORMATION);

    printf("Target For Hooker exiting...\n");

    return 0;
}
```

It is a simple program that shows three simple message boxes.

So, we are going to intercept this message box using detours.

So, for that, we need another file: hooker.cpp

```c
#include <stdio.h>
#include <windows.h>
#include "detours.h"
#pragma comment(lib, "user32.lib")

//-- pointer to the original MessageBox function
int (WINAPI * pOrigMessageBox)(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) = MessageBox;

//-- hook and unhook function prototypes
BOOL HookTarget(void);
BOOL UnHookTarget(void);

//-- the modified MessageBox function
int ModifiedMessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) {

    printf("ModifiedMessageBox() called. No MessageBox popup on screen!\n");

    //pOrigMessageBox(hWnd, lpText, lpCaption, uType);

    //if(strcmp(lpCaption, "2nd MessageBox")==0)
    //  pOrigMessageBox(hWnd, "I have been modified", lpCaption, MB_OK | MB_ICONERROR);

    return IDOK;
}

//-- Set hooks on the MessageBox function
BOOL HookTarget(void) {

    LONG hookingResult;

    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());
    DetourAttach(&(PVOID&)pOrigMessageBox, ModifiedMessageBox);
    hookingResult = DetourTransactionCommit();

    printf("MessageBox() hooked and hooking ressult = %d\n", hookingResult);

    return TRUE;
}
```

```
41  //-- Undo the hooks and revert all changes to original code
42  BOOL UnHookTarget(void) {
43
44      LONG unhookingResult;
45
46      DetourTransactionBegin();
47      DetourUpdateThread(GetCurrentThread());
48      DetourDetach(&(PVOID&)pOrigMessageBox, ModifiedMessageBox);
49      unhookingResult = DetourTransactionCommit();
50
51      printf("Hook removed from MessageBox() with un-hooking result = %d\n", unhookingResult);
52
53      return TRUE;
54  }
55
56  BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved) {
57
58      switch (dwReason)  {
59          case DLL_PROCESS_ATTACH:
60              HookTarget();
61              break;
62
63          case DLL_THREAD_ATTACH:
64              break;
65
66          case DLL_THREAD_DETACH:
67              break;
68
69          case DLL_PROCESS_DETACH:
70              UnHookTarget();
71              break;
72      }
73
74      return TRUE;
75  }
```

So, hooker .cpp is a dll which we are going to build.

And when it going to attach to the process.

So, here we can see in the headers, that we have included the detours library.

Now HookTarget is the function that should be called when it attaches the target for the hooker.

In HookTarget we can see the message being used because we are looking for the message box. So, here we have called a few functions which will perform the hooking:

Here we have used a few functions:

DeteourTransactionBegin: It will just initialize it, by calling this function.

DetourUpdateThread: Then we get the GetCurrentThread, by calling this function. The current thread will be the thread the hooker is attached to, in this case, targetofhooker.

DetourAttach: Here it will intercept the function called pOrigMessageBox, so here we specify one function to intercept it, and in this case, we are going to intercept the message box, this is the first parameter, then the second parameter is the modified API we are going to use to replace the original API.

The message box API looks like this:

```
C++

int MessageBox(
  [in, optional] HWND    hWnd,
  [in, optional] LPCTSTR lpText,
  [in, optional] LPCTSTR lpCaption,
  [in]           UINT    uType
);
```

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

DetourTransctionCommit: Now it will execute all the commands specified up, so it will commit a result, and then store it in a variable “hookingResult”.

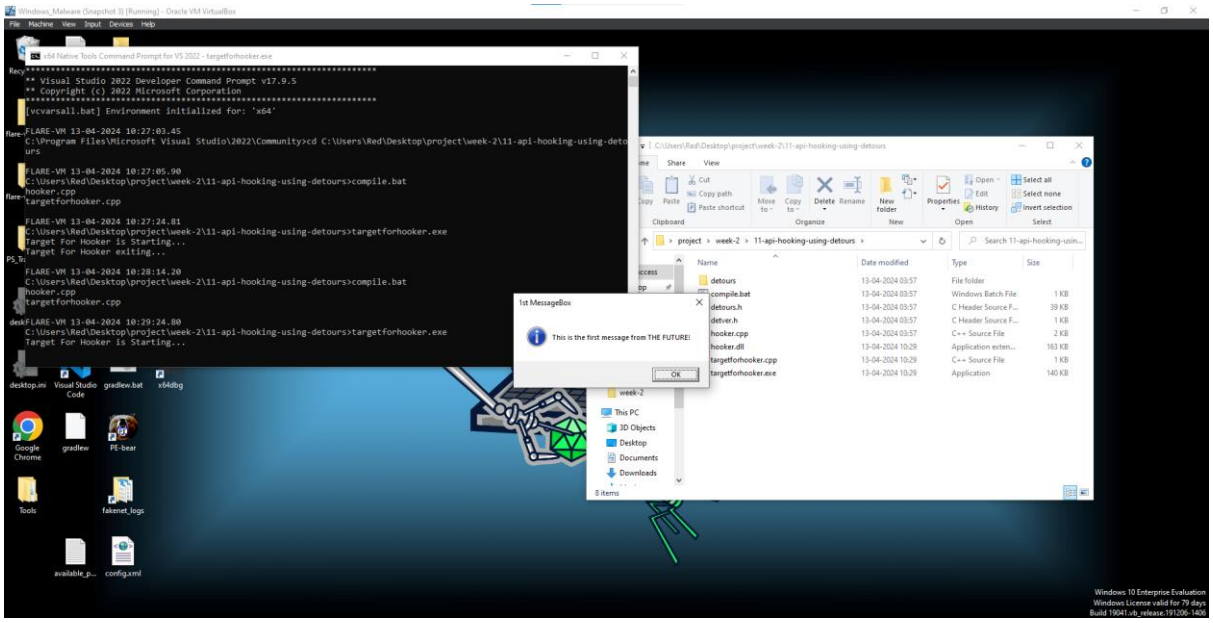
And when the detours detach from the target, then it will call the UnHookTarget function.

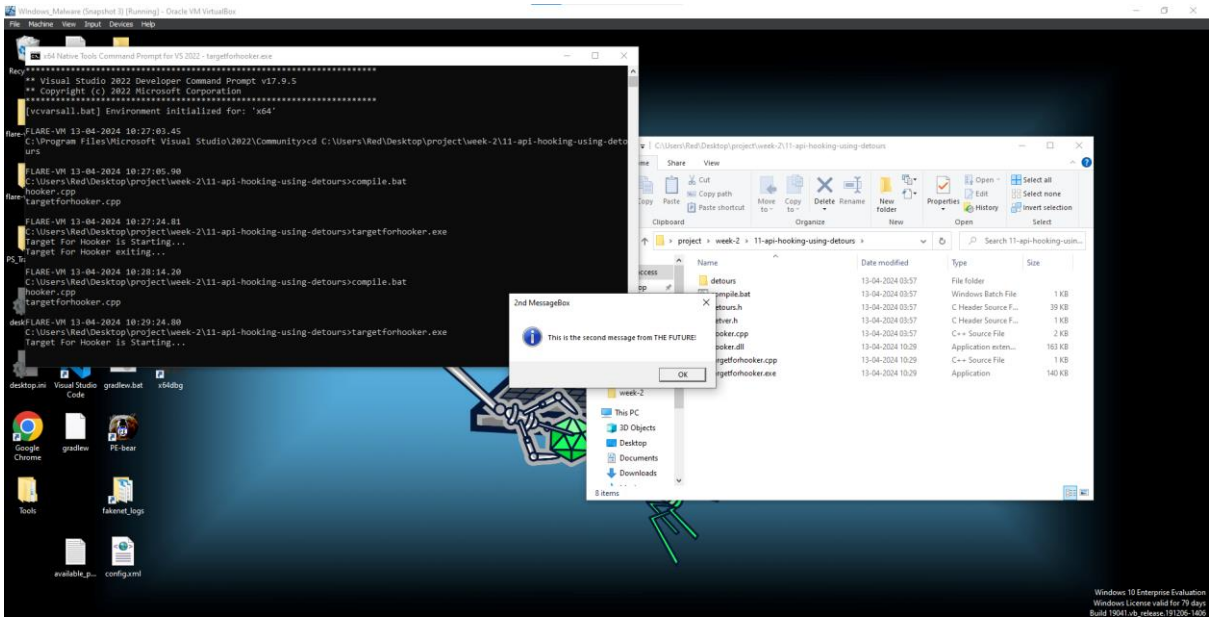
It will undo the hook and revert all the changes to the original code.

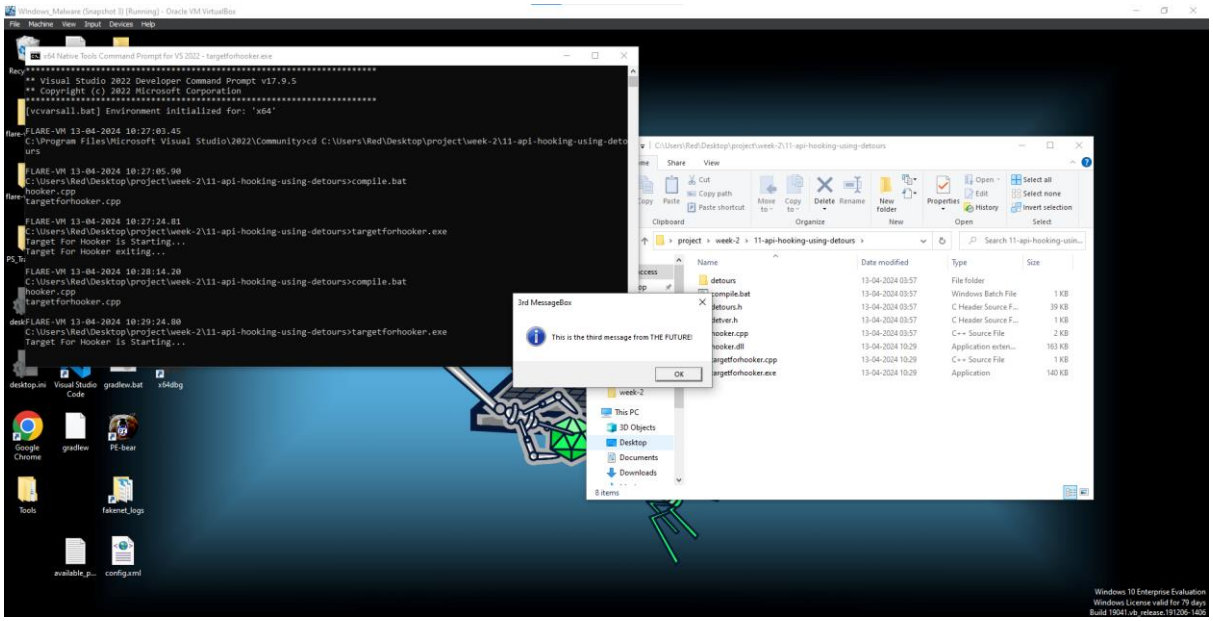
Once again there is the same set of functions, but instead of DetourAttach, we use the DetourDetach function.

Now, let’s compile the files:

Here we can see that, we got a first pop-up:


1st MessageBox
This is the first message from THE FUTURE!
OK

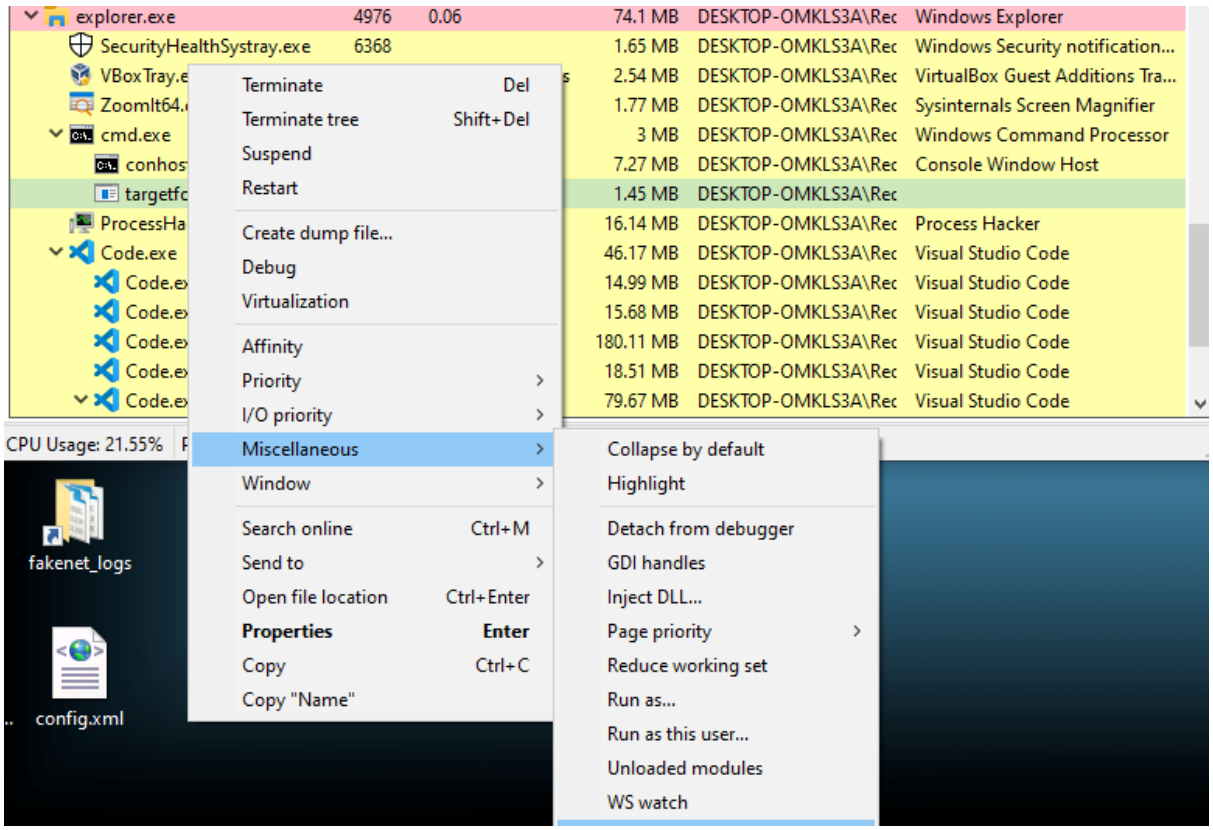

2nd MessageBox
This is the second message from THE FUTURE!
OK



So, if we press “ok”, we will get two more pop-ups after that it will exit.

So, now let’s inject the dll in it:

We will be doing it by Process Hacker:



Select the Inject Dll option, and then press ok on the 1st message box, then we can see that there is no 2nd message pop-up, and it exits the program

```
C:\Users\Red\Desktop\project\week-2\11-api-hooking-using-detours>targetforhooker.exe
Target For Hooker is Starting...
MessageBox() hooked and hooking ressult = 0
ModifiedMessageBox() called. No MessageBox popup on screen!
ModifiedMessageBox() called. No MessageBox popup on screen!
Target For Hooker exiting...
Hook removed from MessageBox() with un-hooking result = 0
```

So, what is happening is that when the DetourAttach function has intercepted the original message, and then replaced it with the ModifiedMessageBox.

When the ModifiedMessageBox function is called, then it will print the above statement in the command prompt.

```
14  int ModifiedMessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) {
15
16      printf("ModifiedMessageBox() called. No MessageBox popup on screen!\n");
```

Then it just returns the IDOR, and then leaves.

And now once again when the 3rd message box is called, it does the same thing, shows the same message, and doesn't allow the message to get popped up, because it has changed the OriginalMessageBox.

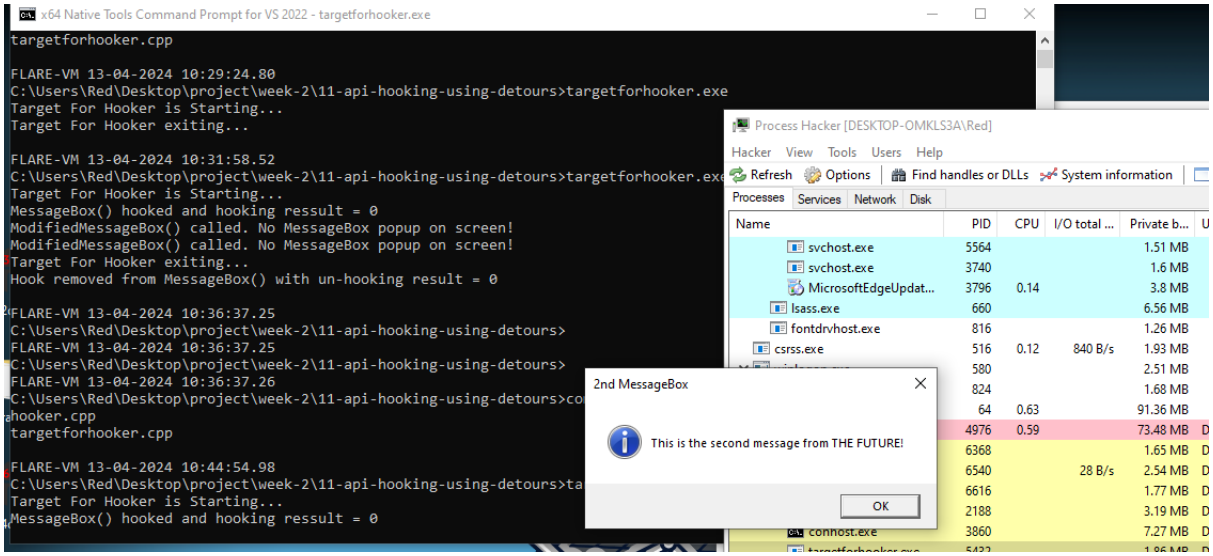
Now, we have made a few changes in the code:

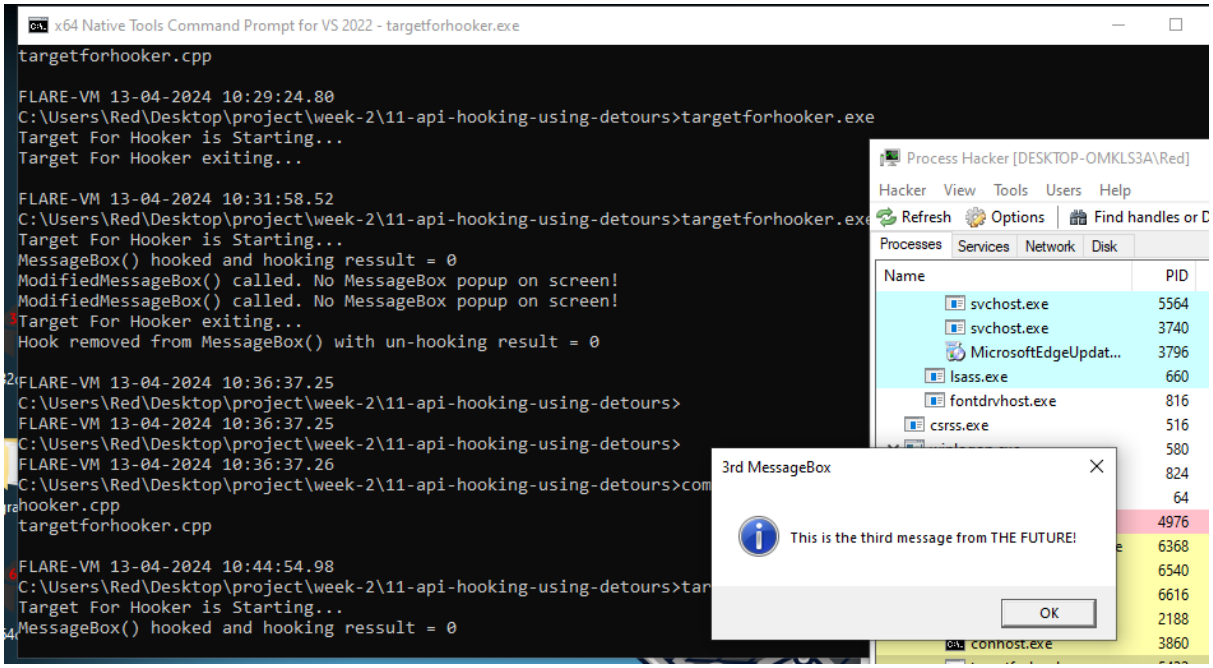
```
14  int ModifiedMessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) {
15
16      //printf("ModifiedMessageBox() called. No MessageBox popup on screen!\n");
17
18      pOrigMessageBox(hWnd, lpText, lpCaption, uType);
```

So, we are going to forward everything and allow the message box to pop up.

And so, compile the file, and once again run it:

Once again inject the dll, and then press "ok" on the message, we can see that we get the rest of the pop-ups.





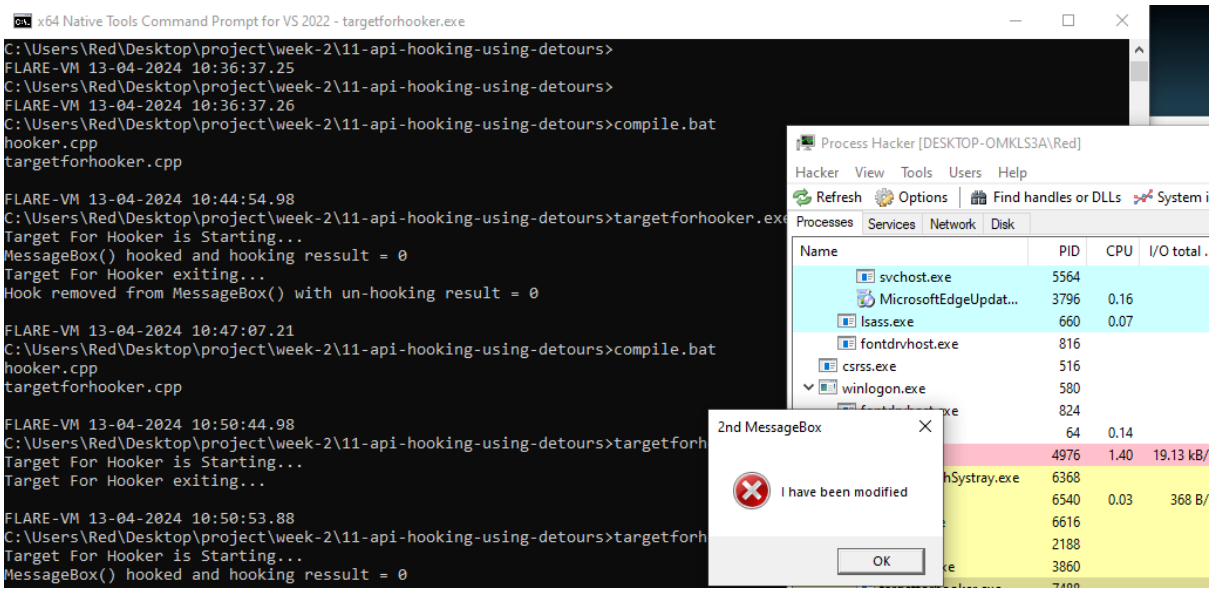
So, now what if we want to intercept only the 2nd message box:

Modify the code to look like this:

```
14  int ModifiedMessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) {
15
16      //printf("ModifiedMessageBox() called. No MessageBox popup on screen!\n");
17
18      //pOrigMessageBox(hWnd, lpText, lpCaption, uType);
19
20      if(strcmp(lpCaption, "2nd MessageBox")==0)
21          pOrigMessageBox(hWnd, "I have been modified", lpCaption, MB_OK | MB_ICONERROR);
22
23      return IDOK;
24  }
```

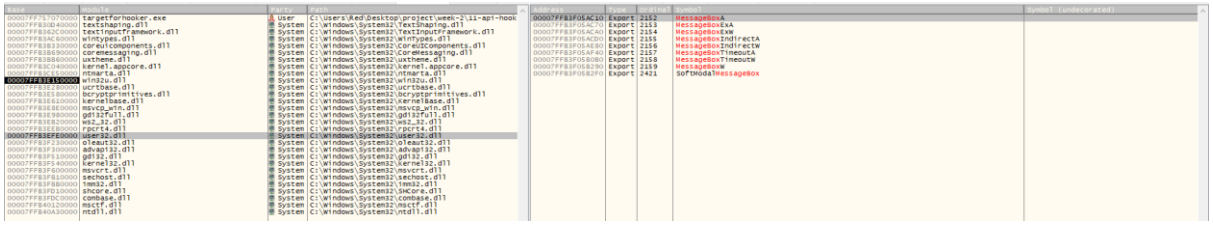
We can see in the code that it will show the error symbol, rather than the information symbol, so basically we are going to change the message.

Now we will compile the file, make sure that once we run the .exe file, you must inject the .dll file.



Here we can see that the message box got changed, and when we pressed ok, it didn't even pop up the 3rd message and just exited the program.

So, now let's use the xdbg, to reverse engineer the DETOURS:



Attach the process, then under the symbol tab, search for user32.dll, because it contains the message box.

Then follow it in the disassembler, then put a breakpoint at this point.

Then inject the .dll file, then choose the Analysis option in the disassembly.

And now, let's the dll, and see what happens:

We can see that it gave us an updated popup.

So, in the xdbg, the instruction got changed to jump, form sub, and it got us a new popup.