# Using Computer Programming Competition for Cyber Education

Oded Margalit
Cybersecurity Center of Excellence
IBM
Israel

*Abstract*— **Contests are one of the best ways to teach. It serves as a gamification of the learning process. In the cyber security field there are two additional unique obstacles: the first is that we don't want to teach criminal activities and the second is that we actually don't really know what the future cyber world will actually need. Both this problems are solved by asking to solve hard out-of-the-box computer programming tasks that are correlated to the current cyber security techniques.**

***Computer science education, Constraint satisfaction, Macro and assembly language, Optimization***

## I. INTRODUCTION

IEEEXtreme is a global challenge in which teams of IEEE Student members compete in a 24-hour time span against each other to solve a set of programming problems. The competition can be viewed as not only student competition but also as a way to promote real-world programming education ([4]).

The author has served as a judge in the last eight competitions and wrote some hard challenges. Some of these challenges can be viewed as an education in the cyber security domain. In this paper we will walk through three of the programming challenges that the author composed, and explain the connection of each one of them to the cyber domain.

Section II raises the main claim (that difficult programming challenges can be used as a cyber education). Section III explains the 8086 assembly language challenge. Section IV gives a hardware-like implementation of an open problem. Section V shows an example of a problem that requires an out-of-the-box thinking. Section VI summarizes the paper and give some ideas to future research.

## II. CHALLENGES AS TOOLS FOR CYBER EDUCATION

### A. What cyber education should teach

When teaching an engineer to build a bridge, one can know quite well what the relevant knowledge is and how to convey it. Unlike such structured curriculum, in cyber domain, where the main idea is to 'think out of the box' and invent things that no one has thought about before.

Therefore, the right way (only way?) is by letting the student experience, first handedly, similar tasks that can lead him toward the conclusion that nothing is impossible. Preferably while working in relevant field (hardware and Software reverse engineering; 'bending the rules'.

### B. Why difficult challenges teach that

In the rest of the paper we will go through three representative examples of such challenges and explain why each one of them is related to the cyber domain and how solving it can teach the student cyber. For more research see [1], [2] and [3].

Hard challenges are not new – see [6] as an example of a CTF (Capture The Flag) game from more than a decade ago, which has mathematical, programming, reverse engineering, cryptography, steganography and even gaming challenges. Such challenges has created generations of hackers who have learned from their peers. One can see evidence for such learning process in the discussion forums where solvers are talking about how they solved the challenges and exchanges ideas.

## III. 8086 ASSEMBLY CHALLENGE (2014)

### A. The Problem

The task was to write a program, in any one of the supported programming languages, which gives the same output for the same input as the following MS-DOS 8086 assembly program which is given as the following hexadecimal dump:

```
BF 00 04 BE C0 00 56 31 C9 B4 00 CD 16
3C 2E AA E0 F7 F7 D1 29 D2 89 CD 5B 53
FE 07 75 03 43 EB F9 BF 00 02 89 F9 89
F8 F3 AA 89 FE AC 89 C3 FE 07 80 FB 2E
75 F6 FE 0F 5E 56 89 E9 AC 89 C3 FE 0F
7C D5 E2 F7 42 5E 56 89 E9 F3 A6 75 CA
5D 92 D4 0A E8 00 00 86 C4 04 30 CD 29
C3
```

This challenge is an example for cyber education since it requires reverse engineering (translating an assembly language program back into high level language) which is a crucial expertise in the cyber domain.

As an evidence for its difficulty we can note that all the programming tasks till this one were solved in less than 15 minutes by dozen of teams, while this task was not solved (even after giving many hints) for hours.

### B. Disassembly

Just like in Malware research and other cyber related tasks, the first step in solving this problem is to reverse engineer it. The hex dump above can be translated from machine language to assembly.

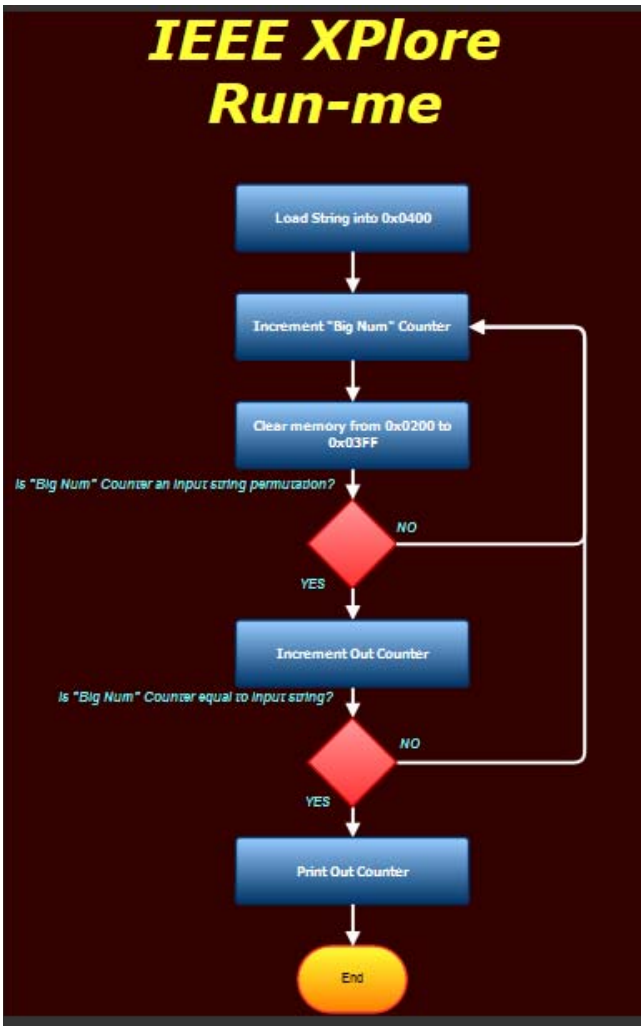
This process is straightforward, especially for such a small program (92 bytes). We are using here (and in the rest of this section) snippets taken from Marcelo's blogpost ([5]).

978-1-5090-1018-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SWSTE.2016.22

```
;====================================
; IEEE Xtreme 8.0 (2014)
; IEEE Xplore Run-me Problem
;====================================
start:    MOV   DI,0400h
          MOV   SI,00C0h
          PUSH  SI
          XOR   CX,CX
L003:     MOV   AH,00h
          INT   16h
          CMP   AL,2Eh
          STOSB
          LOOPNZL003
          NOT   CX
          SUB   DX,DX
          MOV   BP,CX
L005:     POP   BX
          PUSH BX
L001:     INC   BYTE PTR [BX]
          JNZ   L002
          INC   BX
          JMP   L001
L002:     MOV   DI,0200h
          MOV   CX,DI
          MOV   AX,DI
          REPZ  STOSB
          MOV   SI,DI
L004:     LODSB
          MOV   BX,AX
          INC   BYTE PTR [BX]
          CMP   BL,2Eh
          JNZ   L004
          DEC   BYTE PTR [BX]
          POP   SI
          PUSH  SI
          MOV   CX,BP
L006:     LODSB
          MOV   BX,AX
          DEC   BYTE PTR [BX]
          JL    L005
          LOOP  L006
          INC   DX
          POP   SI
          PUSH  SI
          MOV   CX,BP
          REPZ  CMPSB
          JNZ   L005
          POP   BP
          XCHG  DX,AX
          AAM
          CALL  L007
L007:     XCHG  AL,AH
          ADD   AL,30h
          INT   29h
          RET
```

### C. *High Level Understanding*

The next step, after getting from machine code to assembly language, is to obtain a higher level understanding of the program. In our case, the program works as shown in the following diagram:




The program computes the index of the input in the list of all multiset permutations of the input. The way it is done is by generating, for n-long input string all the 256n possible n-tuples of chars; counting only those which are anagram of the input; until reaching the input.

The original 8086 program takes seconds for (a very) short string, but ages for longer string. Understanding the way it works, reverse engineering the algorithm and a basic understanding of computer science algorithms enables the solver to rewrite it in any computer language.

## IV. BOOLEAN FUNCTION CHALLENGE (2015)

### A. *The Problem*

This problem is an open question in disguise. The first part is to understand what the problem is. Here is a Py-thon code that emulates a Boolean circuit

```
I=10*[]
B=50*[0]
M=256*[0]
```

```
def f(I):
    B[49]=B[39];B[48]=B[38];B[47]=B[37];
    B[46]=B[36];B[45]=B[35];B[44]=B[34];
    B[43]=B[33];B[42]=B[32];B[41]=B[31];
    B[40]=B[30];
    B[39]=B[29];B[38]=B[28];B[37]=B[27];
    B[36]=B[26];B[35]=B[25];B[34]=B[24];
    B[33]=B[23];B[32]=B[22];B[31]=B[21];
    B[30]=B[20];
    B[29]=B[19];B[28]=B[18];B[27]=B[17];
    B[26]=B[16];B[25]=B[15];B[24]=B[14];
    B[23]=B[13];B[22]=B[12];B[21]=B[11];
    B[20]=B[10];
    B[19]=B[9];B[18]=B[8];B[17]=B[7];
    B[16]=B[6];B[15]=B[5];B[14]=B[4];
    B[13]=B[3];B[12]=B[2];B[11]=B[1];
    B[10]=B[0];
    B[9]=I[9];B[8]=I[8];B[7]=I[7];
    B[6]=I[6];B[5]=I[5];B[4]=I[4];
    B[3]=I[3];B[2]=I[2];B[1]=I[1];
    B[0]=I[0];
    x1=I[0]|I[1];x2=x1|I[2];x3=x2|I[3];
    x4=x3|I[4];x5=x4|I[5];x6=x5|I[6];
    x7=x6|I[7];x8=x7|I[8];x9=x8|I[9];
    y1=(not x9)|(I[0]&I[1]);
    y2=y1|(x1&I[2]);y3=y2|(x2&I[3]);
    y4=y3|(x3&I[4]);y5=y4|(x4&I[5]);
    y6=y5|(x5&I[6]);y7=y6|(x6&I[7]);
    y8=y7|(x7&I[8]);y9=y8|(x8&I[9]);
    c0=B[0]|B[10]|B[20]|B[30]|B[40];
    c1=B[1]|B[11]|B[21]|B[31]|B[41];
    c2=B[2]|B[12]|B[22]|B[32]|B[42];
    c3=B[3]|B[13]|B[23]|B[33]|B[43];
    c4=B[4]|B[14]|B[24]|B[34]|B[44];
    c5=B[5]|B[15]|B[25]|B[35]|B[45];
    c6=B[6]|B[16]|B[26]|B[36]|B[46];
    c7=B[7]|B[17]|B[27]|B[37]|B[47];
    c8=B[8]|B[18]|B[28]|B[38]|B[48];
    c9=B[9]|B[19]|B[29]|B[39]|B[49];
    c10=not (c0 | c1);
    c11=c0^c1;c12=c0&c1;
    c20=(c10&(not c2));
    c21=(c10&c2)|(c11&(not c2));
    c22=(c11&c2)|(c12&(not c2));
    c23=(c12&c2);
    c30=(c20&(not c3));
    c31=(c20&c3)|(c21&(not c3));
    c32=(c21&c3)|(c22&(not c3));
    c33=(c22&c3)|(c23&(not c3));
    c34=(c23&c3);
    c40=(c30&(not c4));
    c41=(c30&c4)|(c31&(not c4));
    c42=(c31&c4)|(c32&(not c4));
    c43=(c32&c4)|(c33&(not c4));
    c44=(c33&c4)|(c34&(not c4));
    c45=(c34&c4);
    c51=(c40&c5)|(c41&(not c5));
    c52=(c41&c5)|(c42&(not c5));
    c53=(c42&c5)|(c43&(not c5));
    c54=(c43&c5)|(c44&(not c5));
    c55=(c44&c5)|(c45&(not c5));
    c62=(c51&c6)|(c52&(not c6));
    c63=(c52&c6)|(c53&(not c6));
    c64=(c53&c6)|(c54&(not c6));
    c65=(c54&c6)|(c55&(not c6));
    c73=(c62&c7)|(c63&(not c7));
    c74=(c63&c7)|(c64&(not c7));
    c75=(c64&c7)|(c65&(not c7));
    c84=(c73&c8)|(c74&(not c8));
    c85=(c74&c8)|(c75&(not c8));
    c95=(c84&c9)|(c85&(not c9));
    e=(not c95)|y9;
    a=10*[0]
    a[0]=e|
         ((((not c0)&(not c1)& (not c2)&
            (not c3)&(not c4))|
           (c0&c1&c2&c3&c4))^
          c0^c1^c2^c3^c4^
          (c3&(((c0^c8)&c1&c2&c4)^
            ((((c0^c1)&c2&c5)^(c1&c4&c7))&
         c8))));
    a[1]=e|((((not c0)&(not c1)&(c2)&
         (not c5)&(c6))|(c0&c1&((not c2)&
         (not c6))&c5))^c0^c1^c2^c5^c6^
         (c4&((c0&c1&((c2&c3)^(c5&c6)))^
         (((c1&c7)^(c6&c9))&c3&c8))));
    a[2]=e|((((not c0)&(not c1)&(    c3)&
         (not c5)&(not c7))|(c0&c1&
         (not c3)&c5&c7))^c0^c1^c3^c5^c7^
          (c0&c1&c2&(c3^c4)&c5)^
          ((c3^c4)&c5&c7&c8&c9));
    a[3]=e|((c3&c5)^(c3&c6)^(c3&c8)^
         (c3&c9)^(c5&c6)^(c5&c8)^(c5&c9)^
         (c6&c8)^(c6&c9)^(c8|c9)^
         c3^c5^c6^c8^c9^(c0&c1&c3&c6&c9));
    a[4]=e|((c2&c5)^(c2&c7)^(c2&c8)^
         (c2&c9)^(c5&c7)^(c5&c8)^(c5&c9)^
         (c7&c8)^(c7|c9)^(c8&c9)^c2^c5^c7^
         (((c0&c5&c6)^(c1&c3&c4))&c7&c8));
    a[5]=e|((c0&c1)^c0^c2^c4^c6^c7^
         (c0&c1&c2&c3&c4)^
         (((c0&((c3&c5)^(c2&c4)))^
         (c1&c4&c6))&c7&c8)^(c3&c4&c6&
         ((c2&c9)^(c5&c7))));
    a[6]=e|(c0^c1^c3^c4^c7^
         (c0&c1&c2&c4&c9)^(c0&((c1&c4)^
         (c3&c8))&c5&c7)^((((((c0^c1)&c5)^
         (c0&c4))&c2)^(c1&(c2^c7)&c4))&
         c6&c8));
    a[7]=e|(c2^c3^c4^(c0&((c2&c3)^
         ((c2^c3)&c7))&c4&c8)^((((c0^c1)&
         c3&c5)^(((c0^c1)&(c4^c5))&c6))
         &c7&c8));
    M[sum([a[i]<<i for i in range(8)])]=1

output=int(input())
for i in range(output):
    f([ord(c)&1 for c in input()])
output = 1000*sum(M)-output
print(output)
```

As before, this challenge is another example for cyber education since it requires hardware reverse engineering (translating low level hardware design translating back into high level) which is another important expertise in the cyber domain.

### *B. High Level explanation*

There are 252 possible unordered quintuples of digits. The program is mimicking a VHDL implementation of a Boolean circuit that receives a stream of digits (the buffer I

has 10 bits and when exactly one of them is on – it represents a digit) and counts that number of such quintuples. Note that when the Hamming weight of I is not 1 or when five consecutive digits are not different – the program considers it as 255 (0xff).

So the best one can expect is to cover 253 possibilities from the memory array M. The task is to get there in a shortest sequence.

## V. OUT OF THE BOX THINKING (2009)

### A. The Problem

The free text problem definition is:

There is a nice game one can play with phone numbers: trying to make an exercise from it whose result is 100. The rules of the games are that you can add the four basic math operations (+, -, *, /) and parentheses between the digits such that the result is a valid exercise whose result is 100.

Note that you must use all the digits, keep the order of the digits and you are not allowed to combine several digits into a number. Thus the solutions "5*5*4", "5*5*4+0*1*5*5" and "5*(5+5+1+0)+45" are not a valid ones for the number 5551045, while the solution "5*(5+5+1+0+4+5)" is.

But the trick is that we don't ask for a solution for the verbally described problem above, but rather passing the following shell script:

```
#/bin/csh
# Make sure we are using only the input
# digits and allowed operators
tr -d '+\-*/()' < output | diff input -
if ($? == 1) exit 1
# Make sue no two digits are left with-out
# an operator between them
grep '[0-9][0-9]' output
if ($? == 0) exit 1
# Once we passed the preconditions above,
# Score according to number of exercises
# solved correctly
sed 's/$/-99.99999999/' output | bc -l | grep
-c '^.00000'
```

The 'out-of-the-box' part of the problem is to find a way to 'cheat' the shell script to accept a solution for an input for which there is no solution for the problem as described in the beginning of this section.

This last challenge exemplify cyber education since it is a good example of cyber-attack, where one abuses the 'rules of the game' in order to win; but, of course, teaching this specific example, while helps to exercise out-of-the-box thinking in programming environment, it may not lead to any illegal practice.

Just like the other two examples above, this challenge can be used as cyber education since it requires one to think 'outside of the box'.

### B. The Solution

The idea is to use the fact that bc (Unix's basic calculator) is a complete programming language. The first check in the shell script was supposed to make sure that one would not be able to use any of bc's advanced features, but one feature has escaped the testing: one can use comments.

For example, the input 5750004 cannot be solved using the rules of the game, but one can cheat the script to solve it by using a comment: 5*/*7*/5*(0+0+0+4). Note that this solution pass the script since we are using original input; interleaved with the allowed characters: +,-,*,/,(,); with no two consecutive digits; and running bc yields 100.se Times 8-point type, single-spaced. To help your readers, avoid using footnotes altogether and include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence).

## VI. CONCLUSION

This paper gave three examples of computer programing challenges that were used in IEEEXtreme contests over the years. All these challenges are hard to solve, requires knowledge that is related to cybersecurity expertise on one hand, but does not lend itself to any illegal activities on the other hand.

Although the author is not aware of any research to show the effectiveness of these challenges as cyber education means – the research assumption is that it is positive and therefore such challenging programming tasks should be continued in future competitions.


## ACKNOWLEDGMENT

The author would like to thank IEEEXtreme team for the opportunity to write programming tasks and run it on thousands of students from all over the world; and the anonymous referees for their helpful comments.


## REFERENCES


[1] Boopathi, K., S. Sreejith, and A. Bithin. "Learning Cyber Se-curity Through Gamification." Indian Journal of Science and Technology 8.7 (2015): 642-649

[2] Chapman, Peter, Jonathan Burket, and David Brumley. "PicoCTF: A game-based computer security competition for high school students." 2014 USE-NIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). 2014.

[3] Efstratios Gavas, Nasir Memon, Douglas Britton, "Winning Cybersecurity One Challenge at a Time", IEEE Security & Privacy, vol.10, no. 4, pp. 75-79, July-Aug. 2012, doi:10.1109/MSP.2012.112

[4] Machado, Ricardo J and Guerreiro, Pedro and Delimar, Marko and Brito, Miguel A and others. "IEEEXtreme: From a Student Competition to the Promotion of Real-world Programming Education." 39th ASEE/IEEE Frontiers in Education Conference. 2009.

[5] Marcelo, (19 November 2014). Xtreme Xplained. In the blog-post http://xtreme-xplained.blogspot.co.il/2014/11/run-me.html.

[6] Electrica, the puzzle challenge, http://www.caesum.com/game