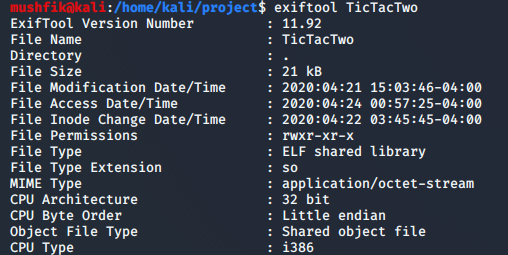
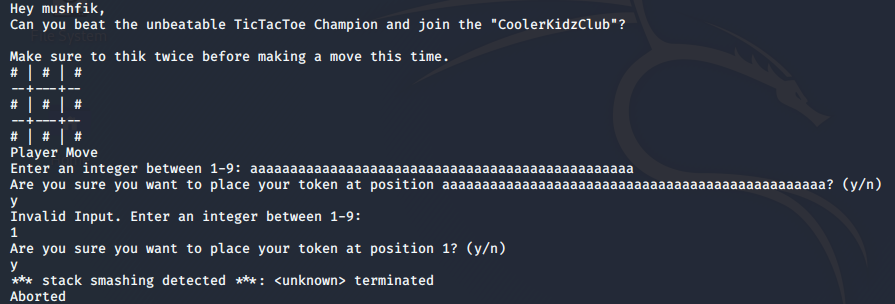
In the higher difficulty of the tutorial, we have the same challenge, to beat the same TicTacToe AI on the program TicTacTwo.

We can examine the binary with the command ‘exiftool TicTacTwo’ which tells us that the program is compiled to 32-bit executable.

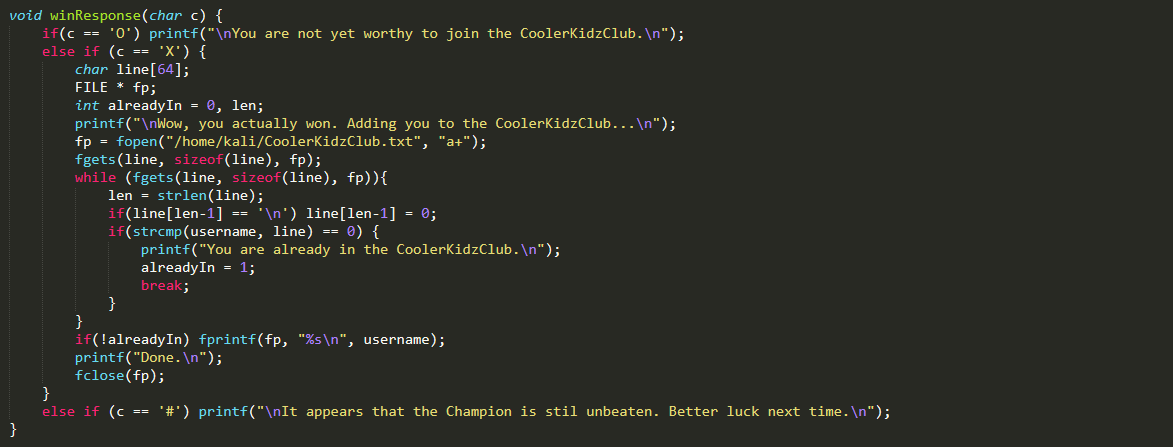


We launch the program and find some minor changes to the output. When we try to smash the buffer with a long input string, we are met with a response to confirm our input. The response simply prints back our input asking us to press ‘y’ to go ahead with the input.



After providing a valid input, the program exits but not with a segmentation fault. Instead we see a message “stack smashing detected”, which indicates that the binary has canary stack protection enabled.

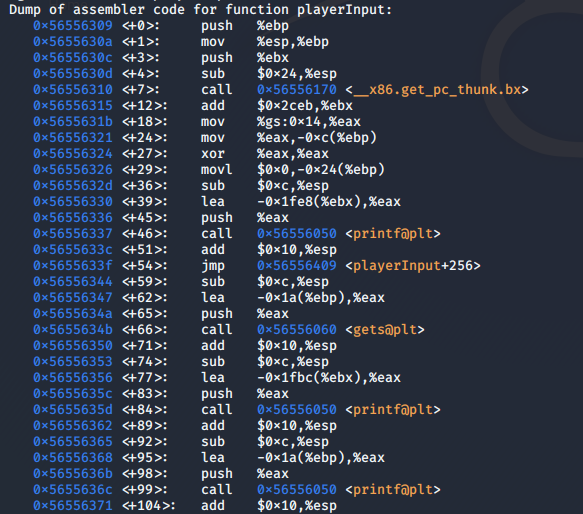
After opening the source code ‘TicTacTwo.c’, we look for the playerInput function and the playerWins function. We find that the playerWins function has been replaced with a ‘winResponse’ function that takes a ‘char’ as an argument and if the argument is ‘X’, it considers that player has won and writes the username to the desired file.

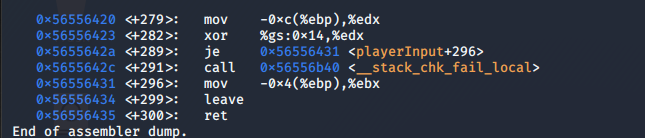


To ‘beat’ the AI this time, we need to overflow the return pointer of the playerInput fucntion with the address of winResponse function, as well as provide the argument ‘X’ on stack.

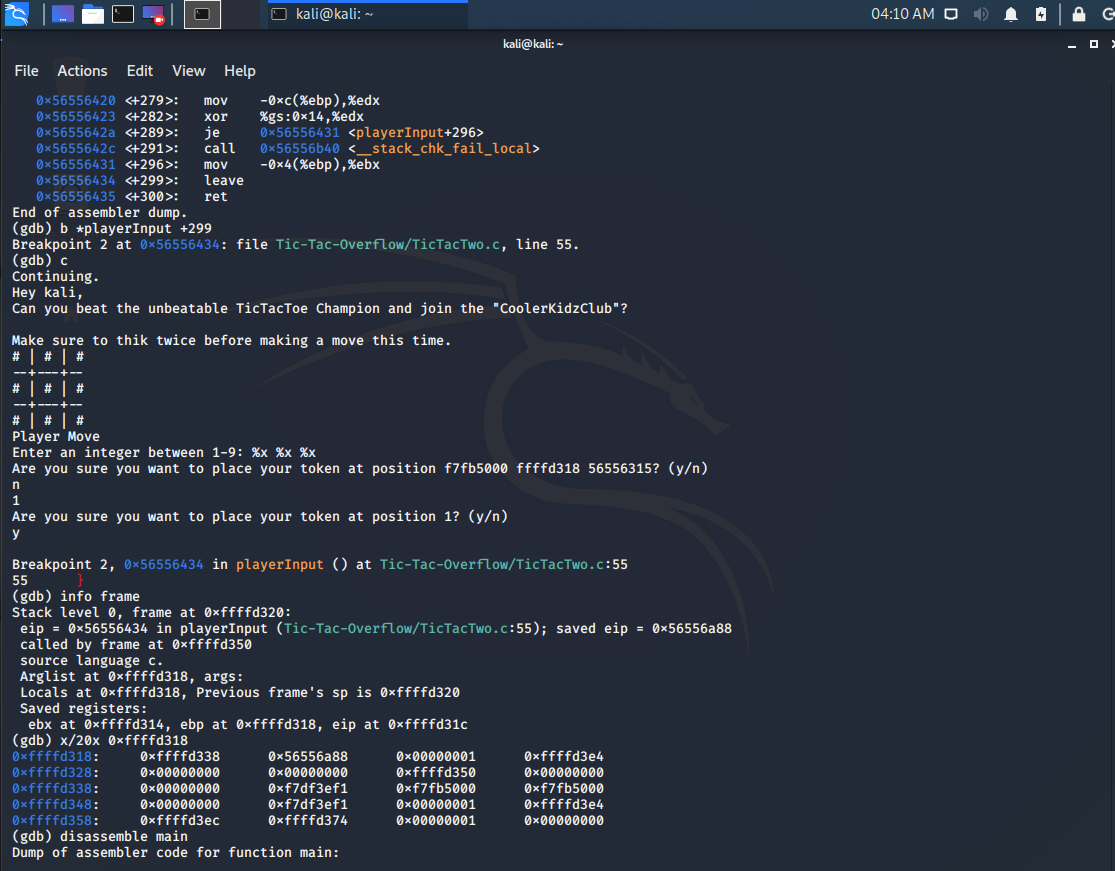
Looking into ‘playerInput’, we notice a printf function that prints out our first buffer. This usage is vulnerable to format string exploit and we can use it to leak parts of the stack, namely the value of the stack canary during runtime.

We open the binary in GDB and disassemble the ‘playerInput’ function to identify a breakpoint right before the function returns to caller which is at playerInput+299. We add the breakpoint there with the command ‘b \* playerInput+299’.



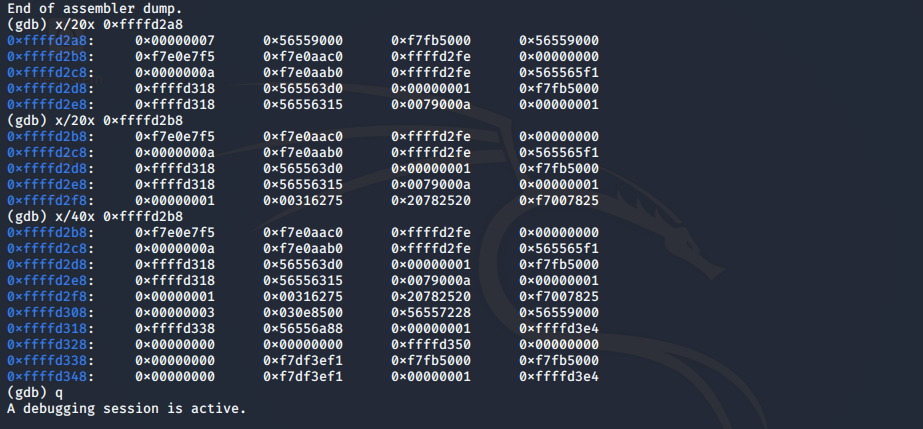


After adding the breakpoint, we continue the gdb and move forward with playing the game. When asked for an integer between 1-9, we type “%x %x %x”. This will print 3 ‘words’ from the stack frame with the following confirmation response.

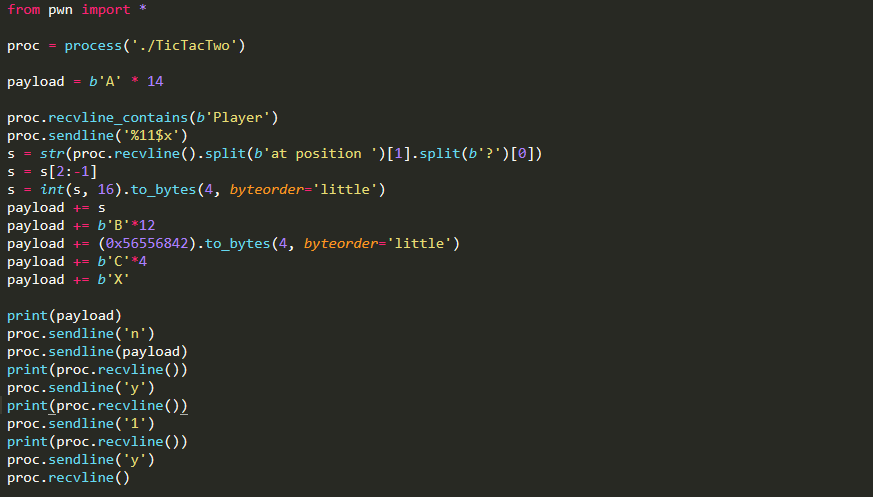


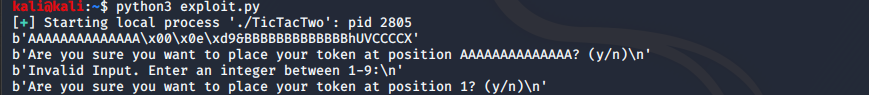
We receive a response with states that: “Are you sure you want to place your token at position f7fb5000 ffffd318 56556315? (y/n).”

We keep record of those 3 values and later supply some valid input. We reach the break point right before the function returns. We can now inspect the stack fram by typing ‘info frame’. Here we find the address of the saved eip on stack (0xffffd32c). We randomly look around to the top of the stack in order to find the position of the stack canary and the three values we leaked from the stack.

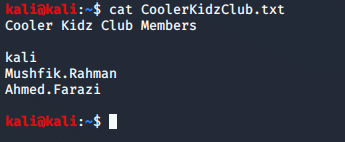


We find the three leaked values from before starting at 0xffff2d4. We look for the canary, an integer value starting with 00 near the EIP. We soon find the canary at 0xffffd31c. We calculate the word offset from the leaked values to the canary. We also calculate the offset from the buffer to canary and canary to saved EIP. If we launch the program again and use %11$x as the first input, it will leak the stack canary every time during runtime. We can write a python script using pwntools library that constructs the payload which overwrites the EIP with the address of winResponse function, does not corrupt the canary and adds the necessary agrument to the stack for the winResponse function.





After running the script, we can check the CoolerKidzClub.txt file and find that our username has been written to the file.



1. <https://zerosum0x0.blogspot.com/2016/11/overflow-exploit-pattern-generator.html>