**Chosen project:** CSpawn – exercise 5, page 8 in Giant Black Book of Computer Viruses

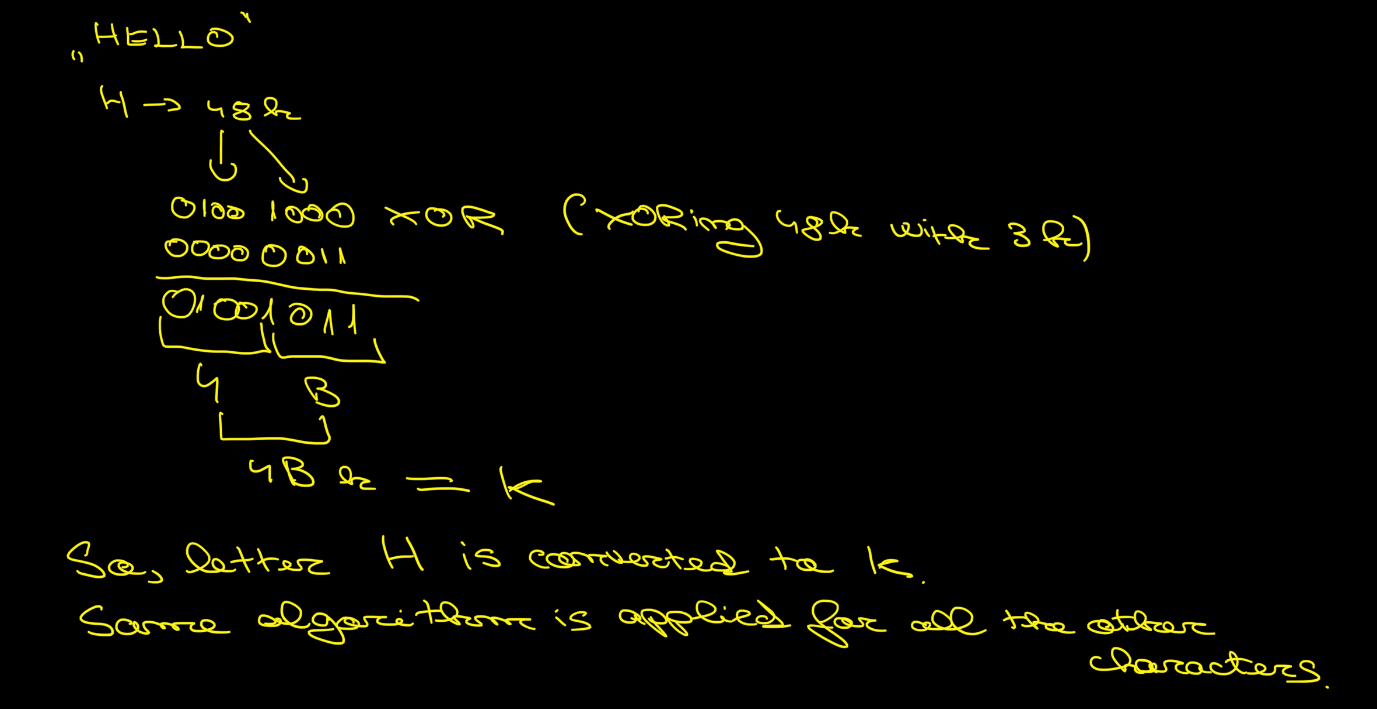
**Students:** Săndulescu Răzvan-Alexandru

Rotaru Emilian-Teodor

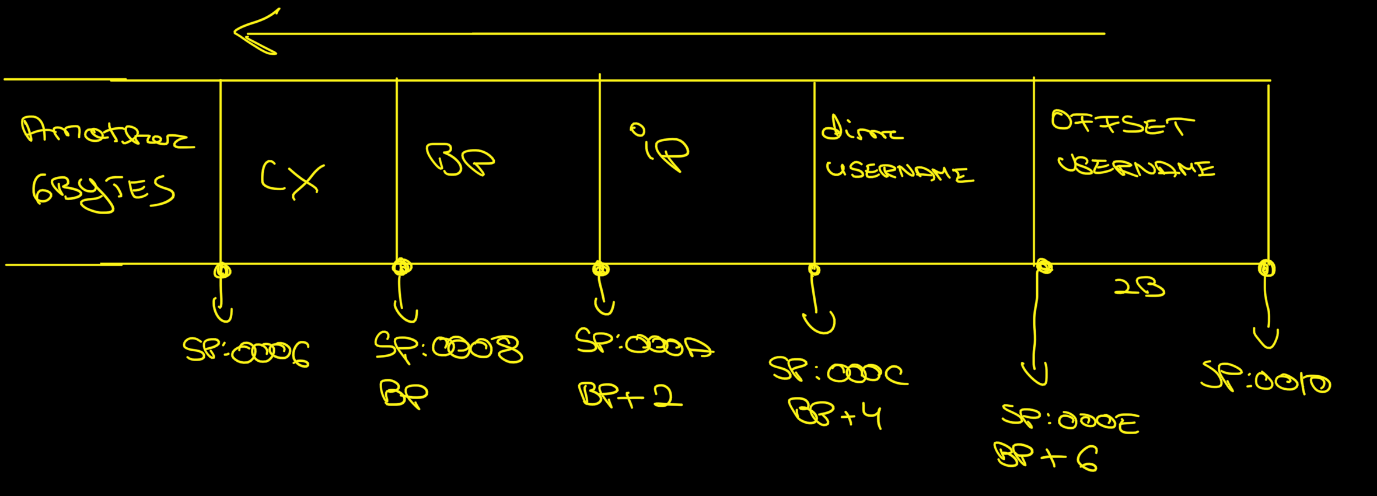
**Our solution explained:**

We will explain the 2 procedures we used (which are 2 encryption/decryption algorithms) and how we integrated them into the CSpawn virus. Let’s get started.

**The first procedure** we use to both encrypt and decrypt the hostname is called “proc.asm”. We use a pretty simple algorithm that consists of XORing each letter in the host name with 3. This algorithm is illustrated in the below picture for the letter H which, after the encryption algorithm, is converted to letter K.

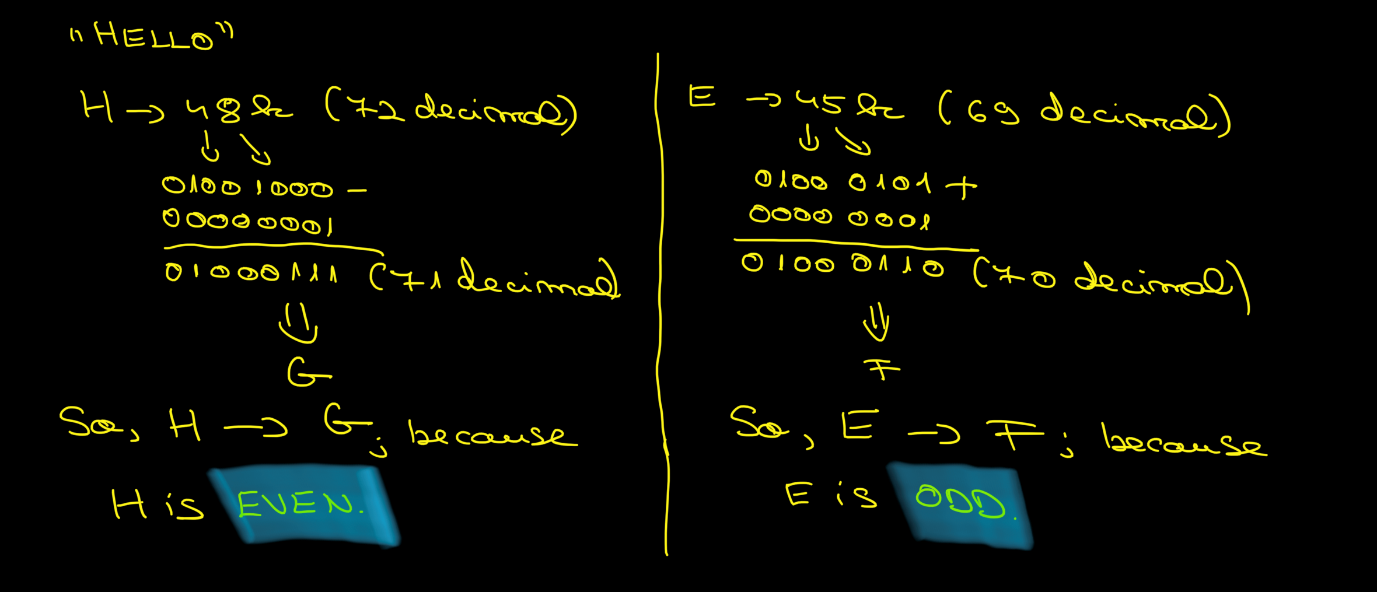


We apply this algorithm for all the other characters. A stack is used in order to successfully run this algorithm. In the below picture is drawn the stack.

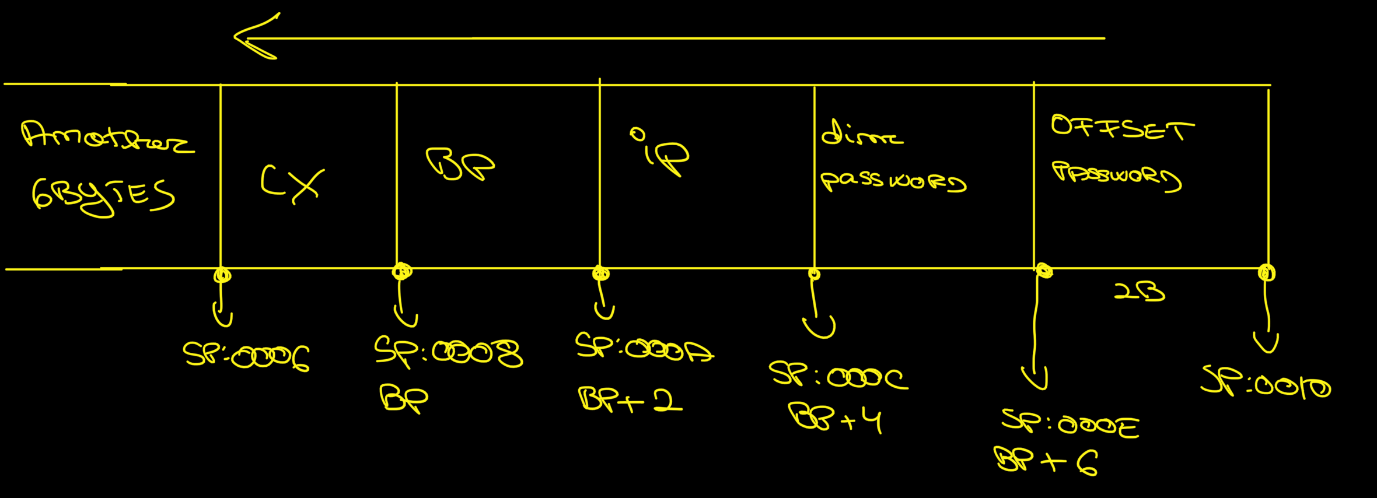


In order to decrypt, we apply exactly the same algorithm, because if a letter is XORed 2 times, the result is the initial letter (H XOR 3 = K; K XOR 3 = H).

**The second procedure** we use to both encrypt and decrypt the password is called “proc2.asm”. We use a pretty simple algorithm that consists of either adding or subtracting 1b (1 in binary - 0001) depending on the “parity” of each letter. If the letter is even, we add 1, otherwise, we subtract 1. The parity of a letter is given by the letter’s representation in binary. A binary number is odd if the lowest bit is set (for instance: 73 – 0100 1001, the last bit is 1 so it is odd, 72 – 0100 1000, the last bit is 0 so it is even). In the below imagine, this algorithm is illustrated for the letter H (which is even because H – 72 decimal – 0100 1000) and for letter E (which is odd because E – 69 decimal – 0100 0101).



We use *test instruction*1 (test AL, 1b) which checks whether the lowest bit of AL is set.[[1]](#footnote-1) If it is, the number is odd. Otherwise, it is even. Afterwards, we use a *jcc instruction*2 (JZ – j[[2]](#footnote-2)ump if even) in order to either add or substract 1.

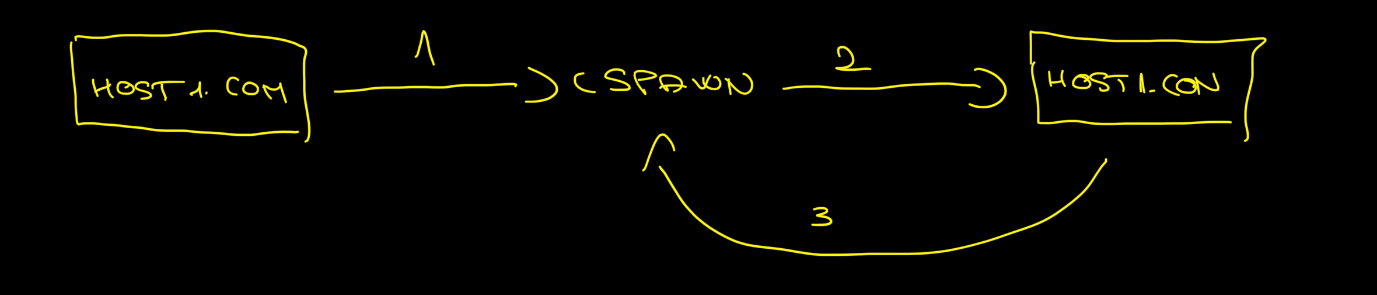
We apply this algorithm for all the other characters. A stack is used in order to successfully run this algorithm. In the below picture is drawn the stack. 

In order to decrypt, we apply exactly the same algorithm: if a letter is even, after running the algorithm, it becomes odd and vice-versa. If the algorithm is run again, then the actual letter becomes the initial one (H – 1b = G; G + 1b = H).

**CSpawn**

As we know, this is a companion virus. It renames the initial program, then creates a copy of itself (virus) with the original program name in order to fool the user who will think he runs the “normal” program.

It is important to emphasize a few things from the start. The user wants to execute “host1.com”. So, the user types host1 or host1.com, not knowing they are starting the virus. So, the virus, CSpawn, is being executed. But it cannot execute for too long because it might produce huge delays and the user will become suspicious. To avoid this, the virus will execute the initial program, which is now called host1.con. After launching the initial program, the control is given back to the virus. In the picture below, there is a representation of I explained above.



There are two key moments. One of them is the moment arrow no. 2 is taking place and the other one is when arrow no. 3 is taking place. These two moments are crucial, because it is important to realize that the virus must know what program (the initial one which had been renamed) to execute so that the user doesn’t suspect anything.

**Encryption.** We asked ourselves “when it is the moment the virus writes on the disk (writes in the host1.com file which is the virus) the name of the (initial – which had been renamed) program that needs to be executed so that the user doesn’t notice anything unusual?”

Text, table

Description automatically generated

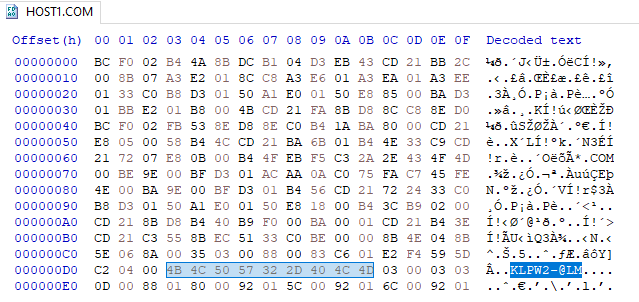
This is what we are trying to find out. Exactly the moment the virus writes the “host1.con” which is the initial program. So, we figured out that the moment we need to run the encryption algorithm is right after renaming the initial file and before creating the new hidden file (host1.com). We thought that this is the moment because we are encrypting the hostname (in memory) before actually creating the new hidden file. So, we apply the XOR encryption algorithm (which is the “proc.asm”).

After encryption, we can clearly see that “HOST1.CON” had been converted to “KLPW2-@LM”.

Table

Description automatically generated

Decryption works in a similar way. We asked ourselves “when it is the moment the virus wants to know the name of the file that needs to be executed so that the user doesn’t notice anything suspicious?”



We are looking for this moment. The moment when the virus needs to read this.

It is important to have in mind that the virus knows where to find the name of program that it needs to execute. He knows where it is located inside itself. So, the virus goes to the location where the name of the program that needs to be executed is, it extracts the name, not knowing that it was encrypted. So, right now, the virus knows that it needs to run “KLPW2-@LM”.

We must find the exact time to decrypt this, in memory, exactly before it is requested to executed. And this moment is before loading into DX register the name of the program that needs to be executed. So, this is when we apply the decryption. It is the same XOR algorithm used previously. It will decrypt in memory, but on the disk, it will remain encrypted. We only decrypt it when we need to use it.

1. <https://www.felixcloutier.com/x86/test> [↑](#footnote-ref-1)
2. <https://www.felixcloutier.com/x86/jcc> [↑](#footnote-ref-2)