The first thing I did, besides creating the Git Hub repository, is begin working on a stub .dll file and the python script.

I modified our example Python script for .dll injection so that It no longer required command line parameters. If the appropriate files are in the expected locations relative to the Python script, it’ll run on its own.

**Requirement #1:**

*Changes the message displayed on an illegal move to “Not in this game.”*

I started by searching in the String section for “That move is not allowed.” But I did not find it. I launched Freecell in IDA Debugger and set a series of breakpoints until I was able to isolate the function ProcessMoveRequest(x, x, x). Under the correct circumstances, this function will eventually call the Windows MessageBeep function and MessageBoxW function and display the “That move is not allowed.” text in a message box. I discovered that the offset had an empty byte in between each character (indicates which Unicode language character set is being used). The contents of this offset get moved into the ESI register before MessageBoxW is called.

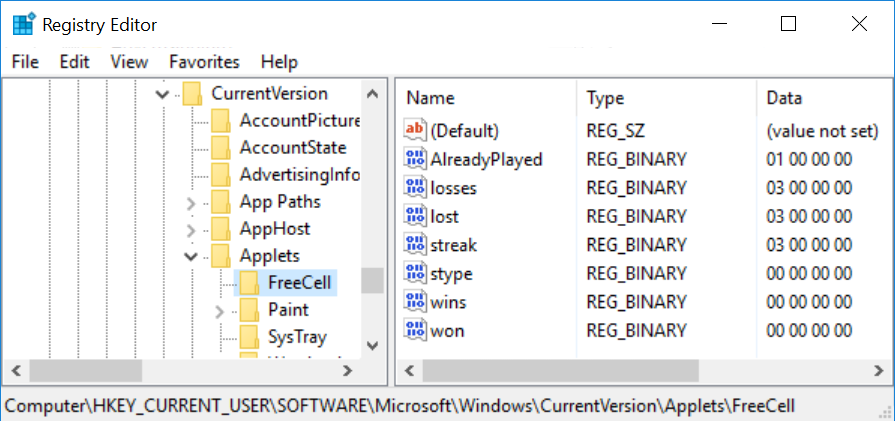
I found a tool called HxD which is another freeware hex editor. It also includes a tool for looking at the contents in RAM of a process you point it at. I used this to find the location in memory where the string is located. I was able to extract the contents of that string from memory to verify that I had located the correct location in memory.

But then I ran into another snag; my attempts to overwrite that section in memory failed. When I asked for assistance, I was reminded that the contents of RAM were protected and that I would need to use VirtualProtect() to change the access for that section on memory so that I could write to it. It took some more fiddling around - I have concluded that I am basically going to do everything incorrectly every possible way before doing it correctly – but I was eventually able to change the permissions to allow me to modify that section in memory, make the change, and then reset the permissions as before.

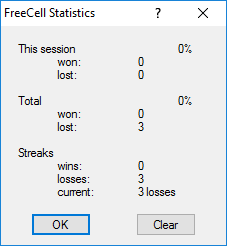
**Requirement #2:**

*Sets the total won to 1000 as shown by the Game->Statistics dialog.*

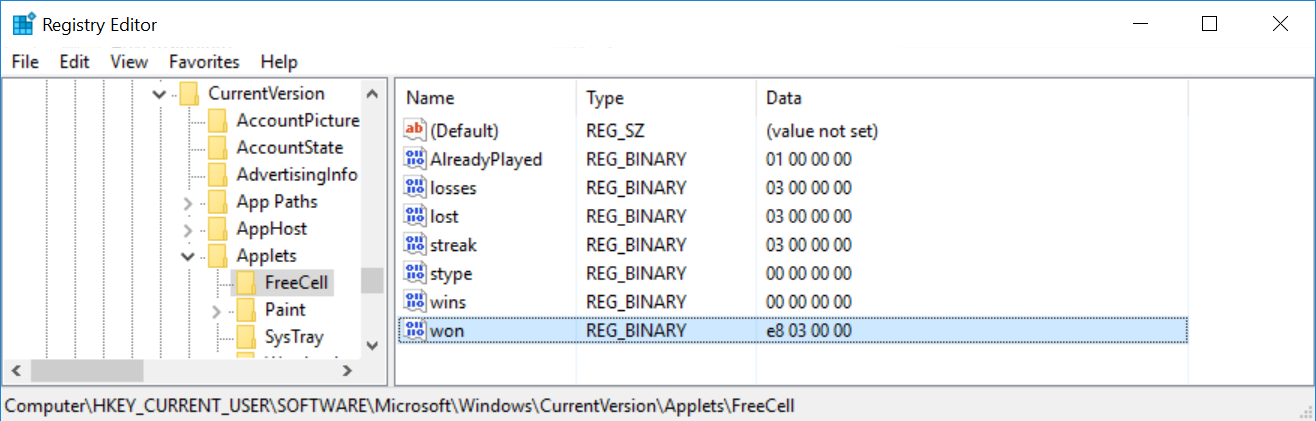
For Freecell to know how many games I have won and lost, it has to have a way to permanently store these statistics in a way that is preserved even when the freecell.exe process is no longer running. I have been writing AutoIt scripts for almost 18 years now for work. These scripts do things like automate installations of programs, create and modify registry keys that affect the behavior of programs, etc. Since I did not see any .INI files being created in the same directory as freecell.exe to store this data, I figured that it was stored in the registry. It would not make sense to store this data in HKEY Local Machine hive since that is for system-wide settings. So I launched Reg Edit and navigated to the HKEY Current User hive since that contains settings specific to my user account (i.e. would contains statistics relevant to my local Windows account). I performed a search using Control-F with the search string “freecell.exe”. I had to skip past references to IDA pro and Visual Studio until I found the key “HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Applets\FreeCell”.

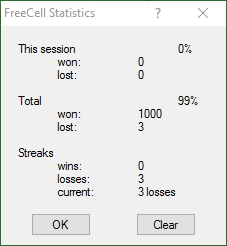


I noticed that the REG\_BINARY value names and values appeared to directly or indirectly reflect those in the FreeCell Statistics window, as shown below.



I have never worked with REG\_BINARY values before so it took a little bit of time to figure out how to manipulate the value properly. I used Windows 10’s calc.exe in Programmer mode and discovered that 1,000 in decimal is equivalent to 0x03E8. When I manually changed the key to that value in Reg Edit the FreeCell Statistics window gave the incorrect value in decimal. I eventually figured out why – I had to enter it in Little Endian format. These screenshots show the correct value and result. I was able to find example code that allowed me to write to the registry once I was able to determine the proper values for the function.





**Requirement #3:**

*The next valid move wins the game.*

Through trial and error I discovered that when you bring up the cheat window and press “Abort”, the result is that the internal variable \_bCheating is set to 2. Any time that you try to perform a move, the function ProcessMoveRequest is executed. In ProcessMoveRequest is a function called \_Cleanup which calls a function Useless. Useless checks the value of \_bCheating; if \_bCheating is equal to 2, then upon attempting to execute another move it will determine that cheat mode is enabled and will act accordingly. I was able to verify that this works by entering this Python command during debugging in IDA Pro:

PatchByte(0x01007130, 2)

NOTE: The value of \_bCheating is reset once the initial game has been won, so the next game will not have cheating enabled by default.

**Requirement #4:**

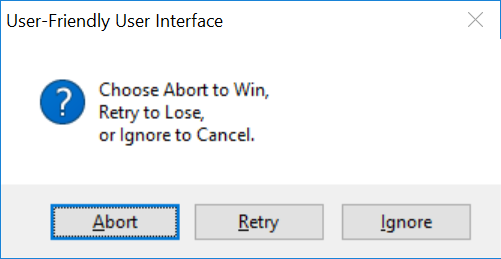
*Freecell contains a cheat code, that, when Ctrl – Shift - F10 is pressed, a dialog is presented to the user to “Abort, Retry, or Ignore”. If the user clicks “Abort”, the game is automatically won. Your DLL should change this cheat-code sequence to Ctrl - Shift – F6.*

Conor and I met multiple times on this project. We used a couple of tools to help with this one - IDA Pro Free to disassemble and debug freecell.exe, and Resource Hacker. Using Resource Hacker, we determined that there is an Accelerator table called FREEMENU which includes the keystroke combination hotkeys for freecell.exe. In the Accelerator table, the line of code which we are interested in is this one:

VK\_F10, 114, CONTROL, SHIFT, VIRTKEY

It is currently set to Control-Shift-F10 but we need to patch it to Control-Shift -F6.

While trying to isolate where in freecell.exe the accelerator tables are referenced, In IDA Pro we found a switch statement at address 0x01001EA1. This switch statement references off\_1002108 which contains a collection of 15 offsets. Each offset contains a memory location to which the program execution should continue based upon which hotkey combination was entered by the user. We identified the offset responsible for jumping to loc\_10020B9, which contains the instructions for displaying the following window:



After a lot of discussion and experimentation, we decided to find the location of the accelerator table in memory and patch it to use Control-Shift-F6 instead of Control-Shift-F10. By setting a breakpoint at 0x0100225F and launching IDA in Debugger mode, we learned that the Accelerator table is loaded into memory during initialization of freecell.exe. We are not permitted to modify the freecell.exe binary so we will patch the freecell.exe process.

We were unable to patch the existing accelerator table in memory, so we tried to destroy the existing FREEMENU, create a new one, and then load the new one. We were unable to destroy the existing accelerator table (it always returned FALSE), even after trying to use VirtualProtect to modify the permissions.

I created a copy of the FREEMENU table ACCEL entries and placed them into an ACCEL array. The array was one cell larger than that of FREEMENU (for a total of 9 cells). In that empty cell I added a new ACCEL object with the Control-Shift-F2 code for requirement #5.

Looking at the last set of notes from the final class, the example code explains that we should use TranslateAcceleratorW(hWnd, hAccel, lpMsg) to load our custom accelerator table. It looks like that is necessary to map the new accelerator table to the window. This requires access to the Msg variable, which I am having a difficult time figuring out how to access.

I created a custom accelerator table and I found what I believe to be the HWND for the main window (0x01008374) but when I tried to use GetMsg() it crashed freecell.exe .

**Requirement #5:**

*Ctrl-Shift-F2 wins the game.*

For this to work, we need to add another entry to the accelerator table which will handle the hotkey combination Control-Shift-F2. We also need to modify the switch statement located at address 0x01001EA1 so that it knows to which memory address to jump when Control-Shift-F2 is input. The action to perform will be patching \_bCheating (0x01007130) to the value 2.

.text:01001E91 movzx eax, word ptr [ebp+wParam]

.text:01001E95 add eax, 0FFFFFF9Bh ; switch 15 cases

.text:01001E98 cmp eax, 0Eh

.text:01001E9B ja loc\_10020F1 ; default

.text:01001EA1 jmp ds:off\_1002108[eax\*4] ; switch jump

At 0x01001E95 we are subtracting 101 (decimal) from the contents of the eax register.

I know that at 0x01001E98 it is comparing eax to 0Eh, which is 14 in decimal. From what I can tell here, the ja operation is saying that if the contents of eax > 14 then perform the default action, which is go to loc\_10020F1. Otherwise, go to the switch table here:

.text:01001EA1 jmp ds:off\_1002108[eax\*4] ; switch jump

That switch table looks like:

.text:01002108 off\_1002108 dd offset loc\_1001EA8 ; DATA XREF: MainWndProc(x,x,x,x)+3A8r  
.text:01002108 dd offset loc\_1001F05 ; jump table for switch statement  
.text:01002108 dd offset loc\_1001F0D  
.text:01002108 dd offset loc\_10020F1  
.text:01002108 dd offset loc\_1002058  
.text:01002108 dd offset loc\_1002086  
.text:01002108 dd offset loc\_1001F0D  
.text:01002108 dd offset loc\_1001EF1  
.text:01002108 dd offset loc\_1002075  
.text:01002108 dd offset loc\_100209E  
.text:01002108 dd offset loc\_10020A4  
.text:01002108 dd offset loc\_10020F1  
.text:01002108 dd offset loc\_10020F1  
.text:01002108 dd offset loc\_10020B9  
.text:01002108 dd offset loc\_10020AF

Those offsets correspond to the locations in memory that execution will continue at based on the accelerator table keystrokes. My hunch is that we would want to patch:

cmp eax, 0Eh   
to  
cmp eax, 0Fh

to allow the addition of the extra accelerator command Control-Shift-F2. Then we would add an additional offset above, something like:

.text:01002108 dd offset 010020D3

Which would take us to this line of code, which enables cheating mode (next move wins game):

.text:010020D3 mov \_bCheating, 2 ; If we are here, then we pressed "Abort"