9x/NT API Hooking via Import Tables | [yAtEs] | UG2003 | 10/Mar/03

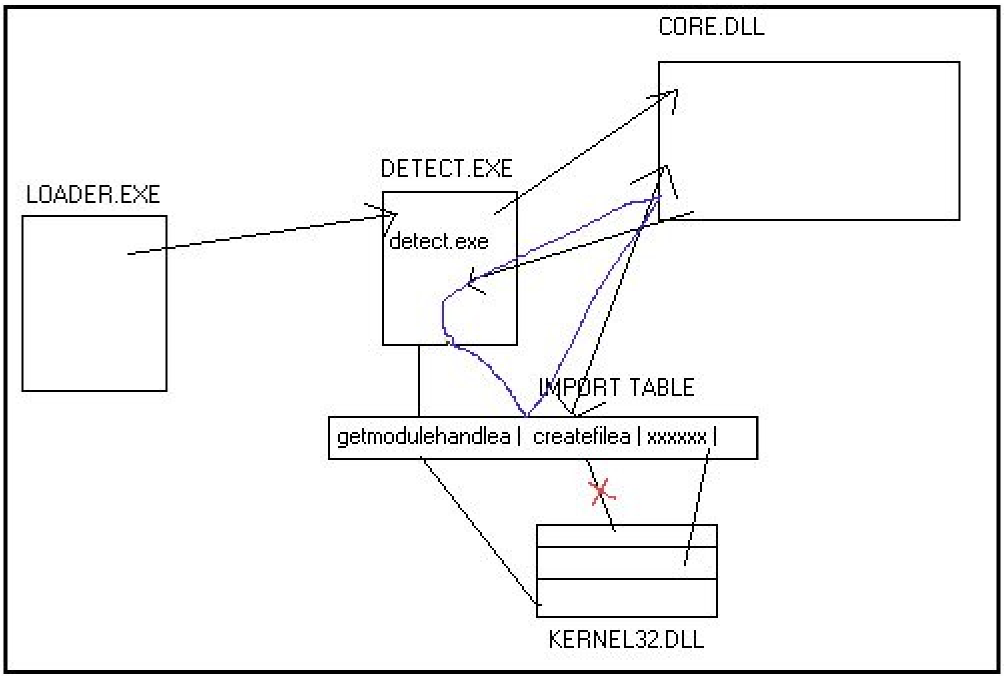
Hi, this is a follow up tutorial from my 'Understanding Import Tables' which
i wrote a couple of years ago, the aim of this tutorial is to show how
easy it can be to hook apis in a single thread process, the tutorial requires
you have knowledge of import table structures, i.e. u've read my previous
document.

wuts goin on

ok, so what we gonna do? im gonna explain how to code an api hook for
a little exe ive created which detects softice through CreateFileA.

The concept

We need to emulate createfilea and change its output, so we'll code a new
createfilea controling api, this will be done inside a dll, and basically the
way its going work is our softice detcting exe has createfilea in the import
table, this'll be in the kernel thunk, we'll create a dll and in its entrypoint
it'll scan the importtable for the createfilea then it'll replace with a pointer
to our new createfilea function. whad'ya mean I dont make sense, RIGHT!
you've asked for it now, perpare for an outstanding diagram.



Republished - 12th November 2007 – Robert Yates

Ok it wasnt quite what i expected but it'll do. Detect.exe will call a create on several SoftICE drivers, if all return false(-1) then softice isnt install but if a success is returned then SoftICE is installed, the Call CreateFile that it'll use is a call dword ptr [xxxxxx] (FF 15 xx xx xx xx) the xxxx is an address in the rdata kernel's first thunk, at run time this'll be the address of createfile in kernel32.dll, we need to simpley redirect this to our own function, so, we're gonna need some free space, i;ve chosen to use a dll. Our dll which ive decided to name core.dll is going to contain some code in its entry point to:

- **Find the EXE import table pointer**
- **Find the kernel import descriptor**
- **Find the First Thunk entry**
- **Scan the thunk for createfileapi**
- **Change entry to our code**

Sounds like a cunning plan doesnt it, ok so all this is going to be done in our DLL in the its entrypoint, the entrypoint is called during the LoadLibrary process, we need all this to be done at the start of our exe, so we need another cunning plan to load our dll on the detect.exe's entrypoint, ahh 2 cunning plans in one day we're on a roll now, for loading the dll on detects entrypoint i've come up with a novel idea of writing a new pieace of code called Loader.exe, heres its objectives:

- **Create the process detect.exe in suspended mode**
- **Readprocessmemory the first 500bytes from entrypoint**
- **Writeprocesmemory over the entrypoint with some dynamic LoadLibrary code**
- **Resume process**
- **Use various getcontext calls to wait until Loadlibrary finished(i.e. all hooking)**
- **Suspend process**
- **Writeprocessmemory over the old entrypoint code**
- **Restore exe to start position**
- **Resume process with hook in place :-)**

First we need a plain dll, here is one

```
.486
locals
jumps
.Model Flat ,StdCall
o equ offset
HINSTANCE EQU DWORD

extern CreateFileA :PROC
extern VirtualAlloc :PROC
extern VirtualFree :PROC
extern GetFileSize :PROC
extern CloseHandle :PROC
extern ReadFile :PROC
extern WriteFile :PROC
extern MessageBoxA :PROC
extern GetOpenFileNameA :PROC
extern ExitProcess :PROC
extern WriteProcessMemory :PROC
extern ReadProcessMemory :PROC
extrn SetFilePointer:PROC
extrn _wsprintfA:PROC
extrn lstrlen:PROC
extrn VirtualProtectEx:PROC
extrn OpenProcess:PROC
extrn GetThreadContext:PROC
extrn GetCurrentThread:PROC
extrn GetCurrentThreadId:PROC
extrn GetModuleHandleA:PROC
extrn GetProcAddress:PROC
extrn GetCurrentProcessId:PROC
extrn lstrcmp:PROC
extrn GetStdHandle:PROC
extrn SendDlgItemMessageA:PROC
extrn SendMessageA:PROC
extrn GetDlgItem:PROC
extrn lstrcat:PROC
extrn RtlZeroMemory:PROC
extrn GetModuleFileNameA:PROC
extrn GetLastError:PROC
include dbg.inc
.data
bytesrw dd 0

.code
DllEntry proc hinstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD

cmp [reason],1
jne no_core_4_u

push hinstDLL
```

```
call InitCore
no_core_4_u:
mov eax,1
ret
DllEntry Endp

;-----------------------------------------------------------------------------
InitCore proc DllModule:DWORD
pushad

call dbg1
db '[+] shh, Ive entered the target process',0
dbg1:
call dbg_string
call dbg_allout,1

popad
mov eax,1
ret
InitCore endp
;-----------------------------------------------------------------------------
End DllEntry
```

Its a basic dll which will output a message to debug.txt once loaded using my dbg.inc all source files, tools, are provided(link at bottom)

Ok so we have a dll, this is gonna the dll we inject into our target process, and like an evil spy its going to steal information and also modify the exes running behaviour, now we need the injecter which is loader.exe, here it is, take note of the points above as this is what its doing.

```
.486
locals
jumps
.Model Flat ,StdCall
o equ offset
extern CreateFileA :PROC
extern VirtualAlloc :PROC
extern VirtualFree :PROC
extern GetFileSize :PROC
extern CloseHandle :PROC
extern ReadFile :PROC
extern WriteFile :PROC
extern MessageBoxA :PROC
extern GetOpenFileNameA :PROC
extern ExitProcess :PROC
extern WriteProcessMemory :PROC
extern ReadProcessMemory :PROC
extern CreateProcessA :PROC
extern GetThreadContext :PROC
extern Sleep :PROC
extern CharLowerA :PROC
extern SetFileAttributesA :PROC
extern LoadLibraryA :PROC
extern FreeLibrary :PROC
extern SetThreadContext :PROC
extern ResumeThread :PROC
extern SuspendThread :PROC
extern VirtualProtect :PROC
extern GetProcAddress :PROC
extrn SetFilePointer:PROC
extrn _wsprintfA:PROC
extrn lstrlen:PROC
extrn VirtualProtectEx:PROC
extrn GetModuleHandleA:PROC
extrn GetStdHandle:PROC
extrn DialogBoxParamA:PROC
extrn LoadIconA:PROC
extrn SendMessageA:PROC
extrn GetWindowRect:PROC
extrn MoveWindow:PROC
extrn GetDesktopWindow:PROC
extrn SendDlgItemMessageA:PROC
extrn GetDlgItem:PROC
extrn ExitThread:PROC
extrn CreateThread:PROC
extrn lstrcat:PROC
extrn RtlZeroMemory:PROC
extrn CreateFileMappingA:PROC
extrn MapViewOfFileEx:PROC
extrn GetLastError:PROC
extrn GetModuleFileName:PROC
include dbg.inc
.data
tStartupInfo dd 44h
db 44h dup (?) ; startup info for the process were opening
tProcessInfo dd 4 dup (?) ; process / thread handles
filename db 'detect.exe',0
titlef db ' Hook Test',0
msgf db 'An error occured, see debug.txt for details.',0

;dynamic loader
write_data:
db 090h ; change to 0CCh if u wanna debug
call OverLibname
db 'core.DLL',0
OverLibname:
call OverLoadLib
VaLoadLibraryA dd 0
OverLoadLib:
pop eax
mov eax,[eax]
call eax
jmp $
```

```
waitp EQU $-write_data
write_data_len EQU $-write_data
Kernel32 db 'Kernel32.dll',0
FuncLL db 'LoadLibraryA',0
jmp_eip db 0EBh, 0FEh
align 4
.data?
threadid dd ?
bytesrw dd ?
fhandle dd ?
buffer dd ?
null dd ?
gEntryPoint dd ?
gImageBase dd ?
oldflags dd ?
memory_ptr dd ?
memory_size dd ?

myBuffer db 1000h dup(?)
align 4
my_context dd 100h dup (?)

.code
main:
LoadApi:
call GetModuleHandleA,o Kernel32
call GetProcAddress,eax,o FuncLL
mov [VaLoadLibraryA],eax
;--------------------------------.
Collect_PE_Image_Information.---------------------------

call CreateFileA, o filename,0C0000000h,0,0,3,80h,0
mov [fhandle],eax

call VirtualAlloc,0,1000h,1000h,4
mov [buffer],eax

call ReadFile,[fhandle],[buffer],1000h,o null,0
mov eax,[buffer]
mov edi,[eax+3ch]
lea eax,[eax+edi] ; EAX = PE-HEADER
mov ebx,[eax+28h]
mov [gEntryPoint],ebx ; save entrypoint
cmp ebx,0
jne ITS
call err3
db ' + Error! Null EntryPointed, close open processes',0
err3:
call dbg_string
call error
ITS:
```

```
mov ebx,[eax+34h]
mov [gImageBase],ebx

add [gEntryPoint],ebx

call VirtualFree,[buffer],0,8000h
call closehandle,[fhandle]
;--------------------------------------------------------------------------------
----------
mov [my_context], 00010000h+1+2+4+8+10h
call CreateProcessA, o filename, 0, 0, 0, 0, 4, 0, 0, o tStartupInfo, o
tProcessInfo
call dbg2
db '+ Loading Process ',0
dbg2:
call dbg_string
call dbg_allout,1
call VirtualProtectEx,[tProcessInfo],[gEntrypoint],1000h,40h,o oldflags

call ReadProcessMemory,[tProcessInfo],[gEntryPoint],o myBuffer,100h,0 ; save oep
data
mov eax,o myBuffer

call WriteProcessMemory,[tProcessInfo],[gEntryPoint],o write_data,100h,0
call dbg3
db '+ Injected Hook ',0
dbg3:
call dbg_string
call dbg_allout,1
jmp t4
db 'hehe omg here goes nothing!'
t4:
call ResumeThread, [tProcessInfo+4]

call dbg4
db '+ Starting Process - Passing Control to DLL, good luck! ',0
dbg4:
call dbg_string
call dbg_allout,1
call sleep,100h ; take a nap, we deserved it
call GetThreadContext, [tProcessInfo+4], o my_context
test eax, eax
jz anerror
mov edi,[gEntryPoint]
add edi,waitp-2 ; calculate the offset where JMP EIP is
ContextLoopE:
call GetThreadContext, [tProcessInfo+4], o my_context
test eax, eax
jz anerror
mov eax, [my_context+0b8h] ; CONTEXT+B8 = EIP
cmp eax, edi ; are we there yet?
jz run_app
```

```
call Sleep, 100h
jmp ContextLoopE

run_app:
call dbg8
db '+ Loader regained control, welcome back sir! ',0
dbg8:
call dbg_string
call dbg_allout,1
call SuspendThread, [tProcessInfo+4] ; STOP!....
call WriteProcessMemory,[tProcessInfo],[gEntryPoint],o mybuffer,100h,0 ; restore
code
mov edi,[gEntryPoint]
mov [my_context+0B8h],edi ; set EIP to start
call SetThreadContext, [tProcessInfo+4], o my_context

call ResumeThread, [tProcessInfo+4] ; CARRY ON! ...
push 0
call exitprocess ; we wont bother sticking around

anerror:
error:
call messageboxa,0,o msgf,o titlef,0
call exitprocess,0

end main
```

So now we have a loader and dll, we can test these on detect.exe which is provided compiled and with source below. If you run detect.exe it will say softice detected or it might say softice not detected, it doesnt really matter what the msg box says is true or not because our aim is more focused on switching the results of createfilea and therefore altering the follow of the program. Ok now if you run loader instead if all goes to plan the msgbox will be displayed again but a debug.txt will appear, take a look inside.

```
+ Loading Process
+ Injected Hook
+ Starting Process - Passing Control to DLL, good luck!
[+] shh, Ive entered the target process
+ Loader regained control, welcome back sir!
```

ah, now we can see that our dll has sucessfully entered the detect exe and printed out the message to the file, so now we have control and can alter things before the exe starts, so the next step is hook the createfilea import, which at the time of this dll loading we'll have the kernel address in the import table, so now we proceed to scan and replace.

we'll need a function to find the correct iid(image import descriptor) for the kernel, our function with take 3 params, location of the import table(iids), a text string of the library we're searching for and our module imagebase, the function will return eax = 1 and edi = IID if sucessfull, or eax = -1(0FFFFFFFh) if failed,

```
------------------------------------------------------------------------
FindThunk proc uses ebx ecx edx esi, IT:DWORD, LIBNAME:DWORD, IMAGEBASE:DWORD

mov esi,LIBNAME
mov edx,0
call lstrlen,esi
mov ecx,esi

mov eax,IT


scan_for_libname:
mov esi,[LIBNAME] ; our dll name
mov edi,[eax+0Ch] ; libname from first iid
add edi,[IMAGEBASE]
mov [temp],eax
call lstrcmp,edi,esi ; compare them
cmp eax,0
mov eax,[temp]
jne next_IID

mov edi,eax ; found a match, save iid into edi
mov eax,1 ; set eax to success

jmp FT_ExitPoint

next_IID:
add eax,14h ; next iid
cmp dword ptr [eax+0Ch],0 ; finished all?
jne scan_for_libname
mov eax,-1 ; set eax to failed


FT_ExitPoint:
ret
FindThunk endp
;*import.inc

------------------------------------------------------------------------
```

ok if you run through that its pretty simple, it takes our string reads the one from the import table and compares until we find which iid its in.

ok now lets use this in our core.dll under the first message.

```
kern32 db 'KERNEL32.dll',0
lkern32 db 'Kernel32.dll',0

call getmodulehandlea,0
mov [tImageBase],eax
add eax,[eax+3Ch]
add eax,80h ; PE+80 = location of import table
mov eax,[eax]
add eax, [tImageBase]
mov [tIID],eax

; *


call FindThunk,eax,o kern32,[tImageBase]
cmp eax,-1
jne hook_api

call FindThunk,[tIID],o lkern32,[tImageBase]
cmp eax,-1
jne hook_api

call err1
db '[o] Failed to find kernel descriptor',0
err1:
call dbg_string
call dbg_allout,1
call exitprocess,0

hook_api:

call dbg2
db '[+] Found kernel descriptor at ',0
dbg2:
call dbg_string
call dbg_dword,edi,0
call dbg_allout,1
```

theres our new code which makes use of the function with some error checking, if you look where the * is, put an int 3 here and i3here on in softice and we can trace to see if its working as well, after running you should get a debug.txt like this,

```
------------------------------------------------------------------------
+ Loading Process
+ Injected Hook
+ Starting Process - Passing Control to DLL, good luck!
[+] shh, Ive entered the target process
[+] Found kernel descriptor at 00403000
+ Loader regained control, welcome back sir!

------------------------------------------------------------------------
```

Ok we're nearly there now :) , now we have a point in the import table, we can get the pointer to the first thunk which will contain an array of kernel addresses and somewhere our createfile is in there, we'll replace the address with an offset to a new createfilea in our dll, ok here comes the big bit, lets start of by creating our new createfilea, i wont add the softice check code but just some simple log msgs for now, heres mine.

```
________________________________________________________________________
NewCreateFile Proc

pop eax
mov [ret_addr],eax


call wdbg1
db 13,10,'CreateFileA: ',0
wdbg1:
call dbg_string
call dbg_dword,[ret_addr],0


call wdbg2
db ' File = ',0
wdbg2:
call dbg_string

call dbg_string,dword ptr [esp]

nowmsg:


call [CreateF]


call wdbg3
db ' Handle: ',0
wdbg3:
call dbg_string
call dbg_dword,eax,0
call dbg_allout,1

push [ret_addr]
ret

NewCreateFile ENDP
```

----

okey dokey, thats our new createfile, the call in the middle to CreateF will be a dword where we hold the real createfilea address to call. ok now we need to store some info which is the real createfilea address and the kernel32 imagebase, after the found kernel message we'll add something like this,

----

```
push edi ; - kernel iid
;int 3

call GetModuleHandleA,o kern32 ; get base for libname
mov [module],eax

call getprocaddress,[module],o CreateFa
mov [CreateF],eax

pop edi
```

----

now we have some important info saved we're going to create the hooking proc now, this procedure with scan the thunk for a match of the api we're trying to hook, it'll then replace this with the offset of NewCreateFile. prodcedure takes the following params, IID the descriptor, imagebase the base of kernel, HookME a string of the api to hook and NewProc the offset of the new proc, i.e NewCreateFile., here it is

```
Hook_IAT proc uses edi ebx, IID:DWORD, IMAGEBASE:DWORD, HOOKME:DWORD, NEWPROC:DWORD
LOCAL TSize:DWORD
LOCAL spare:DWORD
LOCAL KIB:DWORD

call getprocaddress,[module],[HOOKME] ; get the address of what we are hooking!
cmp eax,0
jne hook_it

call err2
db 13,10,'+ Unable to Resolve address for wanted hook',0
err2:
call dbg_string
call dbg_allout,1
call exitprocess,0 ; eejit


hook_it:

mov [hooked],eax
mov edi,IID
mov edi,[edi+10h] ; get pointer to FirstThunk
add edi,[IMAGEBASE] ; EDI = array of all kernel functions

push eax ; save address of what we are replacing
call Get_Thunk_Size,edi ; external proc
imul eax,4
mov [TSize],eax ; save size in bytes


call unprotect,edi,[Tsize] ; get r/w access to whole thunk (external proc)

pop eax ; restore address of what we are replacing

find_import:
cmp [edi],eax ; EDI = current thunk ptr EAX = address to hook
jne scan_next_import ; DID WE FIND THE ADDRESS?

mov ebx,[NEWPROC] ; YES! move loc of new proc into ebx
mov [edi],ebx ; REPLACE THUNK POINTER!

; ****** You may wish to save all edi's here so you can replace the apis back
; if you intend to dump, or copy the original thunk back over, OR! add a new param
; to save the value

mov eax,1 ; success, ahh :-)

jmp Hook_IAT_ExitPoint ; get out of here

scan_next_import:
add edi,4 ; check next next thunk entry
cmp dword ptr [edi],0 ; are we at the end?
jne find_import ; is there still hope?

mov eax,-1 ; nope, dam

Hook_IAT_ExitPoint:


ret
Hook_IAT endp
```

---

there it is, thats going into imports.inc, yo may notice *external proc this are extra functions i've written, heres the code,

---

```
unprotect proc loc:dword, tsize:dword ; provide r/w access to a memory range
pushad

call VirtualProtectEx,[tProcessInfo],loc,tsize,40h,o oldflags

popad
ret
unprotect endp

add following to core.asm near the top

call GetCurrentProcessId
mov [processid],eax
call OpenProcess, PROCESS_ALL_ACCESS, 0, processid
mov [tProcessInfo],eax
```

```
Get_Thunk_Size PROC uses edi, THUNK:DWORD

mov eax,0
mov edi,[THUNK]

count_em_up:
cmp dword ptr [edi],0
je Get_Thunk_Size_ExitPoint
inc eax
add edi,4
jmp count_em_up


Get_Thunk_Size_ExitPoint:
ret
Get_Thunk_Size ENDP
```

---

now we have some groovy functions, lets put em into action, now we are ready to hook createfilea with our hook_iat function, in core.asm add the following, should be after a pop edi,

---

```
CreateFa db 'CreateFileA',0 ; createfilea ascii

call Hook_IAT, edi,[tImageBase],o CreateFa ,o NewCreateFile
cmp eax,-1
jne hookeda

call err3
db '[+] Hook Failed ',0
err3:
call dbg_string
call dbg_dword,edi,0
call dbg_allout,1
call exitprocess,0

hookeda:
call dbg3
db '[i] API Hooked ',0
dbg3:
call dbg_string
call dbg_dword,edi,0
call dbg_allout,1
```

----
ok compile it all up and lets run the loader again now, if all goes to plan then it should run through and produce a debug.txt, lets take a look,

----

```
+ Loading Process
+ Injected Hook
+ Starting Process - Passing Control to DLL, good luck!
[+] shh, Ive entered the target process
[+] Found kernel descriptor at 00403000
[i] API Hooked 00403000
+ Loader regained control, welcome back sir!

CreateFileA: 004012C1 File = \\.\SICE Handle: FFFFFFFF
```

----
wow cool huh, it has logged our createfilea, now depending on your system setup the call to open the softice driver might fail like mine or it might return a handle number, although now i've demostrated the basis of iat hooking i'll add code to change the output of the attempted sice access to make it fail in case you do have softice loaded. note. this is a 9x detect only, and \\.\NTICE fails on 2k anyway with the new DS2.7

```
________________________________________________________________________
NewCreateFile Proc


pop eax
mov [ret_addr],eax


call wdbg1
db 13,10,'CreateFileA: ',0
wdbg1:
call dbg_string
call dbg_dword,[ret_addr],0


call wdbg2
db ' File = ',0
wdbg2:
call dbg_string

call dbg_string,dword ptr [esp]

mov eax,[esp]
add eax,4 ; skip \\.\
cmp [eax],'ECIS'
jne nowmsg

add esp,1Ch ; remove params off stack
mov eax,-1 ; set eax to fail
jmp skip_call ; skip the real api

nowmsg:


call [CreateF]

skip_call:

call wdbg3
db ' Handle: ',0
wdbg3:
call dbg_string
call dbg_dword,eax,0
call dbg_allout,1

push [ret_addr]
ret

NewCreateFile ENDP
________________________________________________________________________
```

ta da! cool huh, now we have successfully altered the way our target works and enabled use to have control over the apis, theres many uses for this technique, e.g. if you were unpacking a program where might be an api during the process which you would want to hook because at that time the import table is unpacked

or theres something that needs changing, the other use is as you may have seen, logging apis and what they're doing, as a little bonus i've included an api logger with a couple more of apis, here the output from me running it on media player.

```
________________________________________________________________________
+ Loading Process
+ Injected Hook
+ Starting Process - Passing Control to DLL, good luck!
[+] shh, Ive entered the target process
[+] Found kernel descriptor at 00422000
[%] Hooked VirtualAlloc
[%] Hooked GlobalAlloc
[%] Hooked GetProcAddress
+ Loader regained control, welcome back sir!
VirtualAlloc : 00416C7D Block Size: 00002000 For Region: 00860000
VirtualAlloc : 00416D2D Block Size: 00001000 For Region: 00860000
VirtualAlloc : 00416A79 Block Size: 00001000 For Region: 00960000
GetProcAddress : 004066EB getting address for api GetOpenFileNameA

________________________________________________________________________
```

with this logger i've edited the code slightly, the hook_iat function now saves the real api into a extra param provided, and it also displays a msg if the hook was successful I didnt add any code to say hooking failed, whatever functions it finds it will hook and display a message in the log, some programs may use getprocaddress to get api addresses if they're not found in the import table, so best solution is we hook getprocaddress and like our createfile we check to see which api it is resolving and if its one we are interested in, we return the offset of our alternative function into eax.

phew, and that brings me to the end of the tutorial, i hope you found it usefull and if you decided to create more usefull hooks for hooks.inc let me know :-)

yates.

Files. *(publishers may want to add mirror links or alter)*

Basic Loader and Template with full compilation tools and configuration
CreateFile Hook
Api Logger

All Code provided is free for public use.