**Защита программ от компьютерных вирусов**

Реферат по информатике ученика 11”А” класса школы №776 Юго – Восточного административного округа Афонина Сергея Викторовича

**Предисловие**

В главе анализируется механизм распространения и функционирования вирусов в операционной системе MS-DOS и на основе анализа предлагаются достаточно эффективные способы борьбы с ними. Приводятся описания трех разных программ, обеспечивающих выявление и ликвидацию вирусов. Модуль F\_Anti может использоваться для автоматической защиты вновь разрабатываемых Турбо Паскалевых программ. Программа AntiVir осуществляет выявление и ликвидацию загрузочных вирусов и контроль любых исполняемых файлов на основе сопоставления наиболее уязвимых для вируса частей файлов с их эталонными копиями. Наконец, комплекс программ SetFag.exe и Fag,prg даст Вам возможность установить антивирусную программу—фаг на любой уже созданный ЕХЕ-файл: в момент запуска программы фаг проверит ее состояние, если программа поражена вирусом, сообщит об этом и удалит вирус.

**1. Что такое компьютерные вирусы**

Если Вы имеете опыт продолжительной работы с ПК, то, возможно, уже сталкивались с компьютерными вирусами или хотя бы слышали о них. Компьютерный вирус-это программа, производящая в Вашем ПК действия, в которых Вы не нуждаетесь и о которых не подозреваете. Главной ее особенностью является способность к «размножению», т.е. к созданию множества готовых к дальнейшей работе экземпляров вируса. Вирусы «цепляются» к обычным исполняемым файлам типа .ЕХЕ, .СОМ или к загрузочным секторам физических носителей информации (дискет) и таким образом перемещаются от одного ПК к другому.

Являвшиеся первоначально вполне невинным развлечением скучающих программистов компьютерные вирусы сегодня стали настоящим бедствием для пользователей ПК: количество и типы таких программ растут с ужасающей скоростью, а сами вирусы в ряде случаев приобрели весьма неприятные свойства-некоторые из них способны уничтожать файловую структуру дисков со всеми катастрофическими для пользователя последствиями. В литературе [14] описывается беспрецедентный случай, когда вирус на Три дня (с 2 по 4 ноября 1988 г.) вывел из строя фактически всю компьютерную сеть США. Были парализованы компьютеры Агентства национальной безопасности, Стратегического командования ВВС США, локальные сети всех крупных университетов и исследовательских центров. Лишь в последний момент удалось спасти систему управления полетом космических кораблей Шаттл. Положение было настолько серьезным, что к расследованию немедленно приступило ФБР. Виновником катастрофы, причинившей ущерб более чем в 100 миллионов долларов, оказался студент выпускного курса Корнеллского университета Р.Моррис, придумавший достаточно хитрую разновидность вируса. Он был исключен из университета с правом восстановления через год и приговорен судом к уплате штрафа в 270 тысяч долларов и трем месяцам тюремного заключения.

Трудно объяснить, для чего программисты тратят силы и время на создание все более изощренных типов вируса, поскольку их авторы почти всегда остаются или надеются остаться анонимными, так что естественное для человека стремление к известности здесь исключено. Может быть это неудачная шутка (этой версии придерживался Р.Моррис), возможно это связано с патологическими отклонениями в психике, а может быть объяснение кроется в стремлении заработать на создании антивирусных программ? Как бы там ни было, нам нельзя не считаться с возможностью заражения ПК компьютерным вирусом.

**2. Цикл функционирования вирусов**

В цикле существования любого вируса можно выделить три этапа. Первоначально вирус находится в неактивном состоянии. В этом состоянии он внедрен в тело исполняемого файла или находится в загрузочном секторе диска и «ждет» своего часа. Именно в неактивном состоянии вирусы переносятся вместе с программами или дискетами от одного ПК к другому (обмен программами между пользователями ПК-явление обыденное, и Вы сами, читатель, возможно копировали полюбившуюся Вам игру или текстовый редактор, не подозревая, что копируете еще и вирус). Разумеется, в неактивном состоянии вирус ничего не может сделать. Для того чтобы он начал свою работу, необходимо запустить исполняемый файл или загрузиться с зараженной дискеты. В этот момент активизируется вирус, который либо создает резидентную в памяти программу, способную порождать копии или производить какие-то разрушительные действия, либо немедленно приступает к работе.

Если вирус создал резидентную программу, то ее активизация осуществляется различными способами - все зависит от фантазии автора вируса. Обычно вирус перехватывает прерывание $21, являющееся ключевым для доступа к любым операциям с MS-DOS. Таким образом, любая попытка чтения или записи информации на диск или обращение к клавиатуре дисплея приводит к активизации резидентной программы вируса. После получения управления (или активизации резидентной программы) вирус приступает к «размножению»: он отыскивает подходящий исполняемый файл и внедряет свой код в его тело (см. ниже). Как правило, вирус заражает лишь один исполняемый файл за раз, чтобы пользователь не обратил внимания на чрезмерное замедление в работе программ: второй этап жизнедеятельности вируса - это этап активного размножения, поэтому вирусная программа стремится максимально скрыть от пользователя ПК результаты своей деятельности.

После того как заражено достаточно много файлов, может наступить третий этап, связанный с внешними проявлениями работы вируса. Ваш компьютер вдруг начнет вести себя странно: зазвучит ли музыкальная фраза, или начнут «сыпаться» символы на экране дисплея-не суть важно, главное, что только в этот момент Вы с ужасом вспомните, что на жестком диске находятся чрезвычайно важные для Вас данные или программы, которые Вы не успели или не смогли скопировать на дискеты. Увы! Некоторые вирусы к этому моменту могут уже безвозвратно нарушить файловую структуру.

Что делать при обнаружении вируса? Прежде всего не паниковать, ведь далеко не все вирусы отличаются «кровожадностью». Следует выключить компьютер, вставить в привод гибкого диска заранее припасенную эталонную системную дискету (никогда не снимайте с нее защиту!) и снова включить компьютер. Если на ПК имеется специальная кнопка для перезагрузки {RESET или INIT), можно использовать ее и не выключать/включать компьютер, но во всех случаях не пытайтесь перезагрузиться с помощью Ctrl-Alt-Del: от некоторых типов вируса Вы не избавитесь таким образом. Затем нужно запустить какую-либо программу—антивирус (например, AIDSTEST Д.Н.Лозинского) и с ее помощью локализовать и удалить вирус. Если антивирусная программа не может обнаружить вирус или у Вас под рукой нет такого рода программ, следует попробовать обратиться к нужному Вам жесткому диску и, если это удастся сделать, скопировать все наиболее ценное (но только не исполняемые файлы!) на дискеты. После этого заново переформатировать жесткий диск, перенести на него эталонную копию ДОС и восстановить с дискет то, что удалось спасти.

**3. «Вакцинация» программ**

Что же следует сделать, чтобы эта безрадостная картина не стала реальностью? Один ответ очевиден-периодически (и по возможности чаще) сохранять жизненно важные для Вас результаты работы на дискетах. Нет ничего проще, чем дать этот совет, гораздо сложнее заставить себя следовать ему: я сам, честно говоря, далеко не каждый день трачу время на архивирование. Второй ответ менее очевиден. С его простой идеей я впервые познакомился в прекрасной статье Ф.Н.Шерстюка.

Вот эта идея: нужно произвести «вакцинацию» исполняемых программ, т.е. придать им свойство самодиагностики, позволяющее произвести контроль собственного файла и выяснить, заражен он или нет. Если факт заражения установлен, программа может попытаться восстановить свой исходный вид, т.е. удалить прицепившийся к ее файлу вирус. Если эту идею последовательно воплощать в жизнь, то большинство Ваших программ приобретет стойкий «иммунитет» к вирусам, во всяком случае, они смогут достаточно быстро сообщить Вам о факте заражения.

Преимущества этой идеи очевидны: в отличие от разработчиков многочисленных антивирусных программ, которые борются с конкретными разновидностями вирусов, Вы можете сохранить в файле программы некоторую ключевую информацию о ее незараженном виде, и поэтому факт заражения любым видом вируса может быть легко установлен в момент запуска Вашей программы.

**3.1. Заголовок исполняемых файлов**

Какую именно информацию о нзараженном файле следует сохранять? Для ответа на этот вопрос необходимо знать соглашение ДОС о формате исполняемых файлов. Как известно, существуют два формата: СОМ и ЕХЕ. Любая программа, обрабатываемая системой Турбо Паскаль версии 4.0 и выше, может быть оттранслирована только в ЕХЕ-файл, поэтому все дальнейшие рассуждения относятся именно к этому формату.

В начале ЕХЕ-файла располагается заголовок, в котором содержится вся информация, необходимая для преобразования дискового файла в готовую к работе программу. Первые 28 байт заголовка соответствуют следующей структуре данных:

Type

HeadExeType = record

Sign: Word; {Признак ЕХЕ-файла}

PartPag: Word; {Часть неполного сектора в конце файла}

PageCnt: Word; {Количество секторов, включая неполный}

ReloCnt: Word; {Количество элементов в таблице перемещения}

HdrSize: Word; {Длина заголовка в параграфах}

MinMem: Word; {Минимальный размер кучи (в параграфах)}

МахМет: Word; {Максимальный размер кучи (в параграфах)}

ReloSS: Word; {Начальное значение сегмента стека SS}

ExeSP: Word; {Начальное значение указателя стека SP}

ChkSum: Word; {Контрольная сумма всех слов файла}

ExelP: Word; {Смещение точки запуска программы}

ReloCS: Word; {Начальное значение сегмента кода CS};

TabiOff: Word; {Смещение первого элемента таблицы перемещения}

Overlay: Word; {Номер оверлея или 0 для основной программы}

end; {HeadExe}

Остальные элементы заголовка содержат так называемую таблицу перемещения, предназначенную для настройки адресов загруженной программы. Таблица начинается с байта TabiOff от начала файла и содержит ReloCnt четырехбайтных элементов следующего вида:

Type

ReloTablltem = record

ItemSeg: Word; {Сегмент перемещаемого адреса}

IternOfs: Word; {Смещение перемещаемого адреса}

end;

Признак ЕХЕ-файла хранится в поле Sign в виде символов «MZ» (код $5A4D) - с этого признака должен начинаться любой ЕХЕ-файл. Поле HdrSize содержит длину всего заголовка в параграфах (участках памяти длиной по 16 байт каждый). Поля PartPag и PageCnt определяют общую длину загружаемой в память части ЕХЕ-файла по следующей формуле:

L = (PageCnt-l)\*512 + PartPag - HdrSize\*16

Остальная часть файла (длина ЕХЕ-файла может быть больше L+HdrSize\*16) при загрузке программы не учитывается. Обычно в остатке файла, созданного системой Турбо Паскаль, (если, разумеется, есть остаток) содержится информация, используемая встроенным отладчиком, или оверлеи.

Подавляющее большинство ЕХЕ-вирусов пристыковывает свою программу в конец файла, а для того чтобы эта программа была загружена в память и ей было передано управление, изменяет поля PartPag, PageCnt, ReloCS, ExelP (адрес точки, куда передается управление после окончания загрузки) и, возможно, некоторые другие поля. При таком способе внедрения общая длина загружаемой в память части файла должна составлять

ExeSize = FileSize + VirusSize,

где FileSize-полная длина ЕХЕ-файла, а VirusSize-длина программы вируса. Так как в остатке фала могут храниться оверлеи (или архив для саморазгружающихся архивных программ), длина ExeSize может оказаться чрез-мерно большой, так что программа не сможет загрузиться в память или не сможет работать нормальным образом. Некоторые безграмотно написанные вирусы не учитывают это обстоятельство и быстро выдают себя, т.к. зараженные программы перестают работать.

Другой способ внедрения вируса-пристыковка кода вируса до начала загружаемой части программы и сразу за заголовком файла. L - загружаемая в память часть файла.

Такой способ внедрения позволяет не загружать в память весь ЕХЕ-файл, а длина загруженной программы увеличивается только на длину кода вируса. Несмотря на кажущееся преимущество такого способа, он используется достаточно редко. Его реализация значительно сложнее, так как перед передачей управления основной программе вирус должен перенести 256 байт префикса программного сегмента {PSP) в конец собственного кода так, чтобы они непосредственно предшествовали телу программы-в противном случае будет нарушена важная связь программы с PSP или относительная адресация в самой программе.

Кроме того, в процессе заражения он должен увеличить на величину VirusSize поле IternOfs каждого элемента таблицы перемещения и абсолютного адреса, указываемого этим элементом. В отличие от стандартного загрузчика ДОС вирусу приходится корректировать не загруженную программу, а ее файловый образ. Так как в ЕХЕ-программе средней сложности может быть несколько сотен элементов таблицы перемещения, процесс настройки таблицы вирусом приводит к заметному увеличению времени запуска программы, что может обнаружиться пользователем. На этапе размножения вирусы стремятся по возможности скрыть от пользователя результат своей деятельности, поэтому ЕХЕ-файлы редко поражаются вирусами, пристыкованными в начало файла.

Разумеется, существует возможность внедрения вируса непосредственно в тело исполняемой программы. Однако на практике это почти всегда означает разрушение логики работы программы, поэтому такой вирус немедленно обнаруживается.

Анализ сказанного позволяет сделать важный вывод: практически любой существующий вирус (или вирус, который еще только будет создан!), рассчитанный на поражение ЕХЕ-файла, пристыковывает свой код в конец файла и изменяет его заголовок. Следовательно, для контроля факта заражения программы и ликвидации вируса необходимо где-то сохранить заголовок файла и его эталонную длину и периодически сопоставлять действительный заголовок и длину с эталонными значениями. При этом следует учитывать то обстоятельство, что некоторые вирусы контролируют любое обращение к дисковым секторам, в которых расположена их программа, и «подсовывают» незара-женные копии этих секторов. Такие вирусы (их называют вирусы—невидимки) вряд ли удастся обнаружить с помощью стандартного обращения к функциям ДОС. Для борьбы с ними используют прямое обращение к BIOS-прерыванию $13.

**3.2. Защита вновь создаваемых программ**

Ключевую информацию (будем для краткости называть ее ключ) о незараженной программе можно хранить в отдельном файле, но в этом случае существует опасность потерять дополнительный файл при копировании программы или ошибочно уничтожить его. Гораздо надежнее хранить ключ в теле самого защищаемого файла. К сожалению, его нельзя подобно вирусу при-стыковать в конец файла, т.к. в случае заражения вирус изменит поля PartPag и PageCnt и мы никогда не сможем определить то место в файле, где он располагается. Вспомним, что все константы (в том числе и типизированные) создаются на этапе компиляции программы, таким образом в файле обязательно имеется область данных, содержащая значения этих констант. Эта область в Турбо Паскалевых программах располагается в самом конце загружаемой части файла (см.рис.6.2).

Следовательно, мы должны объявить в программе типизированную констант, предназначенную для хранения ключа, а затем в область файла, отведенную для ее размещения, поместить нужную информацию.

Каким образом отыскать в ЕХЕ-файле место, занимаемое ключом? Конечно, можно перед ним в программе разместить какую-либо типизированную константу с характерным значением (например, заранее обусловленную текстовую строку) и затем отыскивать ее в файле. Однако такое решение вряд ли можно признать удовлетворительным: во-первых, всегда существует вероятность того, что какой-то фрагмент кодов программы содержит ту же цепочку байт, что и заголовок ключа; во-вторых, придется просматривать подчас большой по объему ЕХЕ—файл в поисках нужной константы. Значительно изящнее выглядит решение, основанное на точном вычислении смещения от начала файла до ключа.

Для этого нужно определить начало области данных в файле. В заголовке файла не предусмотрено никакой информации о начальном значении регистра DS, в котором хранится сегмент данных. Перед передачей управления программе загрузчик устанавливает значение этого регистра так, чтобы он указывал на начало так называемого префикса программного сегмента {PSP), а запущенная программа уже сама должна установить его надлежащим образом.

Префикс программного сегмента имеет длину 256 байт (16 параграфов) и размещается загрузчиком в оперативной памяти непосредственно перед загруженной программой. В PSP содержится различная служебная информация, которая может быть достаточно важной для ДОС и работающей программы (в PSP, например, хранятся параметры обращения к программе), но в нем нет указаний на содержимое регистра DS. Таким образом, найти начало области данных в файле достаточно трудно. Однако это нетрудно сделать в работающей программе-для этого предусмотрена стандартная функция DSeg, возвращающая сегмент данных. Если бы программа могла получить также информацию о сегменте PSP, с которого начинается загруженная программа, можно было бы вычислить смещение в файле от его начала до начала области данных. Такая возможность есть-функция MS-DOS с номером $62 возвращает в регистре ВХ значение сегмента PSP. Следовательно, программа должна получить значения обоих сегментов, вычислить их разницу и таким образом определить место в файле, с которого начинается область данных.

Остается последняя проблема-как найти нужную константу. Турбо Паскаль размещает константы в области данных по мере их объявления в программе. Зная размер каждой константы и порядок их объявления, можно вычислить место размещения нужной нам типизированной константы. Однако этот метод не годится для универсальной программы защиты, так как порядок объявления констант может меняться от программы к программе. К счастью, мы можем использовать операцию получения адреса @. Результатом применения этой операции к адресу константы, выбранной в качестве ключа, является указатель (четырехбайтный адрес); смещение адреса, который он содержит, и является нужным нам смещением начала ключа относительно начала области данных.

**3.3. Модуль F\_Anti**

В этом параграфе описывается модуль F\_Anti, в котором осуществляются все необходимые действия по установке ключа во вновь создаваемый ЕХЕ-файл и последующей проверке этого ключа при каждой загрузке программы. Полный текст модуля см. прил.П7.

В качестве ключа для контроля факта заражения ЕХЕ-файла в модуле F\_Anti используется типизированная константа Head, соответствующая такой структуре данных:

Type

НТуре = record

НЕ : HeadExeType; {Эталонный заголовок файла}

HL : Longint; {Эталонная длина файла}

HF : Boolean; {Флаг установки ключа}

Key: Word ; {Шифр для защиты ключа}

end;

При создании этой типизированной константы компилятор в поле HF помещает значение False. Запущенная программа анализирует это поле. При первом прогоне HF = False, в результате чего программа вызывает процедуру Save, в ходе выполнения которой в файле сохраняются эталонный заголовок и начальная длина файла. При этом одновременно в поле HF (в файле программы!) помещается значение True, поэтому при любом последующем запуске программы вместо Save будет вызвана процедура CheckFile, которая осуществит нужный контроль файла. Если в ходе контроля обнаруживается хотя бы малейшее отличие заголовка файла от его эталона, хранящегося в Head, программа сообщает о факте заражения и предлагает восстановить испорченный заголовок и начальную длину файла.

При восстановлении зараженного файла осуществляются следующие действия:

• зараженный файл копируется в файл с расширением VIR-это позволит Вам в случае неудачной попытки восстановления повторить ее еще раз; кроме того, при желании Вы сможете передать разработчикам антивирусных программ копию вируса для выработки методов его обнаружения и уничтожения;

• анализируется адрес запуска зараженной программы: если относительное значение сегмента точки запуска у зараженной программы меньше, чем в эталоне, то это означает, что вирус располагается в начале или в середине.

Если программа запускается на уже зараженном ПК. не исключено, что заголовок файла будет испорчен вирусом, и тогда система защиты будет ревностно следить за сохранностью вируса! В этом случае программа предупреждает пользователя о невозможности восстановления (если Вас заинтересовал описываемый способ борьбы с вирусами, попробуйте модифицировать программу защиты так, чтобы она работала и в этом случае);

• если вирус пристыкован в конец файла, то в восстанавливаемый файл переносится эталонный заголовок и незараженная часть, начиная с байта TablOff + ReloCnt'4 от начала файла и до байта FileSize-HL Отмечу следующее обстоятельство. Описанная система защиты дублирует заголовок файла в тело программы. Ничто не мешает вирусу проверить тело программы на наличие дубликата и соответствующим образом изменить его. Я далек от мысли, что Вы,, уважаемый читатель, после знакомства с этим разделом захотите создать собственный вирус, учитывающий это обстоятельство. Тем не менее я считаю, что нелишней будет какая-то защита самого ключа. Такая защита реализуется очень просто: для этого достаточно все 16—битные поля эталона заголовка сложить по модулю 2 со случайной константой Key, С этой целью в процедуру Save помещен следующий фрагмент (см. текст модуля в прил.П7):

{Зашифровать ключ:} Randomize; Head.Key := Random($FFFF);

with Head,

Head .HE do for k = I to 14 do

Hem[k] := HE. Hem [k] xor Key;

(массив Hem совмещен в памяти с сохраняемым в файле эталонным заголовком, а НЕ .Нет-с заголовком файла в момент первого запуска).

Инициация генератора случайных чисел с помощью процедуры Randomize гарантирует, что шифр Head.Key не будет повторяться в различных программах. В процедуре CheckFile с помощью операторов

{Дешифровать ключ: }

with Н,Н.НЕ do for k := I to 14 do Hem[k] := Hem[k] xor Key;

восстанавливается исходный вид ключа.

Процедура CheckFile, осуществляющая установку или контроль ключа, вызывается в ходе выполнения установочной части модуля F\_Anti, поэтому для использования описанного метода защиты достаточно указать имя модуля в предложении Uses. Замечу, что в случае разработки программы с оверлеями модуль F\_Anti можно объявить оверлейным, если в установочной части любого неоверлейного модуля инициируется работа администратора оверлея.

В распоряжении программиста имеется глобальная переменная CheckVirasResult, сигнализирующая о результатах проверки программы. Значения этой переменной интепретируются следующим образом:

0; {Не обнаружен факт заражения} 1; {Первый запуск, в программе установлена защита}

-1; {Вирус обезврежен с согласия пользователя}

-2; {Вирус обезврежен автоматически} .

-3; {Контроль подавлен ключом /NOANTI}

-4; {Вирус расположен в начале программы}

Работа процедуры CheckFile может быть подавлена, если программа за-пускается с ключом /NOANTI. Ключ /NOQUERY разрешает автоматическое удаление обнаруженного вируса без разрешения пользователя.

Ключ /NOALARM также разрешает процедуре автоматически удалить вирус, но запрещает выдавать на экран предупреждающее сообщение. Наконец, ключ /NOCOPY запрещает создание резервной копии зараженного файла (с расширением VIR).

Следующая простая программа иллюстрирует технику использования модуля F-Anti. Если Вы скомпилируете эту программу в файл testanti.exe, то после команды testanti на экране появится сообщение

Установлена защита файла TESTANII.EXE. при первом запуске программы и В файле TESTANTI.EXE вирус не обнаружен.

при каждом следующем запуске. Если запустить программу командой

testanti /noanti на экран будет выведено сообщение Контроль блокирован ключом /NOANTI.

Uses FAnti; begin

case CheckVirusResuit of

0: WriteLn(‘B файле ',ParamStr (0),' вирус не обнаружен.\*);

1: WriteLn ('Установлена защита файла ',aramStr (0) ,\*.\*);

-1: WriteLn ('Вирус удален с разрешения пользователя.');

-2: WriteLn ('Вирус удален автоматически.');

-3: WriteLn С Контроль блокирован ключом /NOANTI.\*);

-4: WriteLn С Вирус расположен в начале \*+

' файла - удаление невозможно.')

end

end.

**4. Защита существующих ехе-файлов**

Итак, «вакцинация» вновь разрабатываемых программ не представляет особой сложности. А как защитить уже существующую ЕХЕ-программу? Для этого существуют две возможности: либо использовать отдельную программу, которая хранит ключевую информацию и осуществляет проверку по требованию пользователя (такая программа обсуждается в следующем разделе), либо пристыковать к защищаемой программе небольшой код «вирусного фага»-тогда проверка будет осуществляться автоматически при каждом запуске программы. В этом разделе мы обсудим технику создания и внедрения в защищаемую. программу антивируса—фага, т.е. небольшой ассемблерной программы, которая использует тот же механизм перехвата управления, что и обычный вирус, но осуществляет нужную защиту программы.

Программа – фаг устанавливается специальной программой-установщиком и к моменту начала своей работы уже должна иметь в своем распоряжении эталонный заголовок файла. Получив управление, фаг проверяет заголовок соответствующего файла и, если обнаружены изменения, сообщает об этом пользователю и удаляет вирус. После окончания работы фаг передает управление защищаемой программе. Поскольку фаг пишется целиком на ассемблере, его работа протекает очень быстро без заметного замедления загрузки программы. Если Вы установите такой фаг на большую часть часто используемых программ, Ваш компьютер будет защищен очень надежно.

Чтобы правильно спроектировать фаг, нужно хорошо представлять себе механизм запуска ЕХЕ-программ. Стандартный загрузчик ДОС реализует следующую последовательность действий при запуске программы.

1) Создается префикс программного сегмента PSP. Обычно для этого используется функция ДОС $26.

2) В некоторую локальную область памяти считываются начальные 28 байт заголовка ЕХЕ-файла, соответствующие структуре данных HeadExeType.

3) Определяется размер загружаемой части файла по формуле

LengExe = (PageCnt-l)\*512 + PartPag

4) Определяется файловое смещение загружаемой части:

SeekExe = HdrSize\*16

5) Выбирается сегментный адрес StartSeg для размещения программы.

Обычно StartSeg = Segment {PSP) +16, т.е. программа размещается сразу за PSP, который имеет длину 256 байт (16 параграфов).

6) Считывается загружаемая часть программы в непрерывную область памяти длиной LengExe, начинающуюся по адресу StartSeg:0000.

7) Указатель файла устанавливается на начало таблицы перемещения TablOff,

8) Для каждого элемента перемещения (этих элементов ReloCnt):

• считывается элемент как два 16-битных слова IternOfs, ItemSeg;

• вычисляется ReloSeg === StartSeg+ltemOfs, т.е. сегментная часть смещения абсолютного адреса перемещаемой ссылки;

• извлекается слово по адресу ReloSeg:ltemOfs-сегментная часть перемещаемой ссылки;

• к этому слову прибавляется StartSeg (осуществляется так называемая привязка сегмента);

• результат помещается обратно по адресу ReloSeg:ltemOfs.

9) Выделяется память за концом программы в соответствии со значениями

MinMem и МахМет.

10) Инициируются регистры и запускается программа:

• регистры ES и DS получают значение сегмента, в котором располагается PSP',

• регистр АХ отражает корректность идентификаторов дисков в командной строке (при нормальном запуске содержит 0);

• SS = StartSeg+ReloSS;

• SP = ExeSP;

• CS == StartSeg^ReloCS;

• IP= Exelp;

• содержимое остальных регистров не имеет значения. Регистры сегмента кода CS и указателя инструкций IP обычно инициируются следующими тремя командами:

PUSH StartSeg+ReloCs

PUSH Exelp

RETF

(команда RETF дальнего возврата из подпрограммы извлекает из стека два слова-смещение и сегмент адреса перехода-и помещает их соответственно в IP и CS).

Таким образом, сразу после получения управления фаг должен сохранить значения регистров АХ и DS и поместить в DS значение собственного сегмента данных. На практике сегмент данных в коротких ассемблерных программах обычно совпадает с сегментом кода, т.е. программа и данные размещаются в одном сегменте. Сегмент стека SS можно не изменять, т.к. программа-установщик фага должна позаботиться о том, чтобы стек не разрушил код самого фага, и соответствующим образом настроить ReloSS и/или ExelP. Обычно в ЕХЕ-программе начальное значение ReloSS таково, что стек размещается сразу за концом программы, т.е. в том месте, куда программа-установщик помещает "код фага. Длина стека ExeSP как правило более чем достаточна для того, чтобы работа фага со стеком не привела к разрушению кода фага, поэтому в большинстве случаев установщик оставляет начальные значения ReloSS и ExeSP без изменения.

**4.1. Описание программ SetFag.pas и Fag.asm**

В этом параграфе описываются программы SetFag.pas и Fag.asm, с помощью которых реализуется описанный выше механизм защиты. Программа SetFag (прил.П8.1) осуществляет установку фага, а программа Fag.asm (прил.П8.2) содержит сам фаг. Если Вы захотите воспользоваться предлагаемыми программами, откомпилируйте Турбо Ассемблером файл Fag.asm командой

tasm fag /L

В ходе компиляции на экран будет выведено 13 предупреждений вида

Warning\* Open procedure: XXXXXX а в конце сводка:

Error message: None Warning message: 13 Passes: I Remainig memory: XXXК

Если в строке Error message вместо None указано число обнаруженных ошибок, просмотрите файл листинга компиляции fag.lst, отыщите в нем •сообщения компилятора об ошибках и устраните их. Затем преобразуйте полученный файл fag.obj в программу Fag.prg командой

tiink fag, fag.prg

Компоновщик должен сообщить Warning: No stack

Замечу, что программа Fag.prg не может работать самостоятельно без предварительной настройки установщиком SetFag.exe, поэтому в целях предосторожности ей присваивается нестандартное расширение PRG. Для установки защиты на любой ЕХЕ-фаил следует дать команду

setfag NAME,

где NAME-имя защищаемого файла. Иными словами, имя файла передается программе установки фага SetFag.exe с помощью параметров запуска. В имени NAME можно опускать стандартное расширение ЕХЕ, а также разрешается указать маршрут поиска файла и/или символы—заменители ДОС «\*» и«?» для определения группового имени-в этом случае защита будет установлена на каждый файл, соответствующий групповому имени. Например, команда

setfag d:\mydir \*

означает требование установить защиту на все ЕХЕ-файлы из каталога MYDIR на диске D.

Перед установкой зашиты программа осуществляет серию проверок файла. Она проверяет заголовок файла и блокирует установку защиты, если первые два байта заголовка не соответствуют сигнатуре «MZ» (признаку ЕХЕ-файла). Кроме того, она проверяет «хвост» файла с тем, чтобы убедиться в отсутствии кода фага, и блокирует повторную установку защиты на уже защищенный файл. Далее, защита не устанвливается также в том случае, если длина загружаемой части файла станет слишком большой (превысит доступную память). Если в конце файла обнаружена незагружаемая часть, программа информирует об этом пользователя и запрашивает у него подтверждение на установку за-щиты. После завершения всех проверок программа создает резервную копию исходного файла с расширением ВАК. Создание ВАК-файла можно запретить, если команду вызова дополнить ключом /NOBAK, например setfag myprog /nobak. Для защиты используется ключ, соответствующий такой структуре данных:

Type

HeadType = record

case Byte of

1:(Sign : Word; {Сигнатура 'MZ' = $5MD}

PartPag: Word; {Часть неполного сектора}

PageCnt: Word; {Количество секторов}

ReloCnt: Word; {Количество элементов в таблице перемещения}

HdrSize: Word; {Длина заголовка в параграфах}

MinMem : Word; {Минимальный размер кучи}

МахМет : Word); {Максимальный размер кучи}

end.

Разумеется, фаг нельзя устанавливать на файлы, защищенные средствами модуля F\_Anti так как в этом случае процедура CheckFile этого модуля обнаружит изменение заголовка и удалит фаг. Кроме того, фаг пристыковывается в конец программы и, следовательно, не может защищать крупные программы. Последнее обстоятельство контролируется установщиком SetFag.

ReloSS : Word; {Начальное значение сегмента стека SS}

ExeSP : Word; {Начальное значение указателя стека SP}

ChkSum : Word; {Контрольная сумма всех слов файла}

ExelP : Word; {Смещение точки запуска программы}

ReloCS : Word; {Начальное значение сегмента кода CS});

2:(W: array [1..12] of Word) end;

TAVir = record

Head24: HeadType; {24 байта эталонного заголовка}

Starts: Word; {Относительный сегмент}

StartO: Word; {и смещение точки запуска программы} Leng24: Longint;{Длина незараженной программы минус 24 байта}

Key : Word; {Ключ шифровки}

end;

Как видим, этот ключ-несколько отличается от использованного в модуле F\_Anti: сохраняются только 24 байта заголовка (вряд ли вирус изменит смещение таблицы TablOff и номер оверлея Overlay), исключено ненужное теперь поле HFf добавлены поля StartS и StartO для запоминания относительного адреса точки запуска защищаемой программы. Поле Key по-прежнему содержит шифр для защиты ключа. Суммарная длина ключа SizeOf {TAVir) составляет 34 байта.

Процесс установки защиты состоит из следующих этапов.

1) В динамическую память считывается код фага из файла FAG.PRG. Вы можете создать свой вариант фага и заставить программу SetFag использовать его, если в команду запуска установщика добавите ключ /F: NameFag.Ext, где NameFag,Ext-имя и расширение файла, содержащего разработанный Вами фаг. В этом случае учтите, что SetFag помещает 34—байтный ключ в самое начало кода фага (см. листинг FAG. ASM) и поэтому при считывании из файла пропускает 34 байта от начала его загружаемой части. Выделение кода фага в отдельный P^G—файл понадобилось мне на этапе разработки и отладки кода фага. Я решил сохранить возможность загрузки кода из внешнего файла для того, чтобы Вы смогли при желании поэкспериментировать с этим кодом.

2) В поле Head24 переменной НН типа TAVir считывается заголовок ЕХЕ-файла и осуществляется настройка ключа НН: в полях StartS и StartO запоминается относительный адрес точки запуска защищаемой программы; вычисляется файловое смещение LS в параграфах, соответствующее полной длине файла и выровненное на границу параграфа - с этим смещением от начала файла в него будет помещен ключ и тело фага (выравнивание на границу параграфа необходимо для того, чтобы обеспечить корректность внутрисегментной адресации кода фага); в ReloCS помещается новое значение относительного сегмента точки запуска фага, а в ExelP-смещение этой точки; рассчитывается новое значение длины загружаемой части файла с учетом ключа и тела фага и соответствующим образом изменяются поля PageCnt и PartPag; проверяются и при необходимости корректируются поля MinMem и ExeSP так, чтобы стек не разрушил код фага.

3) В начало ЕХЕ-файла записывается новый заголовок HH.Head24, затем осуществляется смещение файлового указателя на 15\*16 байт от начала файла и в него записывается зашифрованный ключ и тело фага. Ассемблерная программа FAG. ASM работает следующим образом.

Сразу после получения управления фаг сохраняет в стеке регистр АХ, запоминает в переменной PSP значение регистра сегмента данных DS (в этот момент он указывает на префикс программного сегмента) и помещает в DS сегмент кода CS (данные и код фага расположены в одном сегменте). Кроме того, в переменной SPO запоминается вершина стека, а в CSO-сегмент кода фага. Затем вычисляется абсолютный сегмент точки запуска защищаемой программы (как уже говорилось, он равен PSP+16) и найденное значение помещается в StartS-таким образом готовится запуск защищаемой программ.

Вся основная работа фага запрограммирована в серии последовательно вызываемых процедур (при разработке фага использовался метод нисходящего программирования). Вначале с помощью процедуры GetExeNome фаг определяет полное имя защищаемого ЕХЕ- файла. Для этого используется то обстоятельство, что в версиях ДОС 3.0 и выше стандартный загрузчик помещает полное имя загружаемого файла в расширенное окружение ДОС. Окружение ДОС - это область памяти длиной до 32 Кбайт, в которой ДОС сохраняет переменные окружения типа COMSPEC, PATH, PROMPT и т.п. Каждая переменная окружения представляет собой текстовую строку, составленную из кодов ASCII, в конце которой ставится байт 0 как признак конца строки - фирма IBM называет такой код ASCIIZ (Z - Zero, ноль). Переменные окружения располагаются в памяти последовательно друг за другом. В конце «стандартной» части окружения (эта часть поддерживается и в ранних версиях ДОС) ставится дополнительный нулевой байт. За стандартной частью следует расширенная часть, куда загрузчик новых версий ДОС помещает полное имя файла (с указанием диска и маршрута поиска) и, возможно, параметры обращения к программе. Таким образом, чтобы найти имя файла, нужно отыскать в окружении ДОС два ноля подряд - это признак начала расширенной части окружения. Слово, следующее за этим признаком, содержит количество переменных в расширенной части, за ним помещаются сами переменные. Например, в терминах ассемблера структура окружения может быть такой:

db \*COMSPEC==C:\COMMAND .СОМ ',0 ; Переменная

COMSPEC db 'PATH=C:\;C:\DOS;D:\TP\*,0 ; Переменная

PATH db \* PROMPT==$p$g \* , 0; Переменная

PROMPT db 0 ; Признак конца

В этом месте кончается стандартная часть окружения и начинается его расширенная часть (только для ДОС 3.0 и выше!).

dw 2 ; Количество переменных в расширенной части

db 'D:\MYDIR\SETFAG .ЕХЕ ',0 ;Имя файла

db \*/NOBAK\*,0 ; Параметр вызова

Перед передачей управления программе загрузчик копирует окружение в отдельную область памяти и помещает сегмент этой области в PSP (в слово со смещением 44 байта от начала PSP).

В заключение следует сказать, что программы SetFag и Fag.asm не являются эталоном. Просто мне показалось, что такой способ организации защиты ЕХЕ- файлов будет достаточно удобным в использовании и эффективным в работе. Действительно, тестовые заражения программ специально разработанным вирусом, а также вирусом Yankee показали, что фаг успешно выполняет свои функции.

Поскольку программу Fag.asm без особого труда можно изменить, существует потенциальная опасность, что этот материал может быть использован для разработки вирусов. Я очень надеюсь, что к Вам, уважаемый читатель, это не относится.

**4.2. Программа AntiVir**

Итак, мы рассмотрели способ, позволяющий придать вновь создаваемой программе свойства самоконтроля. А как быть с СОМ—файлами или защитить громоздкую программу? Кроме того, существуют вирусы, которые поражают ^е файлы, а загрузочные секторы дисков. Для таких вирусов (их называют загрузочными) контроль PSP может оказаться неэффективным. Одним из возможных способов решения задачи является разработка специальной программы, которая проверяет первый сектор наиболее важных ЕХЕ и СОМ-файлов при каждом включении ПК. Эта же программа может проверить главный загрузочный сектор или даже все загрузочные секторы на всех дисках, чтобы убедиться в отсутств1Щ загрузочных вирусов, а при их обнаружении удалить их.

Мною разработана программа AntiVir (прил.П9.1), реализующая описанные функции. Эта программа создает и поддерживает архив загрузочных секторов и секторов PSP. Она может работать в двух режимах-автоматическом и диалоговом.

В автоматическом режиме AntiVir проверяет текущее состояние загрузочных секторов и первых секторов для заданных файлов и сравнивает их с эталонными копиями, хранящимися в архиве. В случае обнаружения отличий программа сообщает пользователю об этом и предлагает восстановить эталонное состояние соответствующих секторов.

В диалоговом режиме программа предоставляет пользователю возможность просмотреть и скорректировать список проверяемых файлов. Диалоговая часть реализована с помощью объектно—ориентированной библиотеки Turbo Vision. Выбор нужного режима осуществляется автоматически: если программа вызывается командой ДОС

ANTIVIR

она переходит к диалогу с пользователем, если командой

ANTIVIR /AUTO

реализуется режим автоматического контроля.

При контроле загрузочных секторов программа использует меры, позволяющие ей обнаруживать так называемые вирусы—невидимки. Такие вирусы контролируют обращение к функциям и прерываниям ДОС, а также к прерыванию $13 BIOS и при попытке чтения зараженного вирусом участка диска «подсовывают» программе сохраненную вирусом копию незараженного участка. Единственным способом обнаружения таких вирусов является непосредственное обращение к контроллеру диска или прямой вызов прерывания $13в постоянной памяти BIOS. Поскольку BIOS обычно учитывает особенности конкретного контроллера диска, второй путь будет более простым, однако для его реализации необходимо каким-то образом определить начало в BIOS программы, обрабатывающей прерывание $13 (сразу после загрузки ДОС вектор $13 перехватывается программами 1ВМ10.СОМ и показывает на резидентную в оперативной памяти часть ДОС). Для определения «чистого» вектора $13 в программе используется мультиплексное прерывание $2F, которое для подфункции $13 возвращает в регистрах DS'.DX требуемый адрес. Поскольку при этом прежнее содержимое регистров DS'.DX автоматически помещается в вектор $13, это прерывание необходимо вызывать дважды подряд с промежуточным запоминанием регистров DS'.DX в стеке (см. процедуру BuildArch в программе ANTIVIR). В ходе контроля загрузочных секторов программа переназначает обычно пустующий вектор $62 так, чтобы он указывал на вход в обработчик $13, и использует затем этот вектор для скрытого от вируса чтения секторов жесткого диска. К сожалению этот прием нельзя использовать для контроля архивных файлов, так как в последнее время широкое распространение получили программы динамического сжатия/раскрытия информации (самыми популярными из таких программ являются Double Space и Stacker). Контроль динамически сжатых дисков непосредственным чтением секторов невозможен, так как в этом случае отключается не только возможный вирус, но и резидентная программа—деархиватор. В результате не удается найти начальный сектор защищаемого файла и проверить PSP программы.

**5. Некоторые рекомендации**

1) Вставляйте имя модуля F\_Anti в предложение Uses-«вакцинация» программы при ее рождении гарантирует стойкий иммунитет на весь срок ее эксплуатации.

2) Если Вы часто используете ЕХЕ-файлы, содержащие инструментальные программы (например, пс.ехе, lexicon.exe и т.п.), рекомендую поставить на них фаг Fag.prg. Кстати, если Вы устанавливаете фаг на программу, защищенную средствами модуля F\_Anti, встроенный в нее контроль обнаружит инородное тело фага.

3) Все СОМ—файлы операционной программы и слишком громоздкие ЕХЕ-файлы следует защитить программой Antivir. Вызов этой программы в автоматическом режиме (с ключом /AUTO) полезно включить в файл AUTOEXEC.BAT.

4) Старайтесь как можно чаще архивировать на дискетах жизненно важные для Вас (и Вашего компьютера) файлы.