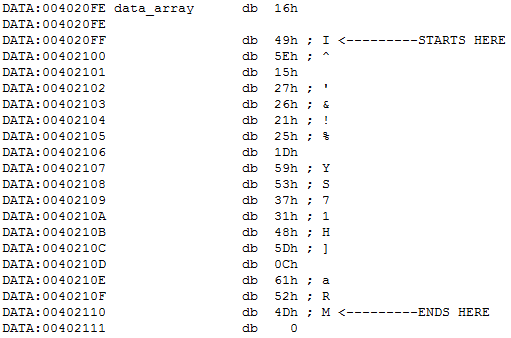
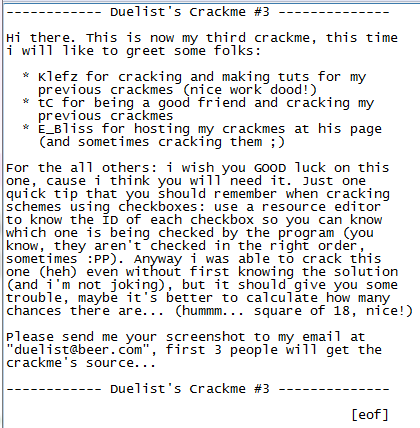
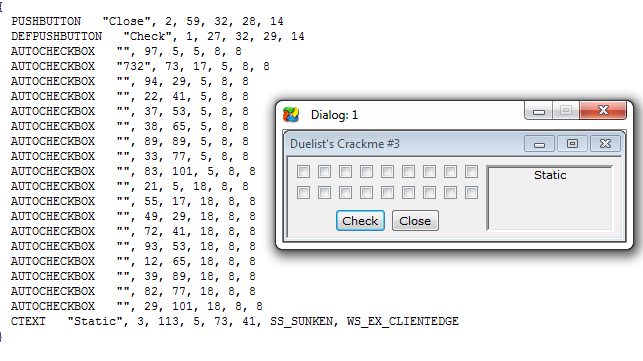
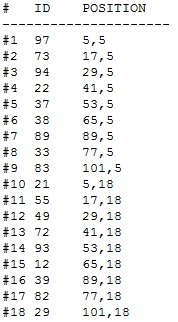
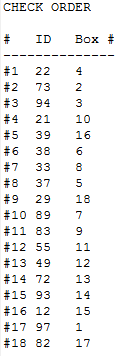
Cracking Duelist #3

1. First take a test run of due-cm3.exe. You will notice that in this particular crackme a series of checkboxes represent the registry key. In this case, you must select the correct check boxes to register the program.
2. Go ahead and open due-cm3.exe up in IDA Pro.
3. Go to the entry point of the program.
4. It may appear at first that nothing is called other than the program’s preamble and DialogBoxParam. However, if you look closely, you will see that as an argument to DialogBoxParam is DialogFunc. This is a pointer to user defined code that is passed as an argument to DialogBoxParam.
5. Go to DialogFunc.
6. The strategy I took in this case was to work backwards from the congratulatory message. In this case the congratulatory message is “Congratulations! Please send a screenshot of your solution to duelist@beer.com!” You can see that the only way to get to the congratulatory message is for the value above in EAX to be 0F35466h.
7. You will also notice that before EAX is compared it is multiplied by 4D. In essence this means that we need EAX = 0F35466/4D = 328FE.
8. Now, if we backtrack a little further we see that the only way to reach the block we were just at is for ECX to have the value 4Dh. Directly above that we see that ECX is an array indexed by ESI. Look at the data inside of the array. We see that the data is the following:
   1. 
9. In the block that led to the current block we see that it zeroes out ESI, EDX, and two data blocks.
10. Now, go back to the block we were at in step 8. If ECX isn’t 4Dh we go to a different block. We see that in this block a call to IsDlgButtonChecked is made. We also see that ECX contains the identifier for the checkbox. That identifier is stored in the same data block that was zeroed out earlier.
11. We see that after the call ESI, which we now know is a counter, is incremented. If the checkbox was checked we then move on to the next block. Otherwise, we loop around to the block from step #8 and repeat.
12. If the checkbox is checked we see that the resource id is moved into EAX and an element from the data array is moved into ECX, indexed by our counter variable in ESI.
13. The following then happens
    1. resource\_id x data\_array\_element x counter + total\_sum = total\_sum
14. This process repeats until the indexed data\_array element is 4Dh.
15. The following is Duelist’s comments to us:
    1. 
16. You will notice that as of now we have no way to figure out which checkboxes have what resource identifiers. In my case I installed a program called PE Explorer which allows me to see exactly that. Here is a screenshot of the resource identifiers. 
17. Here is a simplified version: 
18. Now Duelist also told us that the check boxes aren’t necessarily checked in the right order. Well, I ended up running due-cm3 through a immunity debugger and he’s right. They are checked in the following order: 
19. Well great, now we know what order they are checked in but there’s still the problem of there being 2^18 possible combinations that might add up to 328FEh. So how do we figure out which one is right? Well, in my case, I wrote a ruby script to brute force the problem.