

Forum:[Hack Forums](#)

Thread: [emu8086 Virus Alert](#)

MatthewHodnick - March 3, 2012, 7:30 pm

I compiled the example file for factorial in emu8086, and ESET NOD32 said it's a probable variant of Neko.1990.Damaged virus. Now, I know it's not really a virus, but I am curious to know why it would be considered as one.

Code:

Code ; this example gets the number from the user,
; and calculates factorial for it.
; supported input from 0 to 8 inclusive!

name "fact"

; this macro prints a char in AL and advances
; the current cursor position:

```
putc macro char
    push ax
    mov al, char
    mov ah, 0eh
    int 10h
    pop ax
endm
```

org 100h

jmp start

result dw ?

start:

; get first number:

```
mov dx, offset msg1
mov ah, 9
int 21h
jmp n1
msg1 db 0Dh,0Ah, 'enter the number: $'
n1:
```

call scan_num

; factorial of 0 = 1:

mov ax, 1

cmp cx, 0

je print_result

; move the number to bx:

; cx will be a counter:

mov bx, cx

mov ax, 1

mov bx, 1

calc_it:

mul bx

cmp dx, 0

jne overflow

inc bx

loop calc_it

mov result, ax

print_result:

; print result in ax:

mov dx, offset msg2

mov ah, 9

int 21h

jmp n2

msg2 db 0Dh,0Ah, 'factorial: \$'

n2:

mov ax, result

call print_num_uns

jmp exit

overflow:

mov dx, offset msg3

mov ah, 9

int 21h

jmp n3

msg3 db 0Dh,0Ah, 'the result is too big!', 0Dh,0Ah, 'use values from 0 to 8.\$'

n3:

jmp start

exit:

; wait for any key press:

mov ah, 0

int 16h

ret

```
.....  
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,  
;;; these functions are copied from emu8086.inc ;;;  
.....  
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

; gets the multi-digit SIGNED number from the keyboard,
; and stores the result in CX register:

SCAN_NUM PROC NEAR

PUSH DX

PUSH AX

PUSH SI

MOV CX, 0

; reset flag:

MOV CS:make_minus, 0

next_digit:

; get char from keyboard

; into AL:

MOV AH, 00h

INT 16h

; and print it:

MOV AH, 0Eh

INT 10h

; check for MINUS:

CMP AL, '-'

JE set_minus

; check for ENTER key:

CMP AL, 0Dh ; carriage return?

JNE not_cr

JMP stop_input

not_cr:

CMP AL, 8 ; 'BACKSPACE' pressed?

JNE backspace_checked

MOV DX, 0 ; remove last digit by

MOV AX, CX ; division:

```

    DIV    CS:ten          ; AX = DX:AX / 10 (DX-rem).
    MOV    CX, AX
    PUTC   ''              ; clear position.
    PUTC   8               ; backspace again.
    JMP    next_digit
backspace_checked:

```

```

    ; allow only digits:
    CMP    AL, '0'
    JAE    ok_AE_0
    JMP    remove_not_digit
ok_AE_0:
    CMP    AL, '9'
    JBE    ok_digit
remove_not_digit:
    PUTC   8              ; backspace.
    PUTC   ''             ; clear last entered not digit.
    PUTC   8              ; backspace again.
    JMP    next_digit ; wait for next input.
ok_digit:

```

```

    ; multiply CX by 10 (first time the result is zero)
    PUSH   AX
    MOV    AX, CX
    MUL    CS:ten          ; DX:AX = AX*10
    MOV    CX, AX
    POP    AX

```

```

    ; check if the number is too big
    ; (result should be 16 bits)
    CMP    DX, 0
    JNE    too_big

```

```

    ; convert from ASCII code:
    SUB    AL, 30h

```

```

    ; add AL to CX:
    MOV    AH, 0
    MOV    DX, CX          ; backup, in case the result will be too big.
    ADD    CX, AX
    JC     too_big2        ; jump if the number is too big.

```

```

    JMP    next_digit

```

```

set_minus:
    MOV    CS:make_minus, 1
    JMP    next_digit

```

```

too_big2:
    MOV    CX, DX          ; restore the backup value before add.
    MOV    DX, 0           ; DX was zero before backup!

```

```

too_big:
    MOV    AX, CX
    DIV    CS:ten ; reverse last DX:AX = AX*10, make AX = DX:AX / 10
    MOV    CX, AX
    PUTC   8      ; backspace.
    PUTC   ' '    ; clear last entered digit.
    PUTC   8      ; backspace again.
    JMP    next_digit ; wait for Enter/Backspace.

```

```

stop_input:
    ; check flag:
    CMP    CS:make_minus, 0
    JE     not_minus
    NEG    CX

```

```

not_minus:

```

```

    POP    SI
    POP    AX
    POP    DX
    RET

```

```

make_minus    DB    ?      ; used as a flag.
SCAN_NUM      ENDP

```

; this procedure prints number in AX,
; used with PRINT_NUM_UNNS to print signed numbers:

```

PRINT_NUM     PROC    NEAR

```

```

    PUSH    DX
    PUSH    AX

```

```

    CMP     AX, 0
    JNZ     not_zero

```

```

    PUTC    '0'
    JMP     printed

```

```

not_zero:

```

```

    ; the check SIGN of AX,
    ; make absolute if it's negative:

```

```

    CMP     AX, 0
    JNS     positive
    NEG     AX

```

```

    PUTC    '-'

```

```

positive:

```

```

    CALL    PRINT_NUM_UNNS

```

```

printed:

```

```

    POP     AX

```

```
    POP    DX
    RET
PRINT_NUM    ENDP
```

```
; this procedure prints out an unsigned
; number in AX (not just a single digit)
; allowed values are from 0 to 65535 (FFFF)
```

```
PRINT_NUM_UNS PROC NEAR
```

```
    PUSH   AX
    PUSH   BX
    PUSH   CX
    PUSH   DX
```

```
; flag to prevent printing zeros before number:
MOV     CX, 1
```

```
; (result of "/ 10000" is always less or equal to 9).
MOV     BX, 10000    ; 2710h - divider.
```

```
; AX is zero?
CMP     AX, 0
JZ      print_zero
```

```
begin_print:
```

```
; check divider (if zero go to end_print):
CMP     BX, 0
JZ      end_print
```

```
; avoid printing zeros before number:
CMP     CX, 0
JE      calc
; if AX<BX then result of DIV will be zero:
CMP     AX, BX
JB      skip
```

```
calc:
```

```
MOV     CX, 0    ; set flag.
```

```
MOV     DX, 0
DIV     BX    ; AX = DX:AX / BX (DX=remainder).
```

```
; print last digit
; AH is always ZERO, so it's ignored
ADD     AL, 30h    ; convert to ASCII code.
PUTC    AL
```

```
MOV     AX, DX    ; get remainder from last div.
```

```
skip:
```

```
; calculate BX=BX/10
```

```
PUSH  AX
MOV   DX, 0
MOV   AX, BX
DIV   CS:ten ; AX = DX:AX / 10 (DX=remainder).
MOV   BX, AX
POP   AX
```

```
JMP   begin_print
```

```
print_zero:
    PUTC  '0'
```

```
end_print:
```

```
    POP   DX
    POP   CX
    POP   BX
    POP   AX
    RET
PRINT_NUM_UNS  ENDP
```

ten DW 10 ; used as multiplier/divider by SCAN_NUM & PRINT_NUM_UNS.

MUT4NT - March 4, 2012, 2:29 pm

the program uses the int 16 maybe can identified as keylogger

DexHacker - March 4, 2012, 3:01 pm

Virus Co will report many things has virus, eg anything that's coded in asm is suspect.

Example: if your write a win32 hello world! and use
section '.text' code readable executable writeable
instead of

section '.text' code readable executable
Its will be reported by some AV as a virus.

MUT4NT - March 4, 2012, 3:18 pm

(Mar 4, 2012 04:01 PM)DexHacker Wrote: [\[CITE\]](#) Virus Co will report many things has virus, eg anything that's coded in asm is suspect.that can not be, all code of a program is asm also the program if you do set is a single segment

DexHacker - March 4, 2012, 6:13 pm

(Mar 4, 2012 04:18 PM)Mut4nt Wrote: [\[CITE\]](#) that can not be, all code of a program is asm also the program if you do set is a single segmentYour wrong, even if you forget about .net, if you open a exe coded with c or c++ in a hexeditor theres tel tail signs, of what it was compiled with.

Then there's size, if you encrypted it, its then seen as suspect, note: i did not say that all ASM code apps are seen as virus, but anything coded in full ASM is seen as suspect, and if AV see too many suspect things in a exe, its reported as a virus.

MUT4NT - March 4, 2012, 8:25 pm

(Mar 4, 2012 07:13 PM)DexHacker Wrote: [\[CITE\]](#) Your wrong, even if you forget about .net, if you open a exe coded with c or c++ in a hexeditor theres tel tail signs, of what it was compiled with.

Then there's size, if you encrypted it, its then seen as suspect, note: i did not say that all ASM code apps are seen as virus, but anything coded in full ASM is seen as suspect, and if AV see too many suspect things in a exe, its reported as a virus. clear programs created as .net, java, vb are interpreted not executed, I meant program made in languages such as assembler, C/C + +, pascal, delphi(code native) these are compiled and the machine code is the same different is when the amount of size machine instructions using each for its purpose.
programming in assembly language programming means not only virus

DexHacker - March 4, 2012, 9:21 pm

Take a c++, or pascal compiled program and open them in a hex editor and see the ASCII part, you will find label as what it was coded in in plain txt.

Even shell code, example see my post here:

<https://hackforums.net/showthread.php?tid=1308994&pid=18287143#pid18287143>

Look up what mingw is.

MUT4NT - March 4, 2012, 9:46 pm

what? that are only exported function names

as I said in a compiled program can see the assembly of the program, not necessarily have to be done in assembly for view the machine instructions

DexHacker - March 9, 2012, 5:54 pm

See here: this shows that normally AV co, can tell what a app/virus was coded in

<https://hackforums.net/showthread.php?tid=2287837>