



```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX                                     XXXX                                  XX
XX   /  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  XX
XX  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  XX
XX /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX   /  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  XX
XX  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  XX
XX /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XX                                     XXXX                                  XX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

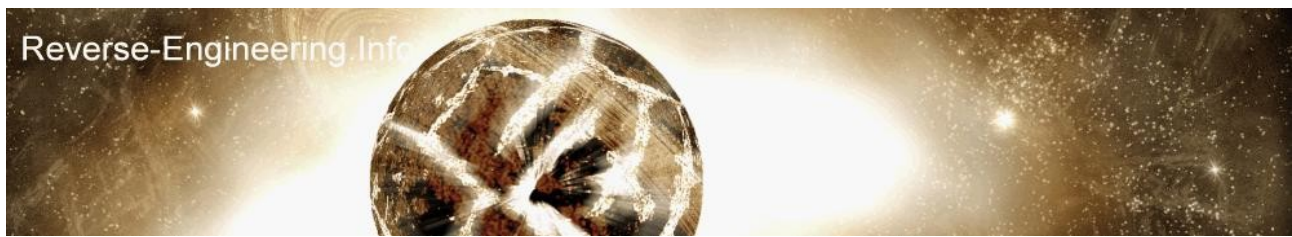
Web: <http://www.ImmortalDescendants.org>
 Author: [yAtEs]
 Date: 26/Sept/00
 Topic: Hooking APIs (VxD)
 (Part III of III)
 Level: Advanced

This is part III of my VxD and the last tutorial I'll do on VxDs to my knowledge. We're going to look at API hooking, API hooking can be fun as we can tell an API to do another function after/before it has been executed, example save tcp/ip packets after it receives them with the winsock!recv API etc.

So how do we hook an API? the basically what happens is we code our extra routine somewhere, then we modify the DLL in memory by placing a push to our routine then a RET so that when the API is called it jumps to our code, we can then restore any bits of code we have wiped over then continue with the API code.

The basis of my tutorial is to show you how to modify the DLL memory and insert and restore your own code, so my example with be nothing amazing nor useful. I am going to make an INT 3 occur after every MessageBoxA... I know your excited now :)

Ok when we want to hook an API, we must take a look at the APIs code and plan how we are going to make our modifications, so BPX on MessageBoxA and instead of pressing F12 lets take a peek at the code :)

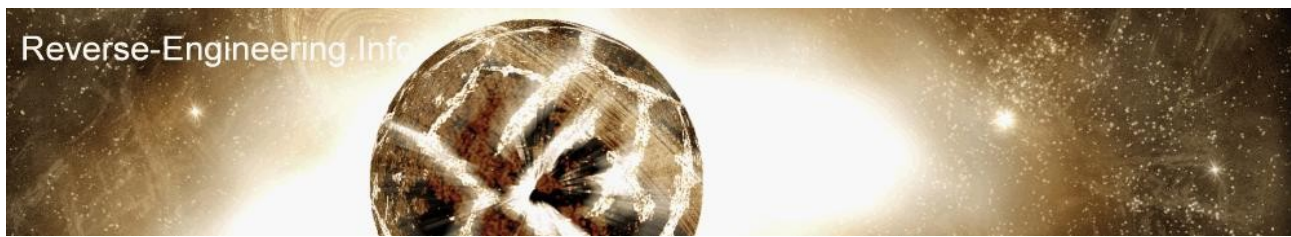


```
USER32!MessageBoxA
0177:BFF5412E  55          PUSH     EBP
0177:BFF5412F  8BEC       MOV     EBP,ESP
0177:BFF54131  6A00       PUSH     00
0177:BFF54133  FF7514    PUSH     DWORD PTR [EBP+14]
0177:BFF54136  FF7510    PUSH     DWORD PTR [EBP+10]
0177:BFF54139  FF750C    PUSH     DWORD PTR [EBP+0C]
0177:BFF5413C  FF7508    PUSH     DWORD PTR [EBP+08]
0177:BFF5413F  E8D8ECFFF CALL     USER32!MessageBoxExA
0177:BFF54144  5D        POP     EBP
0177:BFF54145  C21000    RET     0010
```

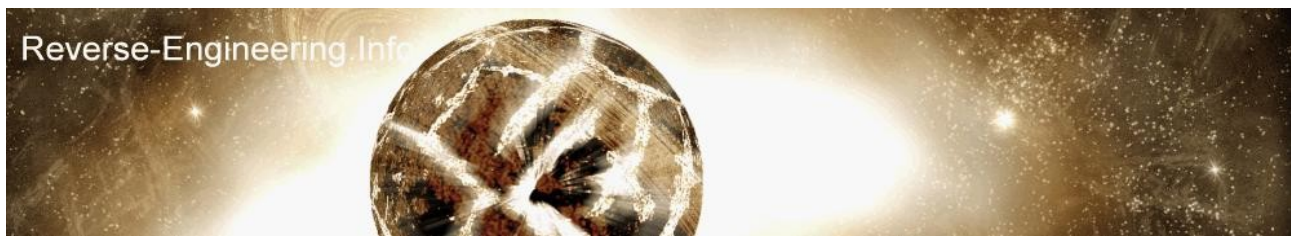
after studying the code i decided to do the following

```
USER32!MessageBoxA
0177:BFF5412E  55          PUSH     EBP
0177:BFF5412F  8BEC       MOV     EBP,ESP
0177:BFF54131  6A00       PUSH     00
0177:BFF54133  FF7514    PUSH     DWORD PTR [EBP+14]
0177:BFF54136  FF7510    PUSH     DWORD PTR [EBP+10]
0177:BFF54139  FF750C    PUSH     DWORD PTR [EBP+0C]
0177:BFF5413C  FF7508    PUSH     DWORD PTR [EBP+08]
0177:BFF5413F  E8D8ECFFF CALL     USER32!MessageBoxExA
0177:BFF54144  xxxxxxxxxx PUSH <our_routine>
0177:BFF54145  C3        RET
0177:BFF54146  xxxxxxxx  RET     0010
```

Now we've decided how to implant our code we should start to code the VxD, to active the installation of our API hook we will send a control message to our VxD with the address of the API we are hooking. let me paste my new loader code



```
;  
_____  
.486P  
locals  
jumps  
.Model Flat ,StdCall  
  
Extrn  MessageBoxA:PROC  
Extrn  exitprocess:PROC  
Extrn  CreateFileA:PROC  
Extrn  CloseHandle:PROC  
Extrn  GetModuleHandleA:PROC  
Extrn  GetProcAddress:PROC  
Extrn  DeviceIoControl:PROC  
  
.data  
  
file1 db "\\.\first.vxd",0  
  
fbox db  'Loader',0  
ftitle db 'you broke it',0  
ftitle2 db 'Loaded',0  
User   db 'User32.dll',0  
BytesReturned dd ?  
handle1 dd ?  
  
MSGb db "MessageBoxA",0  
UHandle dd 0  
  
DIOC_MSGb equ 5  
  
.code  
  
main:  
  
    Call CreateFileA,offset file1,0,0,0,0,4000000h,0  
    cmp eax,-1  
    je fuxor  
    mov handle1,eax  
  
    Call  GetModuleHandleA,offset User  
    mov   UHandle,eax  
  
    Call  GetProcAddress,UHandle,Offset MSGb ; get api address and return it to eax  
    Call  DeviceIoControl,handle1,DIOC_MSGb,eax,0,0,0,offset BytesReturned,0  
  
    Call MessageBoxA,0,offset ftitle2,offset fbox,0  
    jmp endprog  
  
fuxor:  
    Call MessageBoxA,0,offset ftitle,offset fbox,0  
  
endprog:  
    Call CloseHandle,handle1  
    call exitprocess,0  
  
end main  
;  
_____
```



That is our new loader code, you can see we obtain the API address and then send to the VxD where it will remain in lpvInBuffer, the control message we send is also branded an ID, this is DIOC_MSGb in this case.

now in our VxD we must add the code to handle the deviceio message, here is how it looks

```
BeginProc OnDeviceIoControl
    assume esi:ptr DIOCParams
    .if [esi].dwIoControlCode==DIOC_Open
        xor eax,eax
    .ELSEIF [esi].dwIoControlCode == DIOC_MSGb
        mov esi,[esi].lpvInBuffer
        mov dword ptr [orgMSG],esi
        add esi,1Ch
        mov dword ptr [newMSG],esi
        call    MSG_Install
    .endif
    ret
EndProc OnDeviceIoControl
```

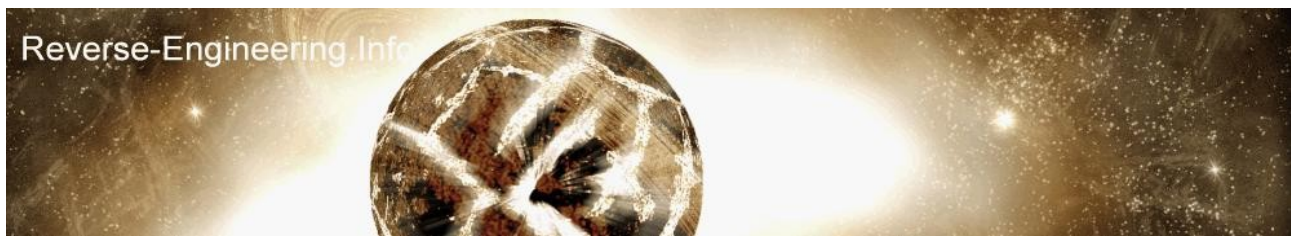
we also must have a data section

```
;
VxD_LOCKED_DATA_SEG
;

DIOC_MSGb equ 5
orgMSG dd 0
newMSG dd 0
;
VxD_LOCKED_DATA_ENDS
;
```

ok now if we look at the new control procedure we can see that after it detects it has received our DIOC_MSGb it extracts the API address from the buffer and stores it in orgMSG.

We add 1Ch to the API pointer for a return address, after we execute our new function, the one that we are going to insert into the API(int 3) we need to jump to the instruction back in the main API routine, hm if that doesn't make sense look at this.



```

USER32!MessageBoxA
0177:BFF5412E  55          PUSH     EBP          <=orgMSG[BFF5412E]
0177:BFF5412F  8BEC       MOV     EBP,ESP
0177:BFF54131  6A00       PUSH     00
0177:BFF54133  FF7514     PUSH     DWORD PTR [EBP+14]
0177:BFF54136  FF7510     PUSH     DWORD PTR [EBP+10]
0177:BFF54139  FF750C     PUSH     DWORD PTR [EBP+0C]
0177:BFF5413C  FF7508     PUSH     DWORD PTR [EBP+08]
0177:BFF5413F  E8D8ECFFF CALL     USER32!MessageBoxExA
0177:BFF54144  xxxxxxxxx PUSH     <our_routine>
0177:BFF54145  C3        RET
0177:BFF54146  xxxxxx    RET     0010         <=newMSG[BFF54146]

```

after our routine is called we jump to [newMSG]

back to the device control procedure, the last bit calls our install procedure MSG_Install.

Now let me show you my install procedure then we can evaluate it

```

;
MSG_Install      Proc

                cmp     [orgMSG],0
                jz     @lskipinstall
                mov    esi,[orgMSG]

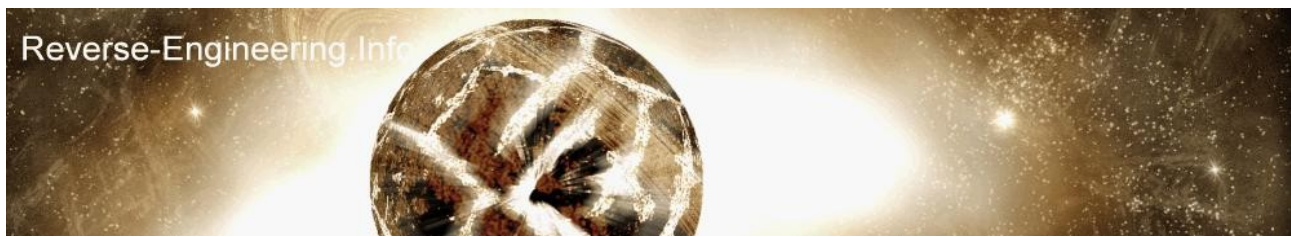
                add   esi,16h
                mov   byte ptr [esi],68h
                inc   esi
                mov   dword ptr [esi],offset32 MSG_Hook
                mov   byte ptr [esi+4],0C3h
                mov   byte ptr [esi+5],0C2h
                mov   byte ptr [esi+6],10h
                mov   byte ptr [esi+7],00h

@lskipinstall:
                ret
MSG_Install      endp
;

```

does that look rather confusing? heh nah thought not, you leetoe ;)

we move the address of messageboxA(contained in orgMSG) into esi we check this for 0, because if it is 0 then something went wrong so we as well skip the install than crash ourselves. providing the address is ok, we increase it by 16h, this takes us to the point just after 'USER32!MessageBoxExA' line, now we start coding in hex :) 68h for push then increase by 1 and put the address to our new proc then increase by 4 because the instruction is 4 byte obviously ;)



now we add the RETs anyway after all this our code would look like the following:-

```
USER32!MessageBoxA
0177:BFF5412E  55          PUSH     EBP
0177:BFF5412F  8BEC       MOV     EBP,ESP
0177:BFF54131  6A00       PUSH     00
0177:BFF54133  FF7514     PUSH     DWORD PTR [EBP+14]
0177:BFF54136  FF7510     PUSH     DWORD PTR [EBP+10]
0177:BFF54139  FF750C     PUSH     DWORD PTR [EBP+0C]
0177:BFF5413C  FF7508     PUSH     DWORD PTR [EBP+08]
0177:BFF5413F  E8D8ECFFF CALL     USER32!MessageBoxExA
0177:BFF54144  689DC2E6CE PUSH     CEE6C29D
0177:BFF54149  C3        RET
0177:BFF5414A  C21000    RET     0010
```

there you can see our 68 we added followed by the address and our rets.

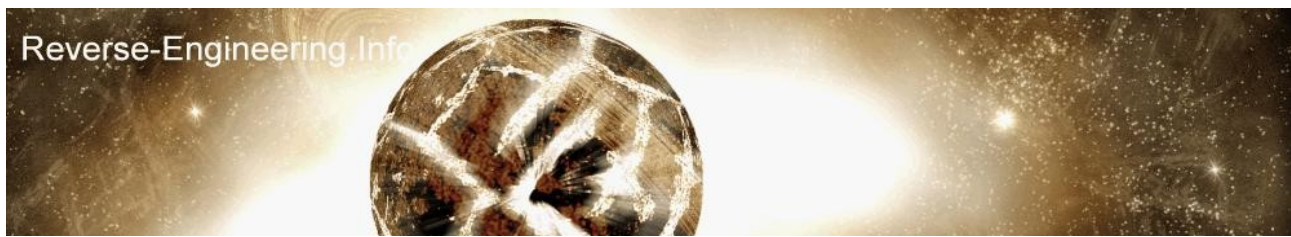
simple huh, right ok,lets view the MSG_Hook routine

```
;  
BeginProc MSG_Hook  
pushfd  
pushad  
  
int 3  
  
popad  
popfd  
pop ebp  
jmp dword ptr [newMSG]  
EndProc MSG_Hook  
;
```

hehe ok this is what happens after a call to messageboxa is made, we issue an int 3 making sure we save and restore registers before/after,

then there is a POP EBP this is very important, if you look back at the original API code we wipped over this with our code, so we must pop whatever value is on the stack to ebp or else our ret will return to some weird location,..and finally we jump to newMSG which you should remember is the RET 0010, if you haven't a clue what i'm talking about, scroll up and reread what i wrote about it, and lay off the wodka ;)

hmmm..hmm.hmmm..ah ha! ok now we must add an uninstall procedure add this to our control message handle, not our control device procedure.



```
;  
-----  
Control_Dispatch Sys_Dynamic_Device_Exit,OnDeviceDestroy
```

```
;  
-----
```

now the destroy proc is as follows

```
;  
-----  
BeginProc OnDeviceDestroy  
        call    MSG_Uninstall  
        cld  
        ret  
EndProc OnDeviceDestroy
```

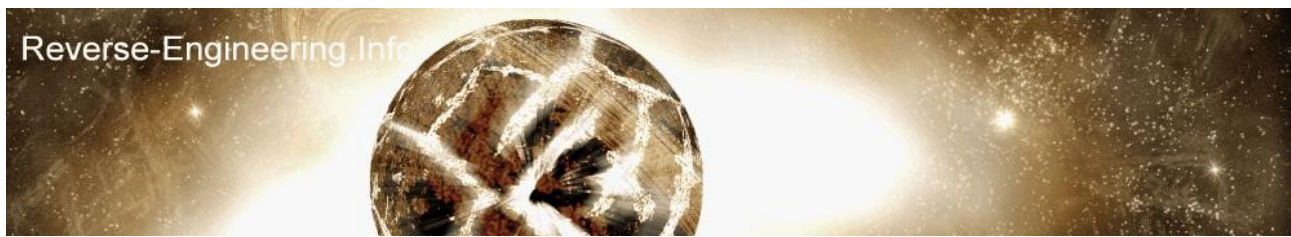
```
;  
-----
```

blah blah simple enough, now the MSG_Uninstall

```
;  
-----  
MSG_Uninstall Proc  
  
        mov     esi,[orgMSG]  
        add     esi,16h  
        mov     byte ptr [esi],5Dh  
        mov     byte ptr [esi+1],0C2h  
        mov     byte ptr [esi+2],10h  
        mov     byte ptr [esi+3],00h  
  
@lskipuninstall:  
        ret  
MSG_Uninstall endp  
;  
-----
```

I bet you could probably figure this out, it gets the API address and rewrites the original bytes back.

and that's pretty much it, I'll paste full source now.



```
.ELSEIF [esi].dwIoControlCode == DIOC_MSGb
    mov esi,[esi].lpvInBuffer
    mov dword ptr [orgMSG],esi
    add esi,1Ch
    mov dword ptr [newMSG],esi
    call    MSG_Install
.endif
ret
EndProc OnDeviceIoControl

MSG_Install          Proc

                        cmp     [orgMSG],0
                        jz     @lskipinstall
                        mov    esi,[orgMSG]

                        add esi,16h
                        mov byte ptr [esi],68h
                        inc esi
                        mov dword ptr [esi],offset32 MSG_Hook
                        mov byte ptr [esi+4],0C3h
                        mov byte ptr [esi+5],0C2h
                        mov byte ptr [esi+6],10h
                        mov byte ptr [esi+7],00h

@lskipinstall:
                        ret
MSG_Install          endp

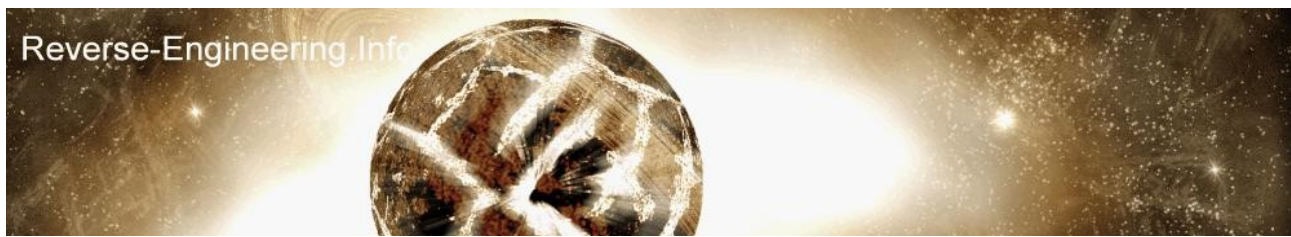
BeginProc OnDeviceDestroy
                        call    MSG_Uninstall
                        cld
                        ret
EndProc OnDeviceDestroy

BeginProc MSG_Hook
pushfd
pushad

int 3

popad
popfd
pop ebp
jmp dword ptr [newMSG]

EndProc MSG_Hook
```



```
MSG_Uninstall Proc

;                               cmp     [orgMSG],0
;                               jz      @lskipuninstall
int 3

                               mov     esi,[orgMSG]
                               add     esi,16h
                               mov     byte ptr [esi],5Dh
                               mov     byte ptr [esi+1],0C2h
                               mov     byte ptr [esi+2],10h
                               mov     byte ptr [esi+3],00h

@lskipuninstall:
                               ret

MSG_Uninstall  endp
;
VxD_LOCKED_CODE_ENDS
;

end
;
```

TADA!!!, that was easier to write than i thought, i bet its harder to understand thou ;d

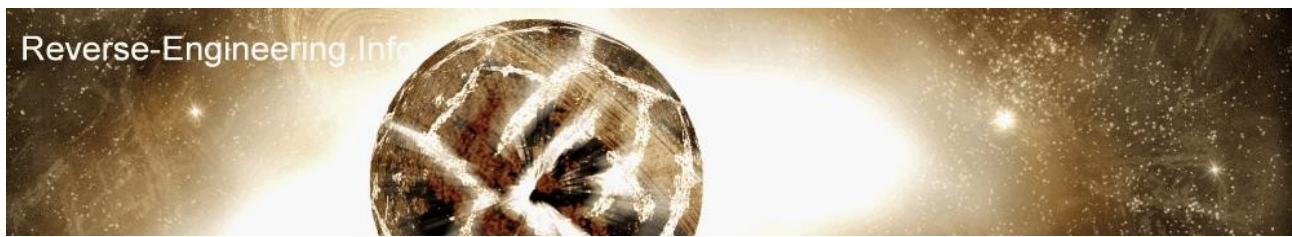
anyway give it a go, you really have to plan how you insert your new code etc, lots of int 3s to debug your code i.e. checking you have your code in the right place.

you can download my working version from the url provided at the top of the document.

If you clear all break points and set i3here on, now load the vxd, after any msg box you will get an int 3 break, also when you click ok to shutdown the msgbox you will get one...amazing or what.

ok thats it ;) if you ever need any help just email me and I'll sort you out.
email your

"you got that wrong"
"i liked this"
"you should of done this"
and
"OMG!! HELP!!"'s



to Jamesluton@hotmail.com

thanks to Defiler for some information ;)

[yAtEs]

"Keep it locked, keep it hardcore. Roots 'n' phuture. Peace."