Жуковский Павел Сергеевич, 4 курс, 12 группа

Лабораторное занятие (30.11.2021)

Эксперименты над 4-ой лабораторной работой

Пароль от архива: **041**

Итак, 09.11.2021 я экспериментировал с LEX.ASM, который позволяет уничтожить исполняемый модуль LEX.EXE при загрузке программы в оперативную память компьютера, а также экспериментировал с 1-ой свиньёй (которую я переименовал в PSWD3TRY.ASM), которая даёт пользователю 3 попытки на угадывание пароля, а при неправильном вводе (на всех трёх попытках) удаляет имя исполняемого модуля из реестра файлов, из-за чего происходит зависание. Я даже проверил, сохраняется ли количество попыток при закрытии терминала после одной или двух неудачных попыток, в этом случае количество оставшихся попыток действительно сохраняется, а не обновляется снова до трёх.

Позже, 16.11.2021, я поэкспериментировал с LEX.EXE в отладчике и сделал для себя два вывода: 1) при загрузке программы из исполняемого модуля LEX.EXE в оперативную память компьютера этот модуль удаляется не только из терминала DOSBOX, но и из отладчика TD.EXE; 2) удаление исполняемого модуля LEX.EXE происходит во время исполнения команды «MOV AX 4C00».

Ещё позже, 23.11.2021, я исполнимый модуль PSWD3TRY.EXE в отладчик и проверял, сохранится ли количество оставшихся попыток после неправильного ввода пароля, но уже не в терминале, а отладчике TD.EXE, после чего пришёл к выводу, что сохранится.

Ещё разок уточним, что необходимо сделать:

*«Получить программу, которая «шифрует» пароль, использует лапшу — «изощренное программирование» (минимум три приёма), при этом выводит на экран сообщение «Ай-да, Имярек, ай-да...» в том случае, если пароль введён верно и при этом программа не находится под отладчиком. А в том случае, когда пароль введен ТРИ раза неправильно, или когда программа находится под отладчиком выводится сообщение «Good bye, my love goodbye», и программа перестаёт существовать в бренном мире компьютеров.»*

Итак, сегодня завершающий этап, к которому я так долго шёл. На основе всех прошлых экспериментов (а также исходного текста HIDEPASS.ASM) я написал следующий исходный текст:

.MODEL SMALL

.STACK 400h

.DATA

EXE\_PATH DB 1024 DUP (0)

PSWD\_DATA DW 06568h, 06553h, 06868h, 06548h, 06565h, 06849h, 06868h, 06553h

DW 06565h, 06854h ; Password: S, H, I, S, T

PSWD\_DATA\_SIZE = $ - PSWD\_DATA

MY\_PSWD DB 255, 0, 256 DUP (0)

NULL\_BYTE DB 0

SUCCESS\_MESSAGE DB "Ђ©-¤ , €¬паҐЄ,  ©-¤ ...$"

FAIL\_MESSAGE DB "Goodbye, my love goodbye$"

RAND\_NUM DW 0

APPROACHES DB 3

INVITE\_MESSAGE DB "Enter your password", 0Dh, 0Ah, "Attempts left: "

INVITE\_NUM DB " "

DB 0Dh, 0Ah, "$"

.CODE

MIX\_PASTA:

PUSH ES

XOR AX, AX

MOV ES, AX

PUSHF

CALL DWORD PTR ES:[68h]

MOV RAND\_NUM, DX

POP ES

RET

RAND\_16\_BITS\_NUM:

MOV AX, 25173

MUL WORD PTR RAND\_NUM

ADD AX, 13849

MOV RAND\_NUM, AX

RET

LOOP\_RAND\_16\_NUMS:

CALL RAND\_16\_BITS\_NUM

MOV CX, AX

CALL RAND\_16\_BITS\_NUM

TEST AX, 1

JNZ .RAND\_16\_NUMS\_START

MOV CH, CL

.RAND\_16\_NUMS\_START:

XOR CH, CH

.RAND\_16\_NUMS\_ON\_LOOP:

CALL RAND\_16\_BITS\_NUM

ROR AX, CL

LOOP .RAND\_16\_NUMS\_ON\_LOOP

RET

PASTA:

JMP .PASTA\_2\_1

.PASTA\_2\_3:

MOV AH, AL

JMP .PASTA\_2\_4

.PASTA\_2\_7:

OR AL, 12h

AND AL, 0FEh

JNZ .PASTA\_2\_8

.PASTA\_2\_4:

XOR AL, 73h

SHL AX, 8

JMP .PASTA\_2\_5

.PASTA\_2\_1:

MOV AX, 0DEh

JMP .PASTA\_2\_2

.PASTA\_2\_9:

XOR AL, 51h

JNE .PASTA\_2\_10

.PASTA\_2\_8:

MOV AH, AL

JMP .PASTA\_2\_9

.PASTA\_2\_5:

.PASTA\_2\_6:

JMP .PASTA\_2\_7

.PASTA\_2\_2:

JMP .PASTA\_2\_3

.PASTA\_2\_10:

RET

CHOOSE\_RAND\_METHOD:

CALL RAND\_16\_BITS\_NUM

XOR DX, DX

MOV CX, 3

DIV CX

TEST DX, DX

JZ .CRM\_EXIT

TEST DX, 1

JNZ .CRM\_PASTA

CALL LOOP\_RAND\_16\_NUMS

JMP .CRM\_EXIT

.CRM\_PASTA:

CALL PASTA

.CRM\_EXIT:

RET

LOAD\_EXE\_PATH:

PUSH ES

PUSH SI

PUSH DI

MOV ES, ES:[002Ch]

XOR SI, SI

MOV AL, 0FFh

.LEP\_GET\_0001h:

MOV AH, ES:[SI]

CMP AX, 1

JE .LEP\_0001h\_GOT

INC SI

MOV AL, AH

JMP .LEP\_GET\_0001h

.LEP\_0001h\_GOT:

INC SI

LEA DI, EXE\_PATH

.LEP\_NAME\_COPYING:

MOV AL, ES:[SI]

INC SI

MOV DS:[DI], AL

INC DI

TEST AL, AL

JNZ .LEP\_NAME\_COPYING

POP DI

POP SI

POP ES

RET

DEBUGGER\_CHECK:

PUSH ES

PUSH BX

CALL CHOOSE\_RAND\_METHOD

XOR AX, AX

MOV ES, AX

MOV BX, ES:[0Ch]

MOV AX, ES:[0Eh]

MOV ES, AX

CMP BYTE PTR ES:[BX], 50h

JNE .DC\_IF\_NO\_DEBUG

STC

JMP .CD\_EXIT

.DC\_IF\_NO\_DEBUG:

CLC

.CD\_EXIT:

POP BX

POP ES

RET

LOAD\_EXE:

LEA DX, EXE\_PATH

MOV AX, 4300h

INT 21h

AND CL, 0FEh

MOV AX, 4301h

INT 21h

MOV AX, 3D02h

INT 21h

RET

GET\_FILE\_SIZE:

XOR CX, CX

XOR DX, DX

MOV AX, 4201h

INT 21h

PUSH AX

MOV AX, 4202h

INT 21h

POP CX

PUSH AX

MOV AX, 4200h

INT 21h

POP CX

RET

FILE\_CLOSE:

MOV AH, 3Eh

INT 21h

RET

DELETE\_EXE\_FILE:

LEA DX, EXE\_PATH

MOV AH, 41h

INT 21h

RET

DEFINE\_SYMB:

PUSH AX

CALL CHOOSE\_RAND\_METHOD

POP AX

XOR AL, BH

MOV CL, 80h

XOR DL, DL

.DS\_LOOP:

TEST AL, CL

JZ .DS\_LOOP\_ON

INC DL

.DS\_LOOP\_ON:

SHR CL, 1

TEST CL, CL

JNZ .DS\_LOOP

TEST DL, 1

JNZ .DS\_USE\_BL

XOR AL, AH

JMP .DS\_EXIT

.DS\_USE\_BL:

XOR AL, bl

.DS\_EXIT:

RET

PSWD\_COMPARE:

PUSH SI

PUSH DI

LEA SI, PSWD\_DATA

LEA DI, MY\_PSWD+2

MOV CX, PSWD\_DATA\_SIZE

SHR CX, 2

CLD

PUSH ES

PUSH DS

POP ES

.COMPARE\_LOOP:

LODSW

MOV BX, AX

LODSW

PUSH CX

CALL DEFINE\_SYMB

POP CX

SCASB

JNE .COMPARE\_LOOP\_STOP

LOOP .COMPARE\_LOOP

.COMPARE\_LOOP\_STOP:

POP ES

TEST CX, CX

JZ .SUCCESS\_COMPARE

STC

JMP .COMPARE\_EXIT

.SUCCESS\_COMPARE:

CLC

.COMPARE\_EXIT:

POP DI

POP SI

RET

OUTPUT\_STR:

MOV AH, 09h

INT 21h

RET

OUTPUT\_INVITATION:

MOV AL, APPROACHES

ADD AL, 30h

MOV INVITE\_NUM, AL

LEA DX, INVITE\_MESSAGE

JMP OUTPUT\_STR

READ\_FROM\_KEYBOARD:

PUSH DI

LEA DX, MY\_PSWD

MOV AH, 0Ah

INT 21h

LEA DI, MY\_PSWD+2

MOV AL, MY\_PSWD+1

XOR AH, AH

ADD DI, AX

MOV BYTE PTR [DI], 0

POP DI

RET

UPDATE\_APPROACHES:

CALL LOAD\_EXE

MOV BX, AX

CALL GET\_FILE\_SIZE

SUB CX, 29h

MOV DX, CX

XOR CX, CX

MOV AX, 4200h

INT 21h

LEA DX, APPROACHES

MOV CX, 1

MOV AH, 40h

INT 21h

RET

DESTROY\_SELF:

PUSH SI

CALL LOAD\_EXE

MOV BX, AX

CALL GET\_FILE\_SIZE

LEA DX, NULL\_BYTE

MOV SI, CX

MOV CX, 1

.DS\_NULLABLE\_FILE:

MOV AH, 40h

INT 21h

DEC SI

JNZ .DS\_NULLABLE\_FILE

CALL FILE\_CLOSE

CALL DELETE\_EXE\_FILE

POP SI

RET

START:

MOV AX, @DATA

MOV DS, AX

CALL MIX\_PASTA

CALL LOAD\_EXE\_PATH

CALL DEBUGGER\_CHECK

JNC .S\_PSWD\_CHECK

JMP .S\_FAIL

.S\_PSWD\_CHECK:

CALL OUTPUT\_INVITATION

CALL READ\_FROM\_KEYBOARD

CALL PSWD\_COMPARE

JNC .S\_PSWD\_CONFIRMED

DEC APPROACHES

CALL UPDATE\_APPROACHES

CMP BYTE PTR APPROACHES, 0

JNZ .S\_PSWD\_CHECK

JMP .S\_FAIL

.S\_PSWD\_CONFIRMED:

LEA DX, SUCCESS\_MESSAGE

CALL OUTPUT\_STR

MOV BYTE PTR APPROACHES, 3

CALL UPDATE\_APPROACHES

JMP .S\_EXIT

.S\_FAIL:

LEA DX, FAIL\_MESSAGE

CALL OUTPUT\_STR

CALL DESTROY\_SELF

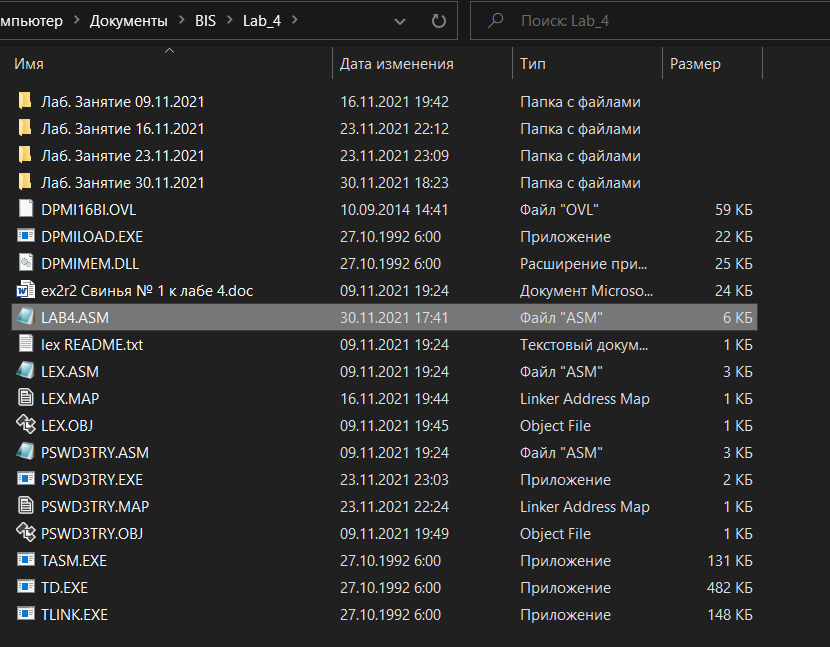
.S\_EXIT:

MOV AX, 4C00h

INT 21h

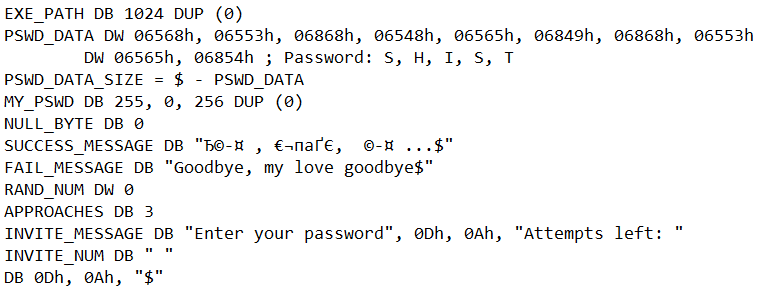
END START

Данный исходный текст я сохранил в файле **LAB4.ASM**:



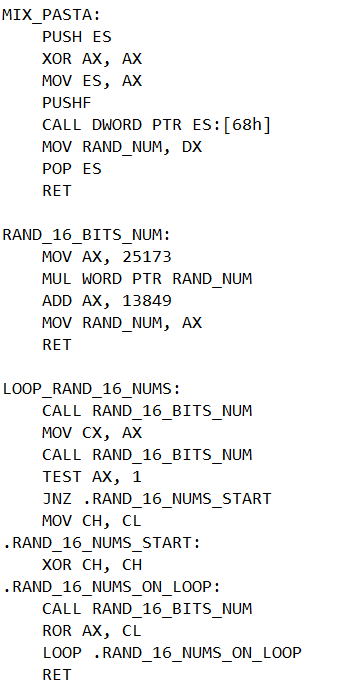
Прежде, чем экспериментировать с программой, которая может быть загружена в оперативную память компьютера с помощью исполнимого модуля, который может быть получен путём линковки соответствующего объектного модуля, который, в свою очередь, может быть получен из вышеуказанного исходного текста, сначала поговорим о том, что эта программа делает.

Здесь хранятся основные данные программы:

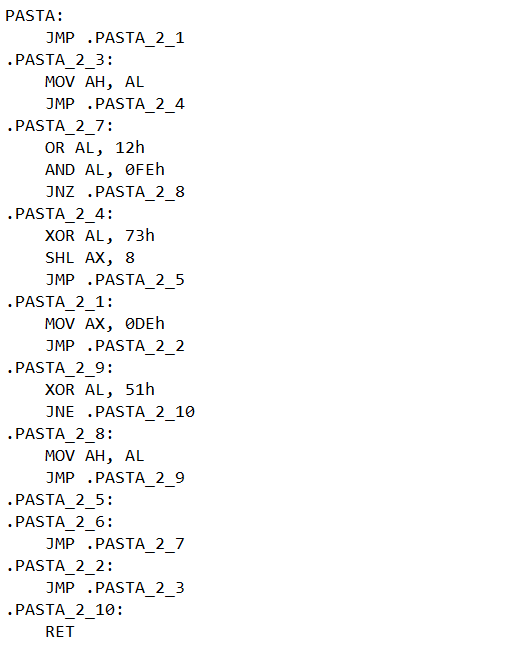


В частности, пароль **SHIST** хранится в виде соответствующих байт (на каждый символ по 2 байта). Фишка точно такая же, как в HIDEPASS. Идея в том, чтобы программа путём своих вычислений по заранее прописанному алгоритму выбирала из двух байт истинный. Таким образом, мы прячем пароль **SHIST** среди лишних байт и запутываем злоумышленника.

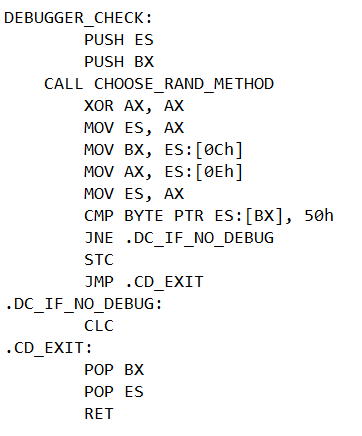
Вот этот участок исходного текста способен генерировать случайные числа и с их помощью ещё сильнее запутывать программу (потому я назвал одну из меток MIX\_PASTA):



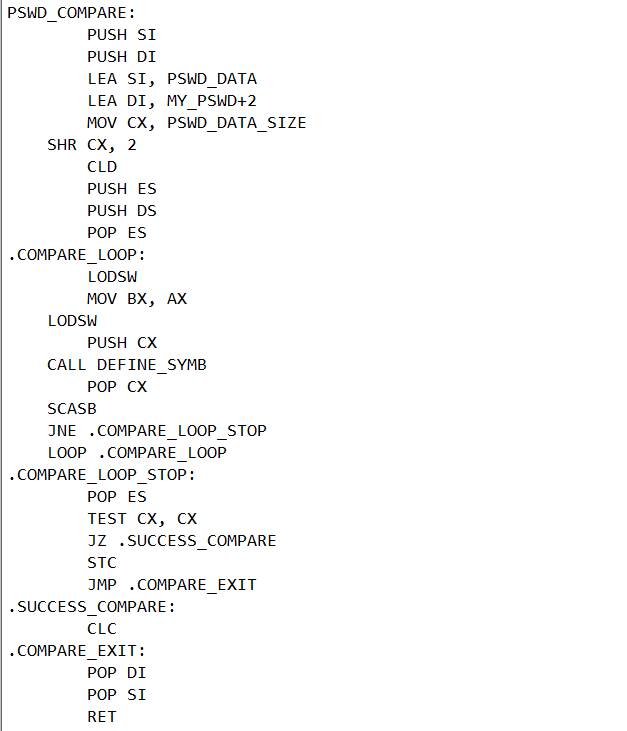
В этой метке мы просто прыгаем между разными метками для ещё большего запутывания:



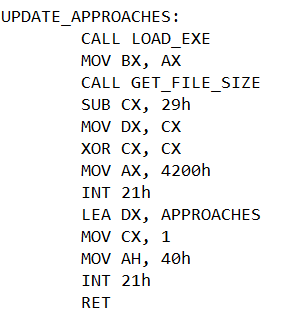
Здесь мы можем проверить, не загружена ли наша программа в отладчик (например, в TD.EXE):



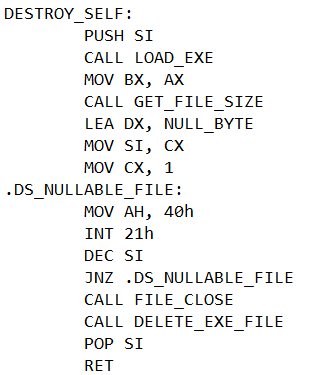
Вот эта последовательность команд сравнивает два пароля (истинный и тот, который был введён с терминала):



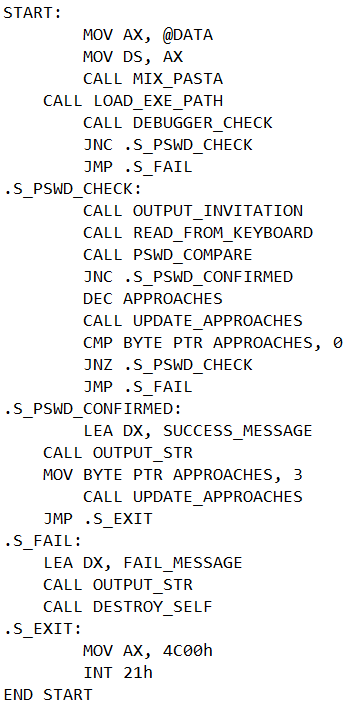
Вот эта последовательность команд обновляет попытки в случае, если пароль был введён успешно:



А вот эта последовательность команд позволит разрушить исполнимый модуль (в случае, если все попытки исчерпаны или программа загружена в отладчик):

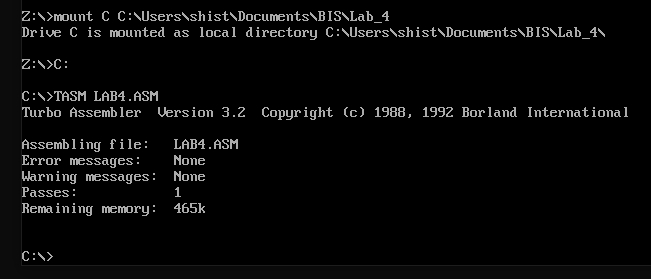


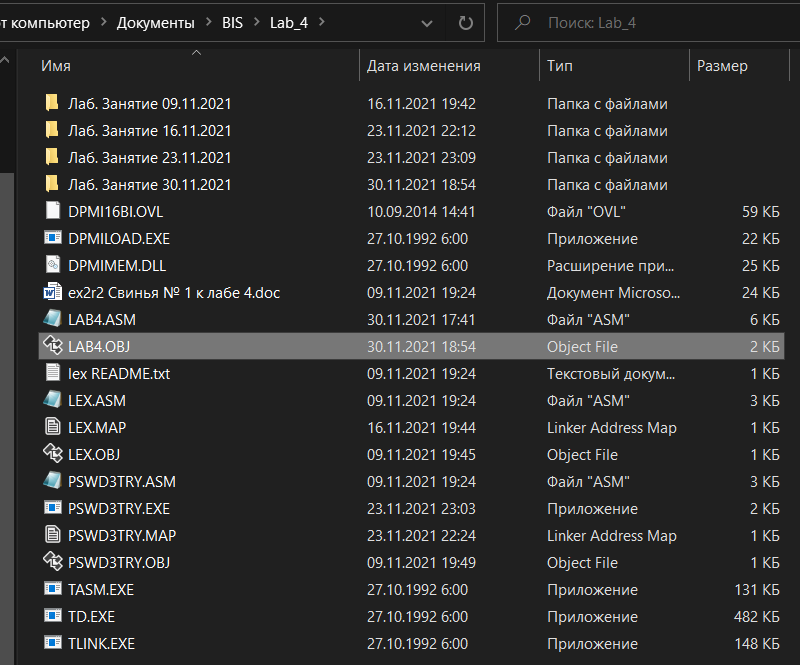
А здесь выполняется основная часть, которая включает в себя вызов всех вышеупомянутых (и не только) меток:



Теперь можно приступать к экспериментам.

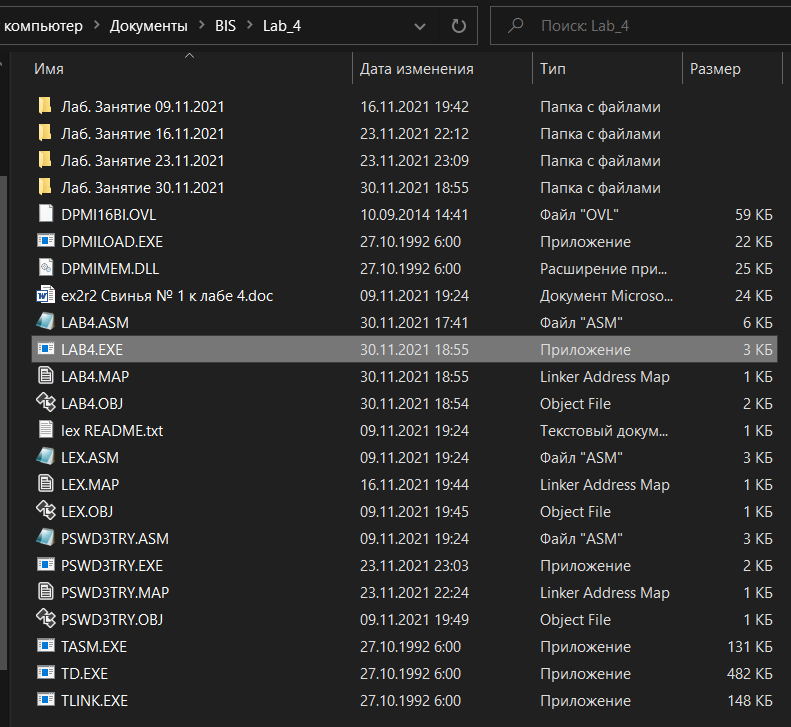
По классике жанра сначала получим объектный модуль **LAB4.OBJ**:



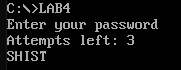


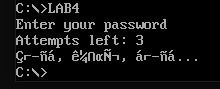
Затем получим исполнимый модуль **LAB4.EXE**:



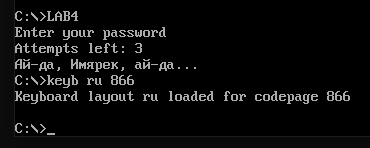


Протестируем полученный исполнимый модуль. Для начала загрузим его в оперативную память компьютера и введём пароль **SHIST** правильно:



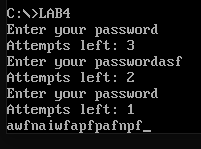


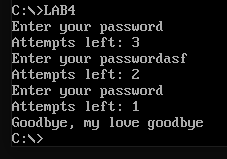
Бедный Имярек, похоже кодировка не та. Настроим DOSBox на нужную кодировку:



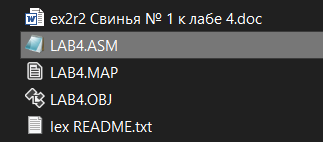
Для этого, я ввёл в терминал **keyb ru 866**. Теперь наш Имярек выводится корректно.

Посмотрим, как поведёт себя программа если мы трижды введём неправильный пароль:



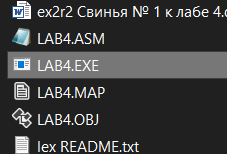


Исполнимый модуль исчез из папки:

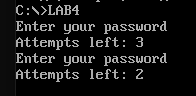


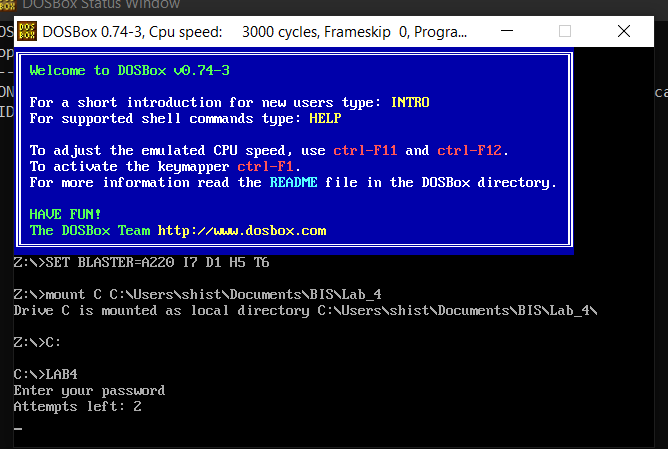
Что ж, создадим его по новой:





Теперь попробуем потратить одну попытку, а потом закрыть окно DOSBox, посмотрим, сохранятся ли оставшиеся попытки в этом случае:

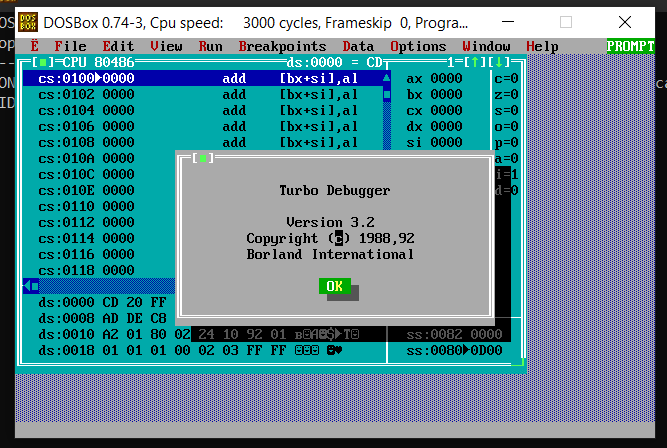


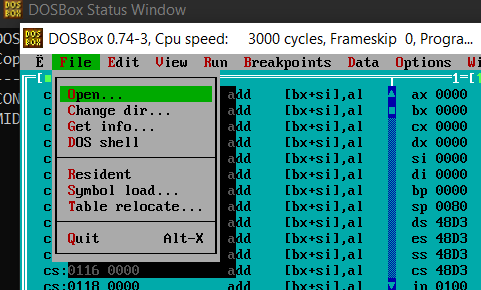


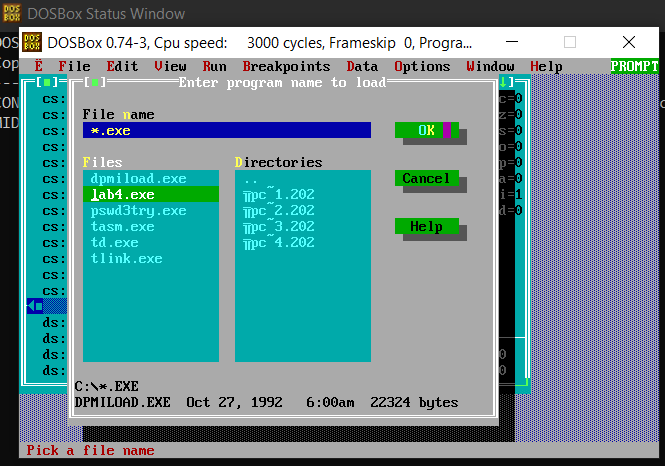
Попытки сохраняются даже после закрытия DOSBox-а до конца выполнения программы. Хитрая редиска теперь не сможет использовать бесконечное число попыток.

Теперь попробуем загрузить исполнимый модуль LAB4.EXE в отладчик TD.EXE:

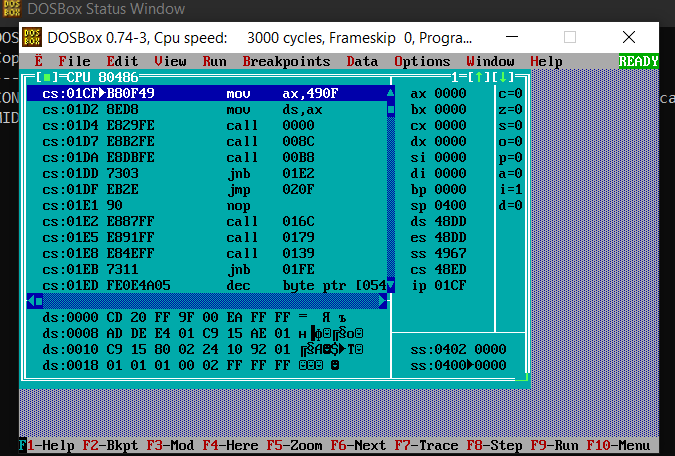




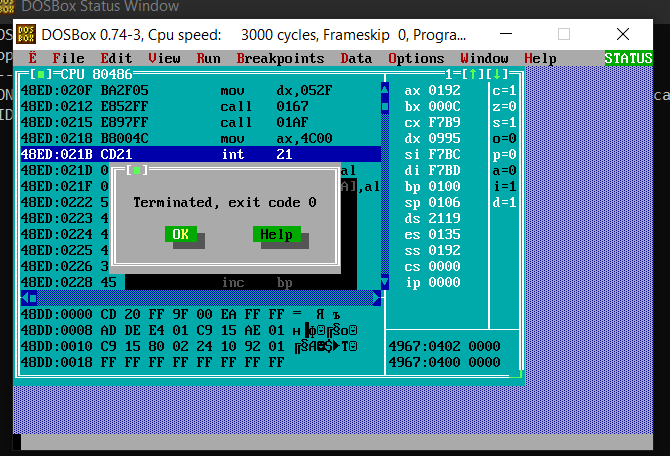




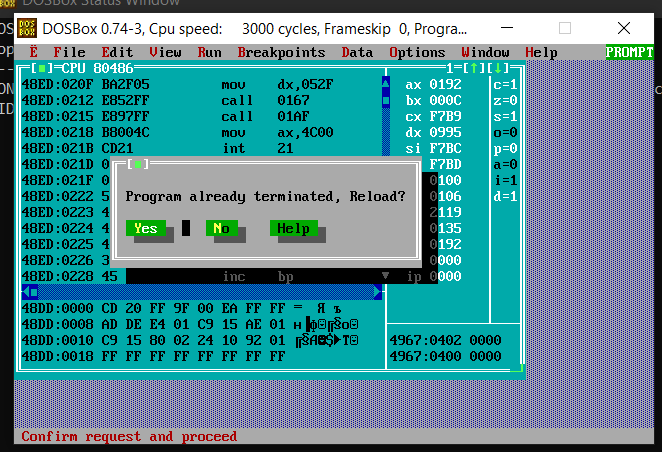
Программа загружена в отладчик:



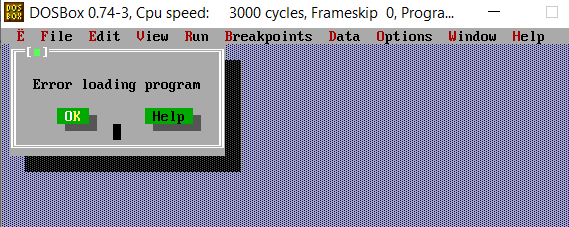
Теперь просто шагаем по программе (клавишей F8) до тех пор, пока программа не запросит нас ввести пароль. Однако на этом шаге программа завершается:



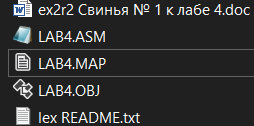
При попытке шагнуть ещё раз отладчик говорит нам, что программа уже завершена, и предлагает попробовать перезагрузить её:



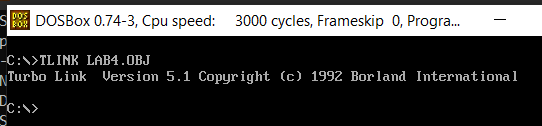
Если попробовать перезагрузить программу, то возникает ошибка:



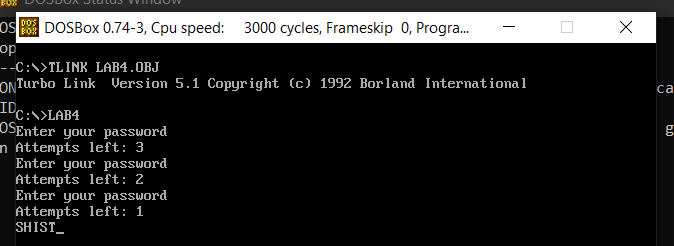
Также, исполнимый модуль программы удаляется в этом случае:

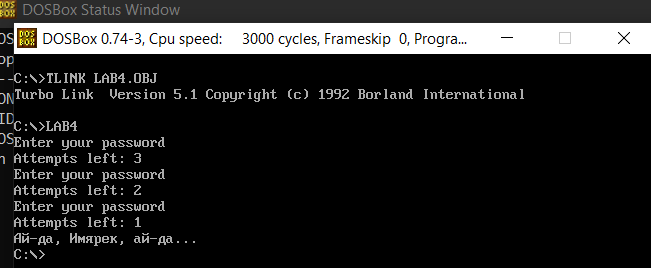


Создадим новый исполнимый модуль:



Теперь попробуем ввести правильный пароль **SHIST**, но не с первой попытки, а, скажем, с третьей:





На этом можно завершить тестирование программы.

Так, мы получили программу, которая способна запутывать злоумышленника самыми разными способами «изощрённого программирования». В случае неправильного ввода пароля 3 раза или использования отладчика, программа удаляет свой исполнимый модуль.

Однако, несмотря на то, что наша программа способна спрятать пароль **SHIST** среди байтов и запутывать саму себя, сложно сказать, что она защищена надёжно.

Для действительно более надёжной защиты пароля следует использовать какой-нибудь метод шифрования, а также защиту от отладки (а не только удаление исполнимого модуля в случае обнаружения отладчика). Так что, нам уж точно есть, куда стремиться

В письме я постараюсь прикрепить архив, который будет включать в себя 3 файла: 1) этот отчёт, 2) исходный текст LAB4.ASM, 3) исполнимый модуль LAB4.EXE.

Пароль от архива: **041** (на всякий случай, укажу пароль на самом верху этого отчёта тоже)