**Лекция 14. «Лапша и защита программ от взлома».**

**Замечание. В сегодняшней Лекции не будет никаких вопросов и не надо слать мне никаких ответов. Просто читаем и работаем НАД ЛАБАМИ – там вполне хватает «работы».**

**На прошлой лекции по сути дела был сформулирован ещё один постулат:**

**Постулат 5. Так как абсолютной защиты не существует, и так как «время-деньги», то первейшей задачей любого разработчика систем защиты данных, в основе которых лежит ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, является ЗАДАЧА УСЛОЖНЕНИЯ для нехорошего человека, нехороших организаций процесса исследования того или иного программного продукта.**

**На прошлой Лекции был представлен один из возможных вариантов подобного усложнения – это определение того, анализируется, находится ли в данный момент времени данная программа «под отладчиком», и удаление соответствующего ей исполнимого модуля в том случае, если такое имеет место быть.**

**Ещё один способ «лапши» - «запутывание редиски – нехорошего человека» предполагает сокрытие пароля непосредственно в тексте программы.**

**Комбинирование этих двух способов «вешания лапши на уши» нехорошему человеку увеличивает вероятность того, что Ваш исполнимый модуль НЕ будет удален, так как: а) Очевидно, что простой взгляд на исполнимый модуль не позволит «увидеть» напрямую пароль – потребуется отладка – будь то отладка с целью «выщемить» пароль, или будь это отладка с целью обойти ветвь проверки пароля и напрямую перейти к функционалу программы; б) А как только будет сделана попытка осуществить «отладку», это приведёт к уничтожению исполнимого модуля.**

**Понятно, что среди Вас есть отличники, есть двоечники, есть супердвоечники. Для двух последних категорий далее приводится фрагмент исходного текста на языке записи алгоритмов ассемблер, в котором как раз делается попытка «спрятать» пароль. Посмотрите, может пригодится в жизни, хотя вряд ли…**

.model tiny

.data

input db 5,?,5 dup (?)

errmsg db 'FAIL',10,13,'$'

scmsg db 'SUCCESS',10,13,'$'

.code

.startup

main proc

; password

mov bx,15Fh

call todigit

push ax

mov bx,120h

call todigit

push ax

mov bx,15Ah

call todigit

push ax

mov bx,166h

call todigit

push ax

; input

mov ah,10

lea dx,input

int 21h

mov ah,4

cmp ah,input+1

jne miscompare

; comparison

mov bx,2

compsym:

pop ax

mov al,input[bx]

cmp ah,al

jne miscompare

inc bx

cmp bl,input+1

jne compsym

jmp success

miscompare:

lea dx,errmsg

jmp outmsg

success:

lea dx,scmsg

outmsg:

mov ah,9h

int 21h

…

mov ah,4ch

int 21h

ret

main endp

todigit proc

; BX byte offset

; returns symbol in AH

mov al,byte ptr [bx]

mov ah,0

mov dl,0Ah

div dl

xor ah,30h

ret

todigit endp

end

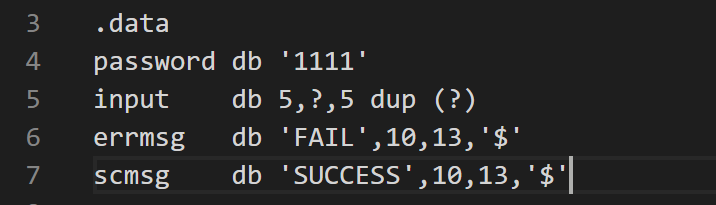
**Интересно, сколько Вам потребуется времени, чтобы определить пароль? Далее представлен текст отчёта по решению задачи упрятывания пароля.**

*Емельянова О. Ю.*

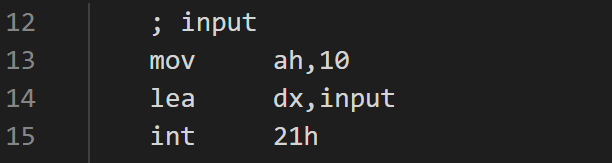
*4 курс 13 группа.*

# Лабораторное занятие

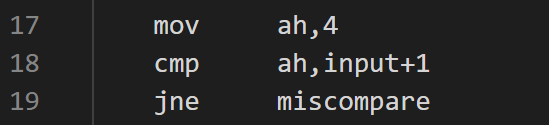
Добавим следующие данные: правильный пароль, область для ввода, сообщения об ошибке сравнения и успехе:



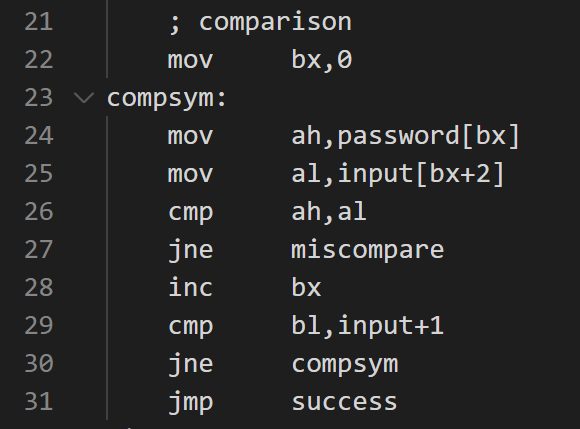
Ввод пароля пользователем:



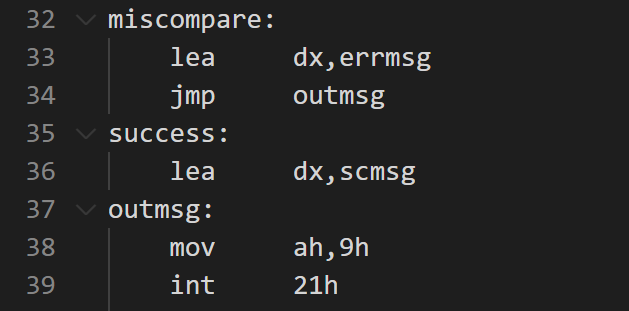
Сразу переходим к выводу сообщения об ошибке, если длина введенной строки и ожидаемая длина не совпадают:



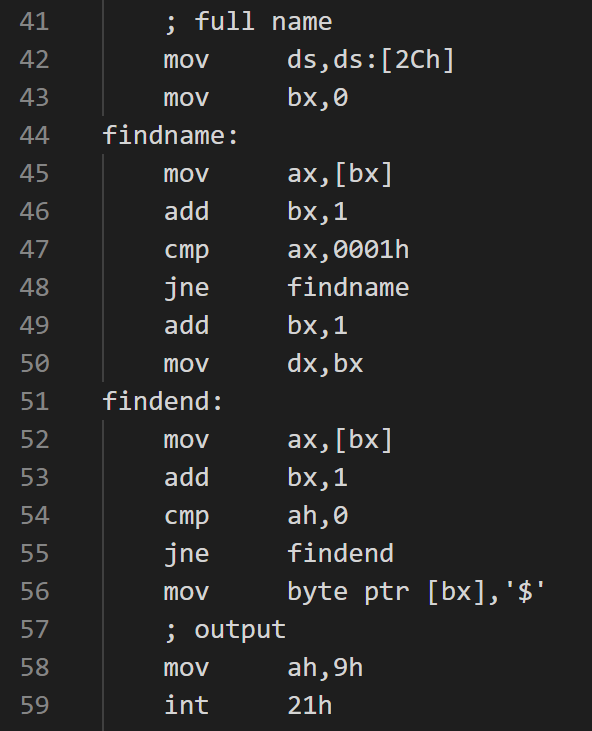
Если длины совпадают, сравниваем поочередно каждый байт. В случае ошибки переходим к выводу ошибки, иначе – к выводу сообщения об успехе:



Вывод:



Вывод полного имени:



Результат при вводе верного пароля:



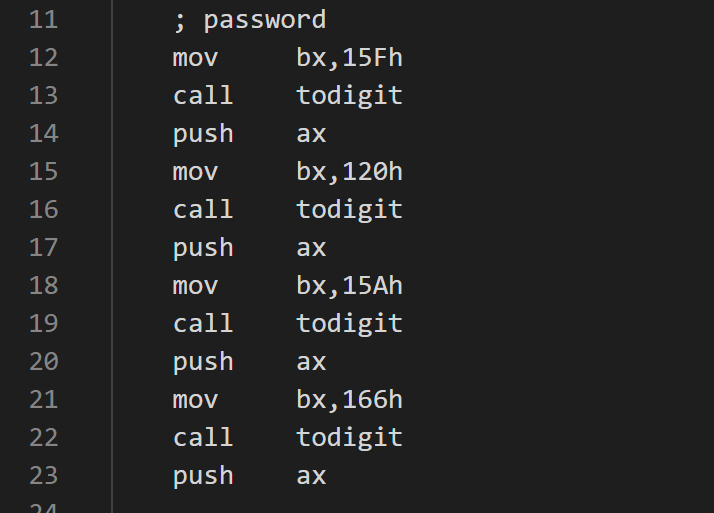


Результат при вводе неверного пароля:

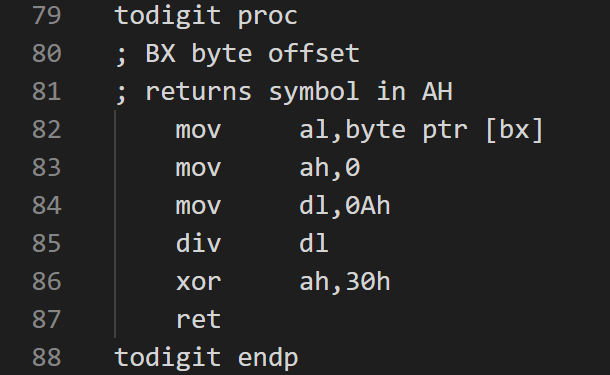




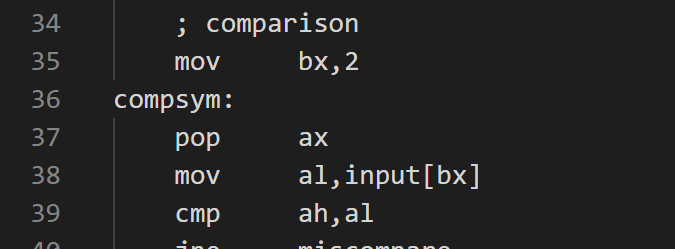
**Чтобы спрятать пароль**, в данном случае «1111», найдем 4 байта, из значений которых можно будет извлечь «1», взяв остаток от деления на 10 и добавив 30h. ***Например, это могут быть байты по адресам 15fh, 120h, 15ah, 166h (в корне неправильная фраза!!! Здесь НЕТ НИКАКИХ АДРЕСОВ!!!! – ЗДЕСЬ ТОЛЬКО КОНСТАНТЫ!!!!***



Каждый из байтов преобразуем следующим образом:

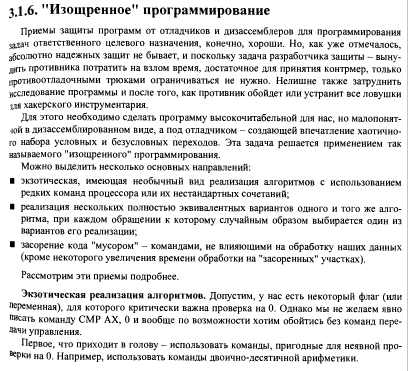


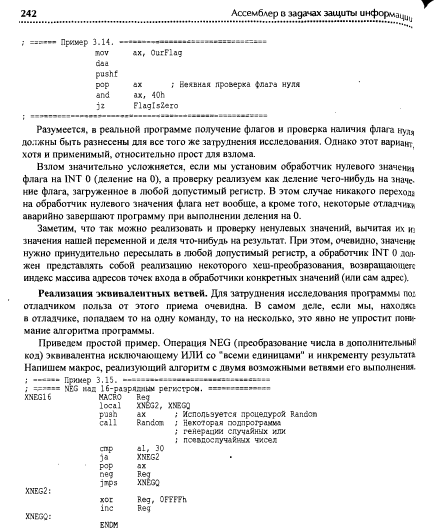
При сравнении байты будем извлекать по одному из стека (точнее, из стека достаем слова, где код символа – в верхнем байте):

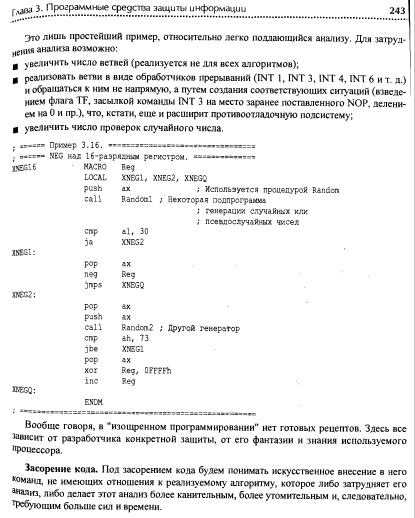


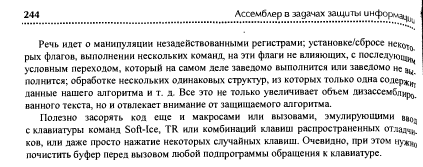
В остальном сравниваем так же.

**В книге про ассемблер представлены ещё несколько вариантов «запутывания» программ. Эти варианты в любом случае потребуют от нехорошего человека попытки запустить программу под отладчиком… В том, что представлено далее, я не совсем «догоняю», поэтому «жевать» не буду, а предложу Вам «пожевать» самостоятельно. Только сильно не увлекайтесь …**









**Ещё один способ запутывания заключается в** получении **программы, которая меняет сама себя, когда кажется, что программа вроде бы должна заставить компьютер сделать некоторую операцию, а на самом деле – в процессе уже выполнения – процессор делает совсем другое.**

**Как это сделано, например, в данном примере, на программе про 36 байтов и исполнимых модулей типа com (программа меняет сама себя таким образом, что всегда на экран выводится квадрат числа при любых значениях цифр. Не обращайте внимание далее на стиль комментария, который совсем неприемлем в нашем курсе лекций – типа «пропишем в коде»):**

Для изменения самой себя, мы обратимся к памяти по адресу и пропишем в коде следующие две команды, которые изменяют 16-ричное представление второй команды mul на нулевое, чтобы при выполнении мы не получали 4ую степень (т.е. при любом значении цифры мы будем получать на выходе квадрат):

.MODEL TINY

.CODE

.STARTUP

ORG 100H

mov ax, 00h

mov [cs:0113h], ax

XOR AX,AX

INT 16H

SUB AL,'0'

MUL AL

CMP AL,16

JG PRINT\_RESULT

MUL AL

PRINT\_RESULT:

MOV BX,2660

PRINT\_SYMBOL:

DIV BL

ADD AL,'0'

INT 29H

MOV AL,AH

MUL BH

CMP AL,0

JNE PRINT\_SYMBOL

MOV AH,4CH

INT 21H

END

На этом пока всё на сегодня.

Просто прочитали?