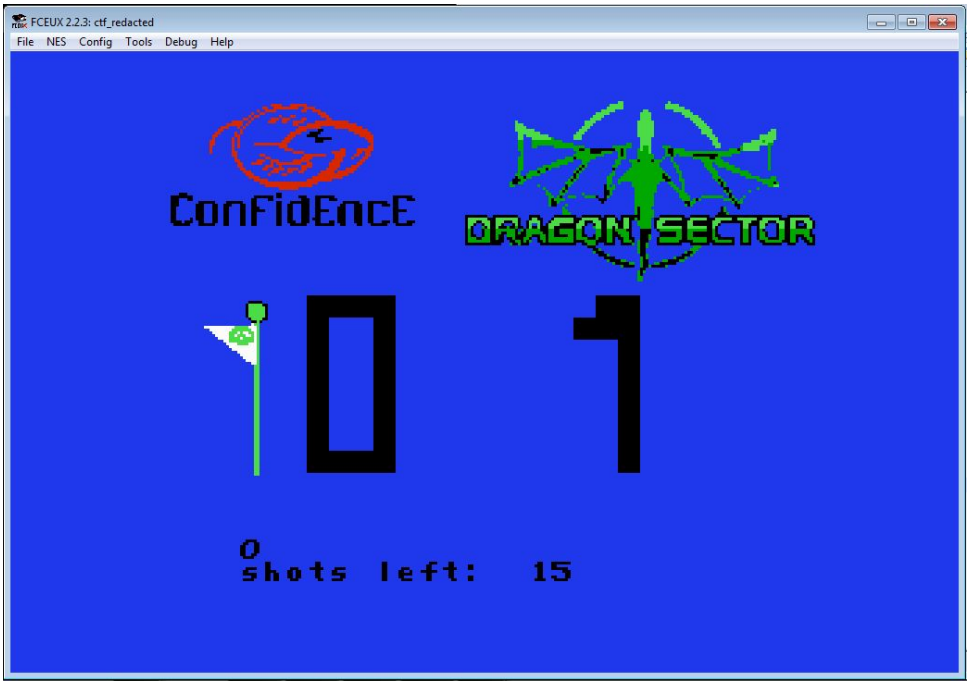
This task was prepared for CTF competition that took place during Confidence 2017 conference. To get the flag we need crack a password in a game running on a Pegasus-like console (Pegasus is the Polish name for Famicom/NES clones). The game is very simple. You can see a flag moving between the left and right edge of the screen.To enter password you must shoot the flag moving between edges. The game accepts digit that is cover by flag during shooting as a next character of password (this game needs a Zapper gun). When you type a correct password (16 bit length) you will see the flag.



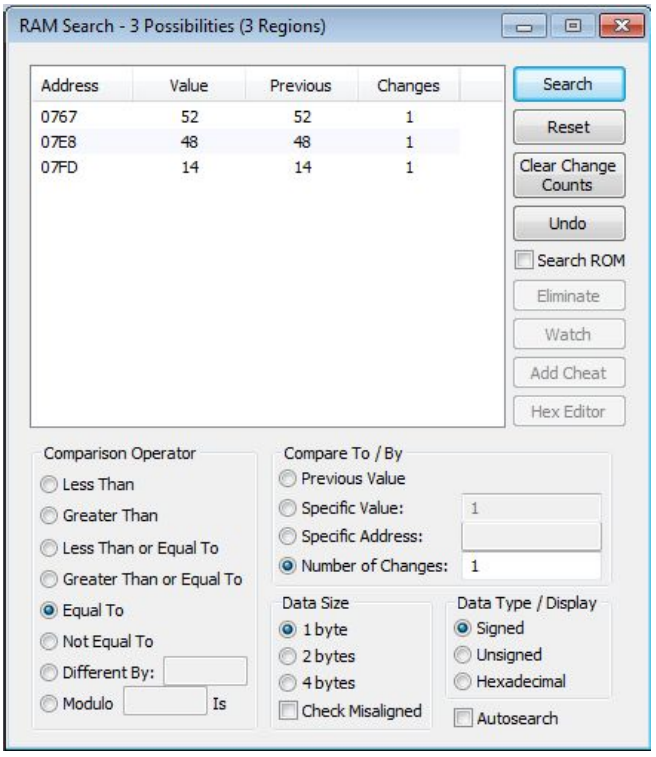
For players was prepared *.nes file but other people had possibility to play the game on a real console connected to a TV. The ROM with the game was available only on website for players so I decided to "stole" the ROM during bodacious action ;-) (fast registration just to download this one file).

Few days after end of the conference I’ve started analyzing the game at home. As an emulator I've chosen FCEUX. I'm recommending version for Windows because version for Linux is poorer (no debugger for example). What I want achieve? I want create script that will brute force the password. Probably that was faster than learning about programming for Famicom.

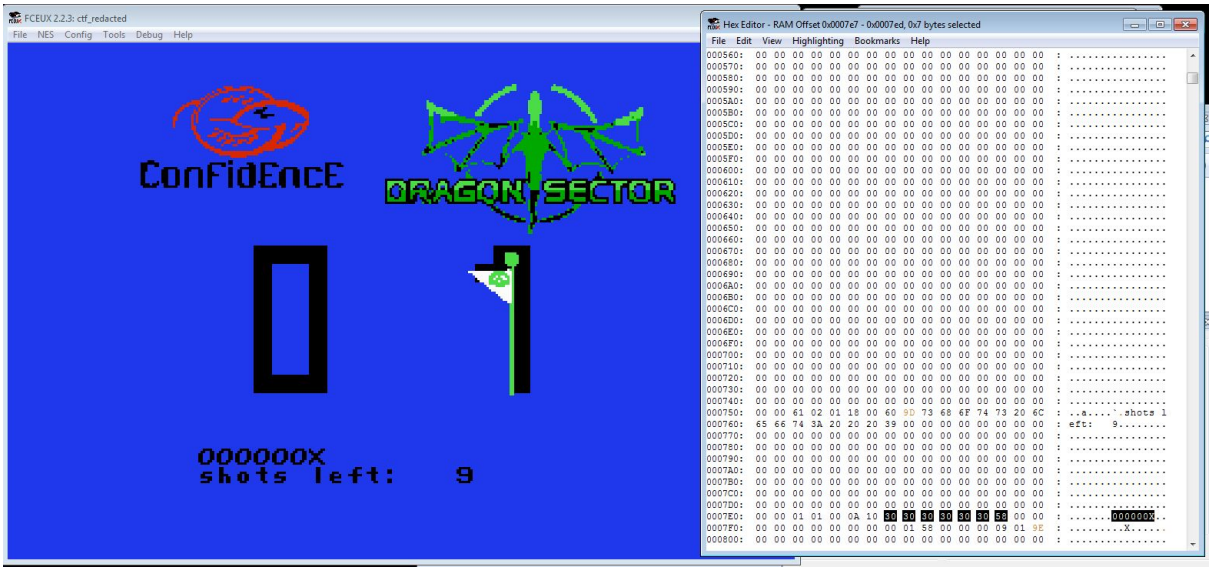
First step was to found way to control the game by changing values in a memory.

Where and how are shots store?

The shots counter must be stored somewhere but at this moment I don't know anything about format of this data. Good idea was to use RAM Search (Tools menu). With this window I can filter data by number of changes.



I've assumed that place where new shot will be saved will be written only once. I set criterios of filtering and I've shoot once. After first search tool has found few values that I had to remember. After reset searching results I've shoot one more time and I searched values again. Comparison between these two shots shows very interesting address which is incrementing with every shoot. Let's see this place in Hex Editor (*Debug* menu).



It looks like shots are just table with ASCII characters. Also few bytes later (0x7f9) we can find very interesting variable that remembers last shoot (later it will be very important). Address 0x7fd is also very interesting. It decrements with every next shot. This value tells us how many shots left.

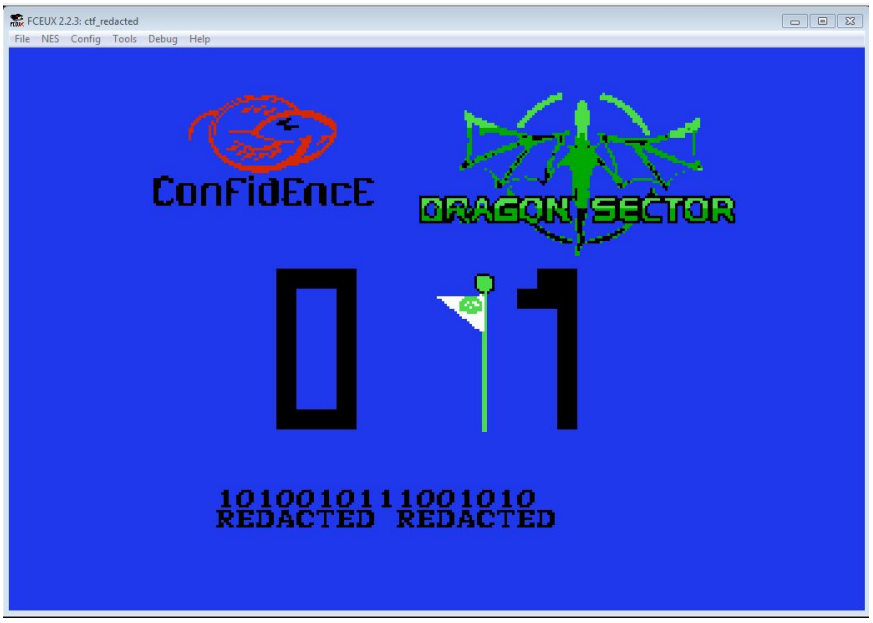
At this moment I know where are the most important values, but how to run function inside the game that will analyze the password? Hex editor shows data in real time. I've realized that when I'm pressing mouse button (Zapper's trigger in the emulator) value at address 0x7e5 is changing. When this value hits zero the game is running shoot procedure (screen blinking that probably you know from games like *Duck Hunt* or *Wild Gunman*).

16th bit of the password was problematic. You see with final shot, last bit of the password is overwriting by value from buffer at 0x7f9. As I said this is some kind of buffer that keeps last shoot value. By using LUA API I can't control Cheat/Freeze feature of the emulator, but still I can do it manually. Disadvantage of this solution is that at single instance of the script I must choose which values I want check: odd or even. To switch between one of these groups of values I must change cheat setting. Cheat tool in the emulator can be found at Tools menu. To set cheat you must type *Addr: 07f9, Val 30* and press *Add*. Now buffer is frozen and as a result of shooting procedure game will always get 0.

How check that password is correct? At 0x759 is buffer with answer that we can see in the screen after shoot 16th bit of the password. I've assumed that correct answer starts from different character than "s" or "W".

Full script and NES file are available here. Shorts comments for the script. *Pass2bytes* function converts number with a new password to ASCII table that represents data in binary format. Unfortunately LUA 5.1 hasn't binary operations like shifting, so I implemented simple algorithm that is know from CS lessons to convert decimal value to binary format. I've created *getResult* function to get first letter of the answer from the game and check that the password is correct. If first letter of the answer is different than "s" or "W" the script will end execution. Very important is function *emu.frameadvance()* from API. This function tells to the emulator to calculates next frame of the game. Without it the emulator will hangs on during waiting for new value inside loop. At the beginning I've tried reset game with next password but this method had one main disadvantage. After restart I must wait for end of game loading. Loading previous state of the emulator is bit faster. This is reason why before main loop the script was saving state of the game. Saved state is loaded at the end of the main loop.

To check even passwords set PARITY variable to 0 and Val in cheat setting to 0x30. For odd values set 1 and 0x31.



About the author

This article was created by Artur “Lacky” Łącki. If you have any questions just send me an email to address alacki93@gmail.com.