***Лекция 10. Защита программ от взлома.***

**Сегодня читаем, формируя по ходу чтения Word-файл с ответами. Полученный в результате чтения лекции и ДУРКования с вопросами файл, с ответами на вопросы ПЕРЕСЫЛАЕМ мне на почту ДО КОНЦА недели.**

**Вопрос 0.**

**Перечитайте название Лекции, сформулируйте все известные Вам постулаты из области БИС и укажите, сделав обоснование, какое из них наиболее соответствует названию Лекции.**

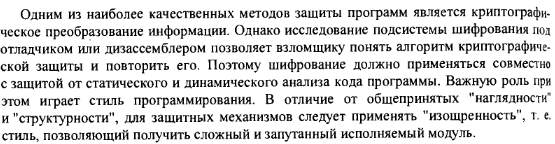
**Одним из постулатов является следующий: Не существует абсолютных методов и средств защиты данных и программ в информационных системах, т.е. любые данные могут быть получены злоумышленником, любая информационная система может быть «взломана».**

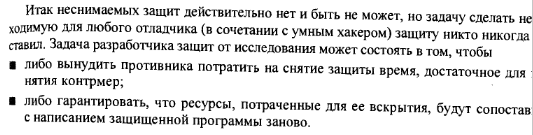
**Еще один постулат звучал так:**

**Защита данных и программ имеет смысл тогда и только тогда, когда СТОИМОСТЬ ОРГАНИЗАЦИИ ЗАЩИТЫ Информационной системы НЕ ПРЕВОСХОДИТ стоимости вреда, понесённого в результате взлома системы.**

**В рамках этих двух постулатов может быть сформулирован ещё один:**

**Так как абсолютной защиты не существует, а 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.**





**Ещё раз. Вопросы по ходу Лекции (после внимательного прочтения предыдущих двух абзацев из книги про ассемблер – на них нужно ответить в Word-файле до конца недели):**

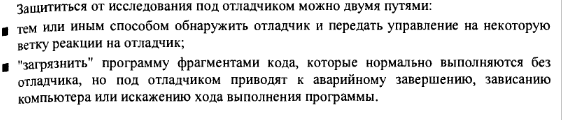
**Вопрос 1.На каких страницах книги записаны набранные мелким шрифтом абзацы.**

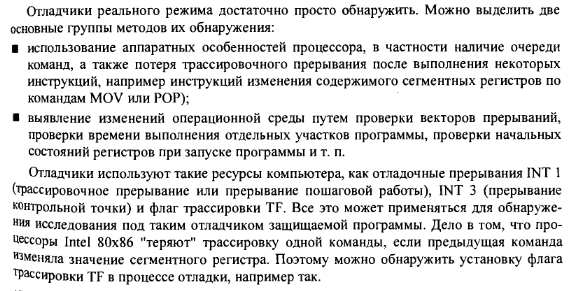
**Самым распространённым способом получения возможности несанкционированного использования программного обеспечения является исследование программ с помощью таких программных средств каковыми являются отладчики и дизассемблеры, с последующим снятием той или иной защиты.**

**Известно, что наиболее часто применяемым методом защиты программ от несанкционированного (бесплатного) использования являются так называемые «Логирование» и «Паролирование». Однако с помощью отладчика или дизассемблера можно запросто исследовать программу и внести в неё изменения с помощью того же отладчика таким образом, чтобы обойти участок программы, «ответственный за проверку пароля» и использовать бесплатно программный продукт. В простонародье сие действо называется «крякнуть продукт». Или нет?**

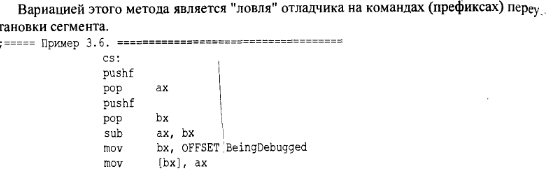
**Вопрос 2. Дать толкование Crack-а на английском языке, взяв его из английской Wikipedia.**

**Сразу же, что приходит на ум в связи с этой «фуагрой»: «А может быть можно как-то защитить свою программу от исследований под отладчиком? Оказывается, что можно:**





**Вопрос 3: КАК ТАК? Записать пример из книги про ассемблер.**



**На страницах 236-238 Вашей нелюбимой книжки про ассемблер описано несколько способов определения того, находится ли Ваша программа под отладчиком. Надо просто почитать. Как и пункты 3.1.4.2 и 3.1.5.Всё равно, как и я, ничего не поймёте. А можете просто открыть книжку на указанной странице и тут же её закрыть, но подтвердив факт этого процесса открытия-закрытия, прислав мне на почту «вырезку» из книжки, начиная со страицы 236 и заканчивая «Изощренным программированием». И это был вопрос 4. Вырезку сопроводить указанием времени, когда эта «вырезка» была сделана.**

**Здесь же и сейчас приведём ещё один текст на ЯЗА ассемблер, который после трансляции и линковки превращается в исполнимый модуль, а он (исполнимый модуль) при загрузке его в ОП превращается в программу. Вопрос 5: В word-файле, который Вам предстоит сформировать ДО КОНЦА НЕДЕЛИ, ПРЕДСТАВИТЬ ОТЧЕТ О НАПИСАНИИ текста, ТРАНСЛЯЦИИ, ЛИНКОВКИ, ИСПОЛНЕНИЯ полученной программы И ДЕМОНСТРАЦИИ РЕЗУЛЬТАТОВ В РАМКАХ ИСПОЛЬЗОВАНИЯ ОТЛАДЧИКА И оболочки DosBox. Показать, что на самом деле клавиатура блокируется (КАК показать? Отвт на этот вопрос как в анекдоте: «Ты же умная, придумай сама» и, в особенности, что понимается под словами «удаляет сама себя» (действительно ли «программа удаляет сама себя» или же она УДАЛЯЕТ ИЗ РЕЕСТРА ФАЙЛОВ файловой системы имя файла, представляющее собой «исполнимый модуль», или же делает что-то ещё: типа «убивает» и (или) изничтожает).**

Исходный текст на ЯЗА ассемблер, из которого (текста) будет получена программа в результате: 1. трансляции исходного текста 2. «линчевания» полученного в результате трансляции объектного модуля, с формированием исполнимого модуля 3. Загрузки этого исполнимого модуля в оперативную память с формирование программы, которая удаляет ИСПОЛНИМЫЙ МОДУЛЬ, а в случае обнаружения отладчика, запрещает работу с клавиатурой – ПРЕДСТАВЛЕН НИЖЕ:

.MODEL TINY

.CODE

ORG 100H

START:

CALL CHECK\_DEBUGGER

JNC DEBUGGER\_NOT\_FOUND\_LABEL

DEBUGGER\_FOUND\_LABEL:

CALL DELETE\_PROGRAM

CALL DEBUGGER\_FOUND

JMP FINISH\_PROGRAM\_LABEL

DEBUGGER\_NOT\_FOUND\_LABEL:

CALL DELETE\_PROGRAM

CALL DEBUGGER\_NOT\_FOUND

FINISH\_PROGRAM\_LABEL:

MOV AX,4C00H

INT 21H

SUCCESS DB 'Debugger not found.',10,13,'$'

FAIL DB 'Debugger found!',10,13,'$'

FILE\_NAME DB 256 DUP(?)

CHECK\_DEBUGGER PROC NEAR

PUSH ES

XOR AX,AX

MOV ES,AX

MOV BX,0CH

XOR DH,DH

MOV DL,BYTE PTR ES:[BX]

SUB DL,08BH

CMP DX,0

JE CHECK\_SUCCESS

STC

JMP FINISH\_CHECK\_DEBUGGER

CHECK\_SUCCESS:

CLC

FINISH\_CHECK\_DEBUGGER:

POP ES

RET

CHECK\_DEBUGGER ENDP

DEBUGGER\_FOUND PROC NEAR

MOV AH,9H

MOV DX,OFFSET FAIL

INT 21H

MOV AL,0ADH

OUT 64H,AL

RET

DEBUGGER\_FOUND ENDP

DEBUGGER\_NOT\_FOUND PROC NEAR

MOV AH,9H

MOV DX,OFFSET SUCCESS

INT 21H

RET

DEBUGGER\_NOT\_FOUND ENDP

DELETE\_PROGRAM PROC NEAR

MOV BX,OFFSET FILE\_NAME

CALL COPY\_FILE\_NAME

MOV DX,OFFSET FILE\_NAME

CALL OPEN\_FILE

CALL CLEAR\_FILE

CALL CLOSE\_FILE

MOV DX,OFFSET FILE\_NAME

CALL DELETE\_FILE

RET

DELETE\_PROGRAM ENDP

COPY\_FILE\_NAME PROC NEAR ; DS:BX - WHERE TO COPY FILE NAME

PUSH ES

MOV AX,ES:[2CH]

MOV ES,AX

MOV SI,-1

SEARCH\_01:

INC SI

MOV AL,ES:[SI]

CMP AL,0

JNE SEARCH\_01

MOV AL,ES:[SI+1]

CMP AL,1

JNE SEARCH\_01

ADD SI,2

COPY\_NAME:

INC SI

MOV AL,ES:[SI]

MOV [BX],AL

INC BX

CMP AL,0

JNE COPY\_NAME

POP ES

RET

COPY\_FILE\_NAME ENDP

OPEN\_FILE PROC NEAR ; DS:DX - FILE NAME, BX - RESULT (FILE HANDLE)

MOV AX,3D02H

INT 21H

MOV BX,AX

RET

OPEN\_FILE ENDP

CLOSE\_FILE PROC NEAR ; BX - FILE HANDLE

MOV AH,3EH

INT 21H

RET

CLOSE\_FILE ENDP

GET\_FILE\_LENGTH PROC NEAR ; BX - FILE HANDLE, SI - RESULT

MOV AX,4202H

XOR CX,CX

XOR DX,DX

INT 21H

MOV SI,AX

MOV AX,4200H

XOR CX,CX

XOR DX,DX

INT 21H

RET

GET\_FILE\_LENGTH ENDP

CLEAR\_FILE PROC NEAR ; BX - FILE HANDLE

CALL GET\_FILE\_LENGTH

MOV CX,SI

MOV DX,OFFSET TRASH\_BUF

PRINT\_TRASH\_TO\_FILE:

PUSH CX

MOV AH,40H

MOV CX,1

INT 21H

POP CX

LOOP PRINT\_TRASH\_TO\_FILE

RET

TRASH\_BUF DB 0

CLEAR\_FILE ENDP

DELETE\_FILE PROC NEAR ; DS:DX - FILE NAME

MOV AH,41H

INT 21H

RET

DELETE\_FILE ENDP

END START

**Следует обратить внимание на то, каким образом определяется, что программа находится под отладчиком:**

CHECK\_DEBUGGER PROC NEAR

PUSH ES

XOR AX,AX

MOV ES,AX

MOV BX,0CH

XOR DH,DH

MOV DL,BYTE PTR ES:[BX]

SUB DL,08BH

CMP DX,0

JE CHECK\_SUCCESS

STC

JMP FINISH\_CHECK\_DEBUGGER

CHECK\_SUCCESS:

CLC

FINISH\_CHECK\_DEBUGGER:

POP ES

RET

CHECK\_DEBUGGER ENDP

**Вопрос 6. Указать номер примера из книги про ассемблер, который был использован при написании процедуры CHECK\_DEBUGGER. И какое прерывание анализируется в этом случае…. Указать его номер.**

**Теперь пожуём. Как известно, при загрузке программной части BIOS, в Оперативную память, начиная с нулевого адреса (ES) = 0, загружается так называемая ТАБЛИЦА ВЕКТОРОВ ПРЕРЫВАНИЙ, причём каждый вектор прерывания представляет собой четырехбайтовую область памяти, в которой ХРАНИТСЯ АДРЕС ОБРАБОТЧИКА соответствующего ПРЕРЫВАНИЯ. Таблица векторов прерываний организована таким образом, что вектора прерывания расположены по порядку возрастания их номеров. То есть по адресу 0:0 расположен адрес функции – обработчика прерывания с номером 0 – события под названием «Деление на ноль». По адресу 0:4 располагается адрес функции – обработчика прерывания с номером 1 – события …**

**Вопрос 7. Дописать вместо трёх точек событие, которое называется прерыванием с номером 1.**

Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. Это прерывание не вырабатывается после выполнения команды MOV в сегментные регистры или после загрузки сегментных регистров командой POP.

**По адресу 0:12 (0:0Сh) располагается адрес функции – обработчика прерывания с номером 3 – события под названием «**Прерывание для трассировки. Это прерывание генерируется при выполнении однобайтовой машинной команды с кодом CCh и обычно используется отладчиками для установки точки прерывания.**»**

Для того, чтобы добраться до первого байта вектора прерывания прерывания INT 3 и был записан следующий фрагмент:

MOV ES,AX

MOV BX,0CH

XOR DH,DH

MOV DL,BYTE PTR ES:[BX]

**Затем от значения первого байта вектора прерывания с номером 3 отнимается значение 08Bh. Значение 08Bh представляет собой в двоичной системе значение, равное = 0100 1011, а если доступ к этому байту осуществить как к коду команды, то оказывается, что это команда MOV. В том случае, если разность между значением, находящимся в первом байте вектора прерывания с номером три и значением 08Bh, не равно нулю, то это свидетельствует о том, что программа НАХОДИТСЯ ПОД ОТЛАДЧИКОМ.**

**Вопрос 8: Обманул-не-обманул? В каком месте?**

SUB DL,08BH

CMP DX,0

JE CHECK\_SUCCESS

STC

JMP FINISH\_CHECK\_DEBUGGER

CHECK\_SUCCESS:

CLC

FINISH\_CHECK\_DEBUGGER:

POP ES

**Команда STC - Инструкция STC** в Ассемблере устанавливает флаг переноса CF. Синтаксис:

**STC**

После выполнения этой команды флаг переноса CF будет установлен. Остальные [**флаги**](http://av-assembler.ru/asm/afd/asm-flags-register.htm) не изменяются.

**Команда CLC** в Ассемблере - это команда сброса флага CF, которая выполняет сброс флага переноса. Синтаксис:

**CLC**

После выполнения этой команды флаг CF сбрасывается (очищается, то есть будет равен 0). Остальные [**флаги**](http://av-assembler.ru/asm/afd/asm-flags-register.htm) не изменяются.

**Вопрос 9. Что такое РЕГИСТР ФЛАГОВ. Используя средства Microsoft Word нарисовать его и указать в 10 с.с. НОМЕР БИТА, который «отвечает» ЗА ПЕРЕНОС.**

**НЕ ЗАБЫВАЙТЕ О ТОМ, ЧТО НУЖНО СДЕЛАТЬ ДО КОНЦА НЕДЕЛИ!!!! НЕДЕЛЯ ЗАКАНЧИВАЕТСЯ ВОСКРЕСЕНЬЕМ! ФАЙЛ С ОТЧЁТОМ!**