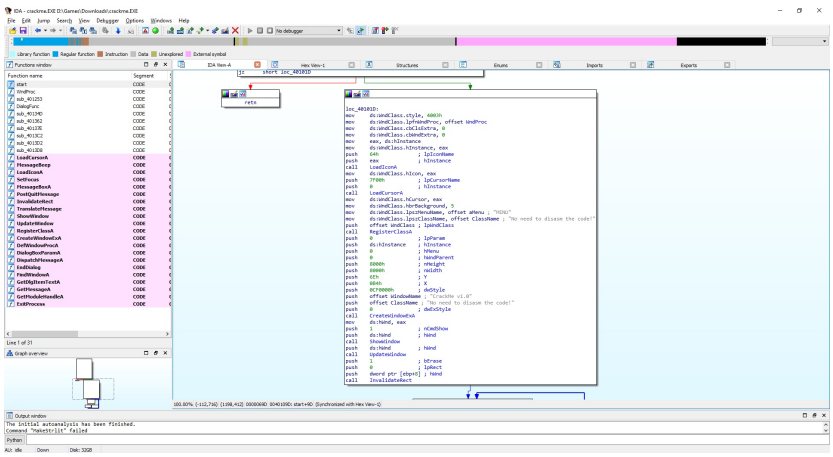
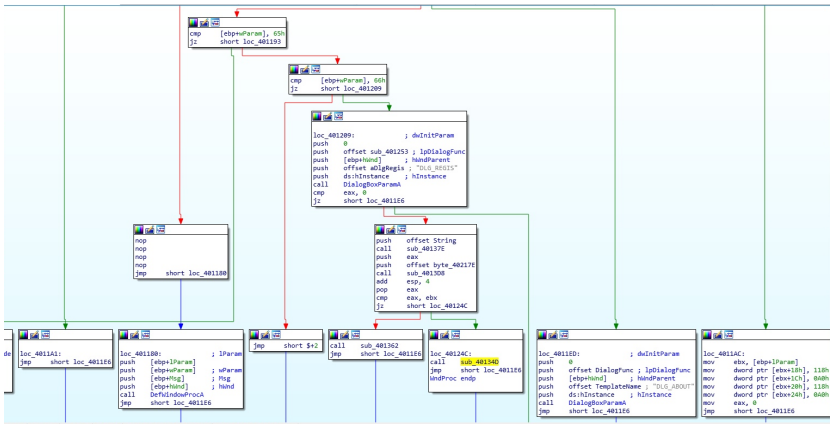
Решение CrackMe

Первым делом скачаем наш файл и откроем его в каком-нибудь дизассемблере, например, IDA:



Сперва глянем на все функции, которые есть, их не так много, если убрать те, которые нужны для нормального функционирования оконного приложения:

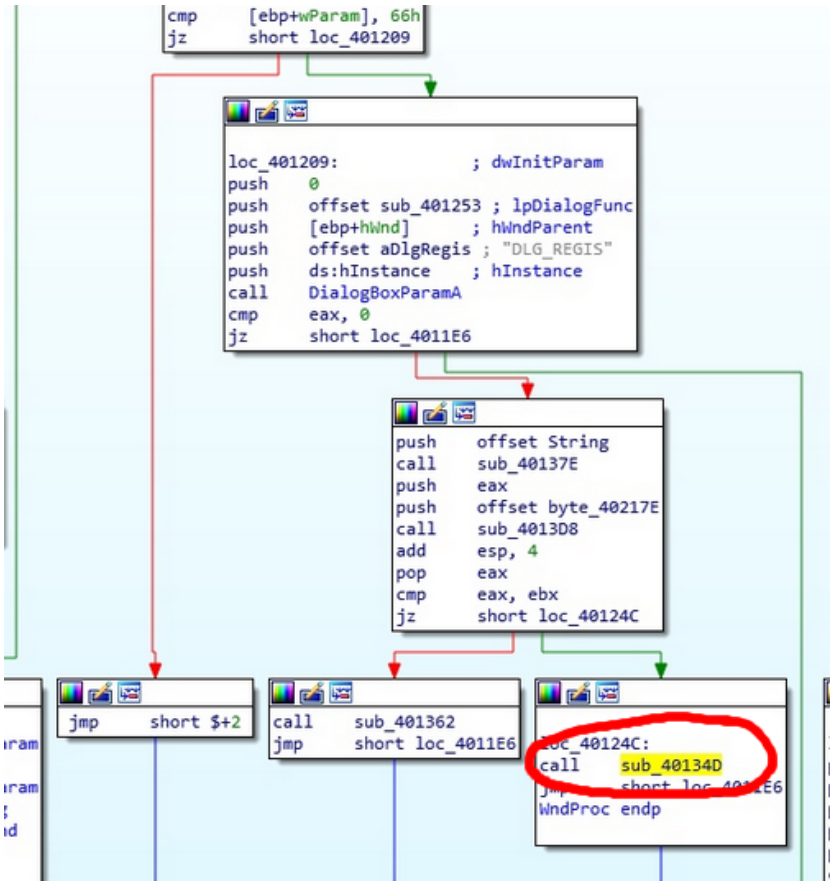
Сам приложение начинается с функции start, которая уже вызывает WindProc. Сама WindProc выглядит довольно громоздко и с кучей условий (это лишь часть):



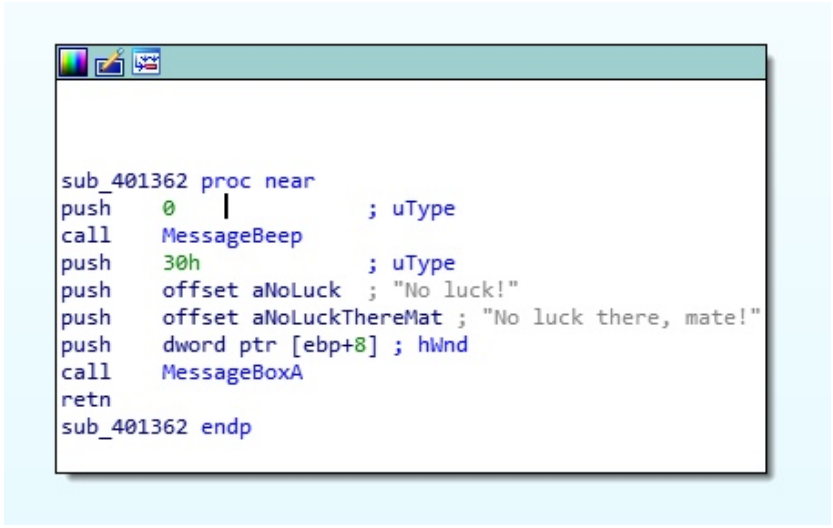
Пооткрывав другие функции, мы можем обнаружить одну с интересной строчкой:

```
sub_40134D proc near
push    30h             ; uType
push    offset Caption  ; "Good work!"
push    offset Text     ; "Great work, mate!\rNow try the next Cra"...
push    dword ptr [ebp+8] ; hWnd
call    MessageBoxA
retn
sub_40134D endp
```

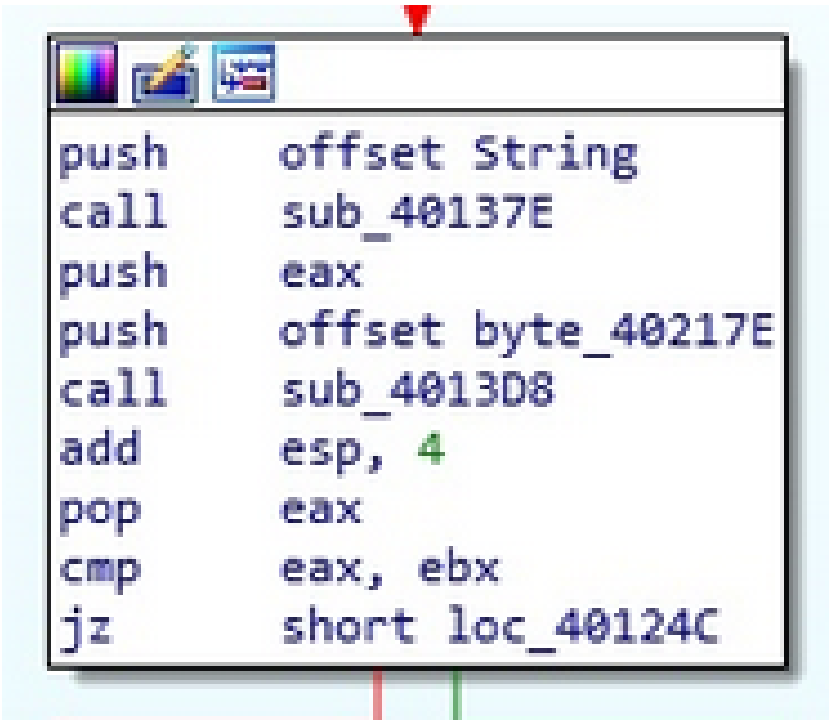
Хм, похоже на ту функцию, которую хотелось бы вызвать, так как она явно говорит о хорошо проделанной работе. Что ж, перейдём в то место, где она вызывается:



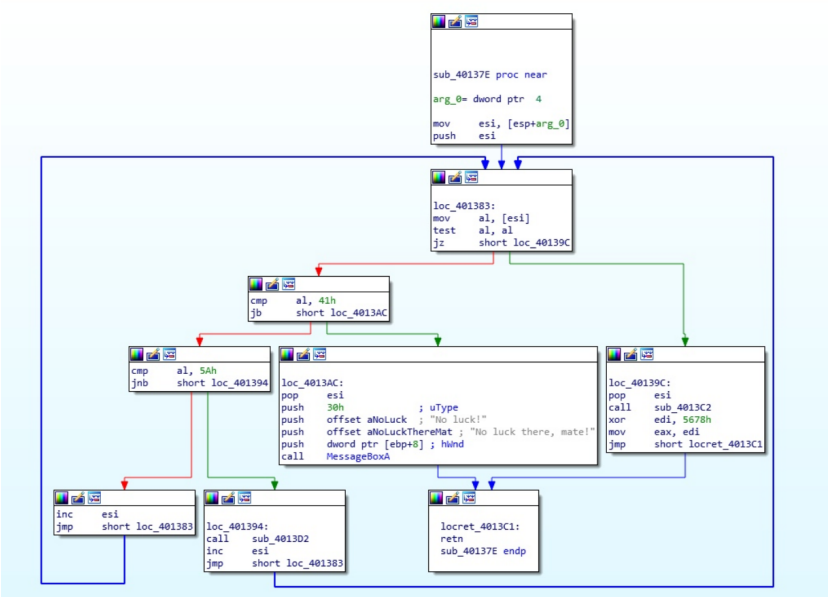
Переименуем её тогда в success. Прямо перед этим стоит проверка на равенство регистров eax и ebx. В случае их неравенства вызывается левая функция sub_401362. Глянем на неё:



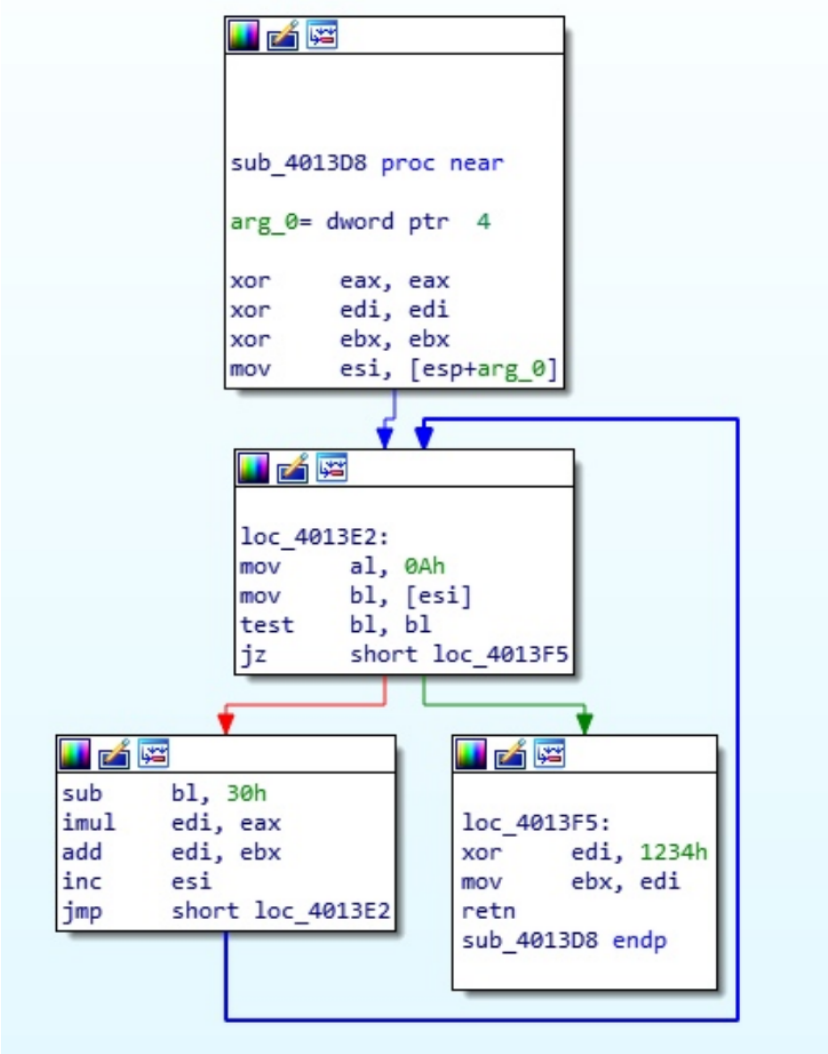
А эта похожа на неудачу. Ровно такой же текст выводится, если ввести случайные логи и пароль. Значит условие равенства eax и ebx это то, что нужно выполнить. Что ж, переименуем соответствующие функции и для наглядности раскрасим в понятные цвета. Теперь посмотрим на блок кода, который идет перед сравниванием ebx и eax:



После вызова sub_40137E eax закидывается в стек, откуда вытаскивается только перед самим сравнением. Значит, последнее изменение с ним могло произойти только в sub_40137E. Вот собственно и она:



Итак, тут цикл, проходящийся по элементам стека, который работает, пока не встретит «
0», т.е. конец строки. Далее, если вдруг текущий элемент со стека меньше, чем 0x41 (что в таблице ASCCI соответствует символу «A»), то сразу вызовет функцию с выводом «No luck there, mate!». Таким образом, вот первое условие — все символы логина должны идти строго после «A» в таблбице ASCII. Далее, если все-таки текущий символ больше или равен «A», идет сравнивание его с 0x5A (что соответствует символу «Z»), и в случае, если текущий символ больше, вызывается функция, которая просто уменьшает его на 0x20 (что для букв в нижнем регистре попросту вызывает uppercase()) и переходит к следующему символу. Когда же мы дошли до конца строки, вызывается функция простого суммирования всех кодов символов и с полученной суммой делается xor 0x5678, результат записывается в eax. Тогда назовем эту функцию например HashLogin. Значит первое, что мы должны сравнить, это сумма всех кодов логина поксоренная с 0x5678. Но с чем мы должны её сравнить? Давайте переёдем к следующей функции, вызывающейся перед сравнением, может в ней и записывается как раз ebx:



Анализируя функцию, видим, что она также проходится по всем символам пароля, из кода вычитается 0x30 (что эквивалентно «0»), текущая сумма умножается на 0xA (10) и прибавляется наш символ с ввычтенным числом. Когда достигнем конца строки, результрующее число мы ксорим с 0x1234, результат записывается в ebx. Итого мы получаем, что пароль можно считать просто в цифрах (потому что вычитается «0»), Это число ксорится с 0x1234 и сравнивается с хэшем, полученным от логина.

В итоге мы получаем, что программа примет логин и пароль, если сумма кодов символов логина (-0x20 в случае с символами больше «Z») xor 0x5678 xor 0x1234 должно быть равно паролю в десятичной записи. итоговый кейген лежит в файле

keygen.c