```
.text:08000C0D                jmp     loc_8000D18

.text:08000D18 loc_8000D18:  ; CODE XREF: .text:08000C0D?j
.text:08000D18                dec     eax
.text:08000D19                jns     short loc_8000D53
.text:08000D1B                jmp     short loc_8000D2B

.text:08000D53 loc_8000D53:  ; CODE XREF: .text:08000D19?j
.text:08000D53                inc     eax
.text:08000D54                mov     [ebp+8000466h], eax
.text:08000D5A                mov     edx, eax
.text:08000D5C                jmp     short loc_8000D6C
```

Кстати говоря, рассмотренный нами алгоритм не совсем корректен. Цепочка NOP может встретиться в любом месте программы (например, внутри функции), и тогда зараженный файл перестанет работать. Чтобы этого не произошло, некоторые вирусы выполняют ряд дополнительных проверок, в частности убеждаются, что NOP расположены между двумя функциями, опознавая их по командам пролога/эпилога.

Внедрение в секцию данных осуществляется еще проще. Вирус ищет длинную цепочку нулей, разделенную читабельными (точнее – printable) ASCII-символами и, найдя таковую, полагает, что он находится на ничейной территории, образовавшейся в результате выравнивания текстовых строк. Поскольку текстовые строки все чаще располагаются в секции .rodata, доступной лишь на чтение, вирус должен быть готов сохранять все модифицируемые им ячейки на стеке и/или динамической памяти.

Забавно, но вирусы этого типа достаточно трудно обнаружить. Действительно, наличие нечитабельных ASCII-символов между текстовыми строками – явление вполне нормальное. Может быть, это смещения или еще какие структуры данных, на худой конец – мусор, оставленный линкером!

Взгляните на рисунок 5, приведенный ниже. Согласитесь, что факт зараженности файла вовсе не так очевиден:

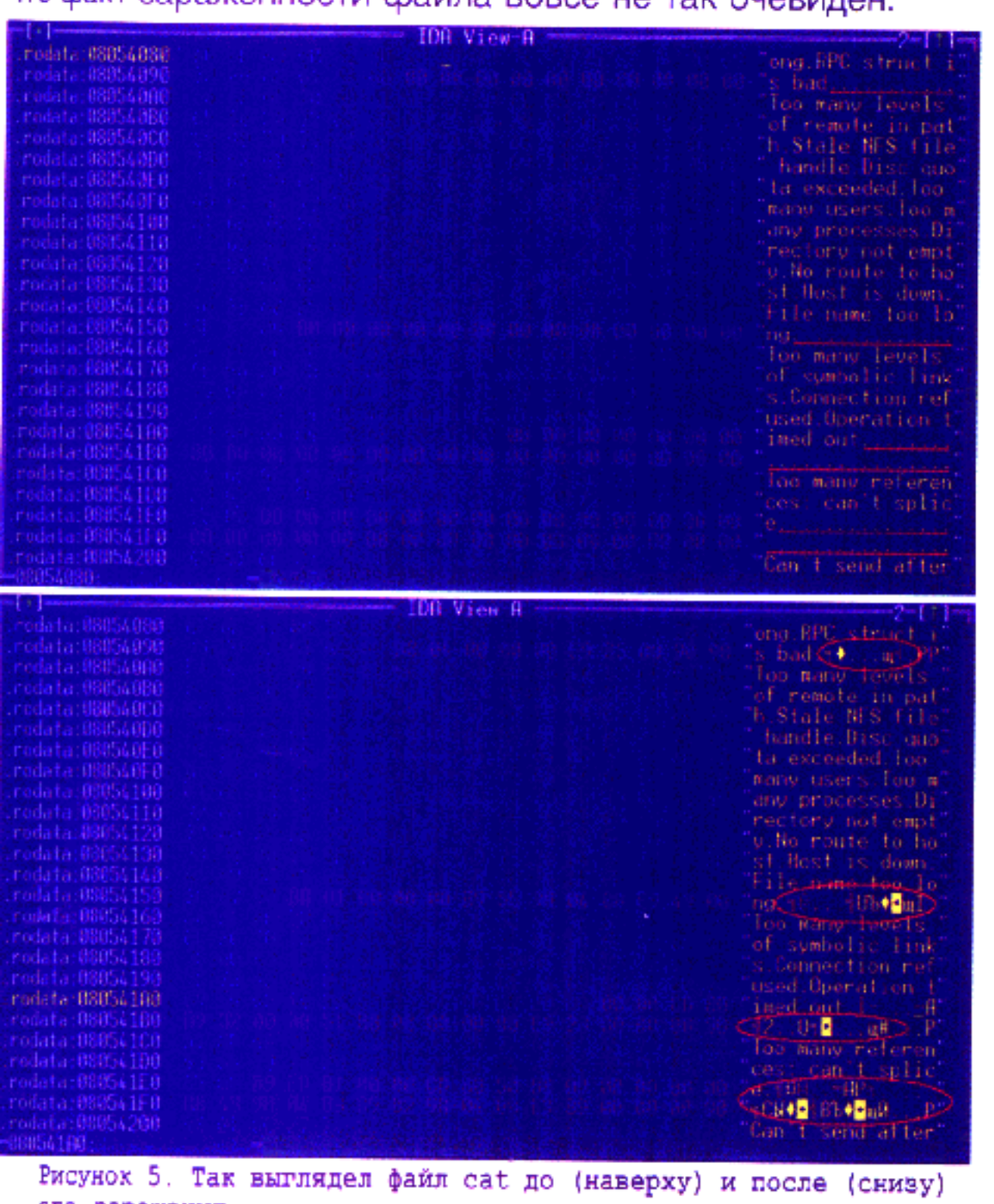

Рисунок 5. Так выглядел файл cat до (наверху) и после (снизу) его заражения

Исследователи, имеющие некоторый опыт работы с IDA, здесь, возможно, возразят: мол, какие проблемы? Подогнал курсор к первому символу, следующему за концом ASCIIZ-строки, нажал на <C>, и... дизассемблер мгновенно распахнул код вируса, живописно вплетенный в текстовые строки (см. листинг 7). На самом деле так случается только в теории. Среди нечитабельных символов вируса присутствуют и читабельные тоже. Эвристический анализатор IDA, ошибочно приняв последние за «настоящие» текстовые строки, просто не позволит их дизассемблировать. Ну, во всяком случае до тех пор, пока они явно не будут «обезличены» нажатием клавиши <U>. К тому же вирус может вставлять в начало каждого своего фрагмента специальный символ, являющийся частью той или иной машинной команды и сбивающий дизассемблер с толку. В результате IDA дизассемблирует всего лишь один-единственный фрагмент вируса (да и тот некорректно), после чего заткнется, подталкивая нас к индуктивному выводу, что мы имеем дело с легальной структурой данных, и зловредный машинный код здесь отродясь не ночевал.

Увы! Какой бы могучей IDA ни была, она все-таки не всесильна, и над всяким полученным листингом вам еще предстоит поработать. Впрочем, при некотором опыте дизассемблирования многие машинные команды распознаются в HEX-дампе с первого взгляда (пользуясь случаем, отсылаю вас к «Технике и философии хакерских атак/дизассемблирование в уме», ставшей уже библиографической редкостью, т.к. ее дальнейших переизданий уже не планируется):

Листинг 7. Фрагмент файла, зараженного вирусом UNIX.NuxBe.jullet, «0размазывающим» себя по секции данных

```
.rodata:08054140 aFileNameTooLon db 'File name too long',0
.rodata:08054153 ; ---------------------------------------
.rodata:08054153                 mov     ebx, 1
.rodata:08054158                 mov     ecx, 8049A55h
.rodata:08054158                 jmp     loc_80541A9
.rodata:08054160 ; ---------------------------------------
.rodata:08054160 aTooManyLevelsO db 'Too many levels ↵
    of symbolic links',0
.rodata:08054182 aConnectionRefu db 'Connection refused',0
.rodata:08054195 aOperationTimed db 'Operation timed out',0
.rodata:080541A9 ; ---------------------------------------
.rodata:080541A9 loc_80541A9:
.rodata:080541A9                 mov     edx, 2Dh
.rodata:080541AE                 int     80h             ; LINUX -
.rodata:080541B0                 mov     ecx, 51000032h
.rodata:080541B5                 mov     eax, 8
.rodata:080541BA                 jmp     loc_80541E2
.rodata:080541BA ; ---------------------------------------
.rodata:080541BF                 db      90h ; P
.rodata:080541C0 aTooManyReferen db 'Too many references: ↵
    can',27h,'t splice',0
.rodata:080541E2 ; ---------------------------------------
.rodata:080541E2 loc_80541E2:
.rodata:080541E2                 mov     ecx, 1FDh
.rodata:080541E7                 int     80h     ; LINUX - sys_creat
.rodata:080541E9                 push    eax
.rodata:080541EA                 mov     eax, 0
.rodata:080541EF                 add     [ebx+8049B43h], bh
.rodata:080541F5                 mov     ecx, 8049A82h
.rodata:080541FA                 jmp     near ptr unk_8054288
.rodata:080541FA ; ---------------------------------------
.rodata:080541FF                 db  90h ; P
.rodata:08054200 aCanTSendAfterS db 'Can',27h,'t send ↵
    after socket shutdown',0
```

Однако требуемого количества междустрочных байт удается наскрести далеко не во всех исполняемых фай-

лах, и тогда вирус может прибегнуть к поиску более или менее регулярной области с последующим ее сжатием. В простейшем случае ищется цепочка, состоящая из одинаковых байт, сжимаемая по алгоритму RLE. При этом вирус должен следить за тем, чтобы не нарваться на мину перемещаемых элементов (впрочем, ни один из известных автору вирусов этого не делал). Получив управление и совершив все, что он хотел совершить, вирус забрасывает на стек распаковщик сжатого кода, отвечающий за приведение файла в исходное состояние. Легко видеть, что таким способом заражаются лишь секции, доступные как на запись, так и на чтение (т.е. наиболее соблазнительные секции .rodata и .text уже не подходят, ну разве что вирус отважится изменить их атрибуты, выдавая факт заражения с головой).

Наиболее настырные вирусы могут поражать и секции неинициализированных данных. Нет, это не ошибка, такие вирусы действительно есть. Их появление объясняется тем обстоятельством, что полноценный вирус в «дырах», оставшихся от выравнивания, разместить все-таки трудно, но вот вирусный загрузчик туда влезает вполне. Секции неинициализированных данных, строго говоря, не только не обязаны загружаться с диска в память (хотя некоторые UNIX их все-таки загружают), но могут вообще отсутствовать в файле, динамически создаваясь системным загрузчиком на лету. Однако вирус и не собирается искать их в памяти! Вместо этого он вручную считывает их непосредственно с самого зараженного файла (правда, в некоторых случаях доступ к текущему выполняемому файлу предусмотрительно блокируется операционной системой).

На первый взгляд, помещение вирусом своего тела в секции неинициализированных данных ничего не меняет (если даже не демаскирует вирус), но при попытке поимки такого вируса за хвост он выскользнет из рук. Секция неинициализированных данных визуально ничем не отличается от всех остальных секций файла, и содержать она может все, что угодно: от длинной серии нулей, до копирайтов разработчика. В частности, создатели дистрибутива FreeBSD 4.5 именно так и поступают (см. листинг 8).

Листинг 8. Так выглядит секция .bss большинства файлов из комплекта поставки Free BSD

```
0000E530:  00 00 00 00 FF FF FF FF | 00 00 00 00 FF FF FF FF
0000E540:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000E550:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000E560:  00 47 43 43 3A 20 28 47 | 4E 55 29 20 63 20 32 2E   GCC: (GNU) c 2.
0000E570:  39 35 2E 33 20 32 30 30 | 31 30 33 31 35 20 28 72   95.3 20010315 (r
0000E580:  65 6C 65 61 73 65 29 20 | 5B 46 72 65 65 42 53 44   elease) [FreeBSD
...
0000F2B0:  4E 55 29 20 63 20 32 2E | 39 35 2E 33 20 32 30 30   NU) c 2.95.3 200
0000F2C0:  31 30 33 31 35 20 28 72 | 65 6C 65 61 73 65 29 20   10315 (release)
0000F2D0:  5B 46 72 65 65 42 53 44 | 5D 00 08 00 00 00 00 00   [FreeBSD] ■
0000F2E0:  00 00 01 00 00 00 30 31 | 2E 30 31 00 00 00 08 00     ©   01.01   ■
```

Ряд дизассемблеров (и IDA Pro в том числе) по вполне логичным соображениям не загружает содержимое секций неинициализированных данных, явно отмечая это обстоятельство двойным знаком вопроса (см. листинг 9). Приходится исследовать файл непосредственно в HIEW или любом другом HEX-редакторе, разбирая a.out/ELF-формат «вручную», т.к. популярные HEX-редакторы его не поддерживают. Скажите честно: готовы ли вы этим реально заниматься? Так что, как ни крути, а вирусы этого типа имеют все шансы на выживание, пусть массовых эпидемий им никогда не видать.

Листинг 9. Так выглядит секция .bss в дизассемблере IDA Pro и большинстве других дизассемблеров

```
.bss:08057560  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:08057570  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:08057580  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:08057590  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:080575A0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:080575B0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:080575C0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:080575D0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
.bss:080575E0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? "????????????????"
```

Заражение посредством расширения кодовой секции файла

Наибольшую скрытность вирусу обеспечивает внедрение в кодовую секцию заражаемого файла, находящуюся глубоко в середине последнего. Тело вируса, сливаясь с исходным машинным кодом, виртуально становится совершенно не отличимым от «нормальной» программы, и обнаружить такую заразу можно лишь анализом ее алгоритма (см. также «Основные признаки вирусов»).

Безболезненное расширение кодовой секции возможно лишь в ELF- и COFF-файлах (под «безболезненностью» здесь понимается отсутствие необходимости в перекомпиляции файла-жертвы), и достигается оно за счет того замечательного обстоятельства, что стартовые виртуальные адреса сегментов/секций отделены от их физических смещений, отсчитываемых от начала файла.

Алгоритм заражения ELF-файла в общем виде выглядит так (внедрение в COFF-файлы осуществляется аналогичным образом):

- вирус открывает файл и, считав его заголовок, убеждается, что это действительно ELF;
- заголовок таблицы секций (Section Header Table) перемещается вниз на величину, равную длине тела вируса. Для этого вирус увеличивает содержимое поля e_shoff, оккупирующего 20h – 23h байты ELF-заголовка, (примечание: заголовок таблицы секций, равно как и сами секции, имеет значение только для компоновочных файлов, загрузчик исполняемых файлов их игнорирует, независимо от того, присутствуют они в файле или нет);
- просматривая Program Header Table, вирус находит сегмент, наиболее предпочтительный для заражения (т.е. тот сегмент, в который указывает точка входа);
- длина найденного сегмента увеличивается на величину, равную размеру тела вируса. Это осуществляется путем синхронной коррекции полей p_filez и p_memz;
- все остальные сегменты смещаются вниз, при этом поле p_offset каждого из них увеличивается на длину тела вируса;
- анализируя заголовок таблицы секций (если он только присутствует в файле), вирус находит секцию, наиболее предпочтительную для заражения (как правило, заражается секция, находящаяся в сегменте последней: это избавляет вируса от необходимости перемещения всех остальных секций вниз);
- размер заражаемой секции (поле sh_size) увеличивается на величину, равную размеру тела вируса;

- все хвостовые секции сегмента смещаются вниз, при этом поле sh_offset каждой из них увеличивается на длину тела вируса (если вирус внедряется в последнюю секцию сегмента, этого делать не нужно);
- вирус дописывает себя к концу заражаемого сегмента, физически смещая содержимое всей остальной части файла вниз;
- для перехвата управления вирус корректирует точку входа в файл (e_entry) либо же внедряет в истинную точку входа jmp на свое тело (впрочем, методика перехвата управления – тема отдельного большого разговора).

Прежде чем приступить к обсуждению характерных «следов» вирусного внедрения, давайте посмотрим, какие секции в каких сегментах обычно бывают расположены. Оказывается, схема их распределения далеко не однозначна и возможны самые разнообразные вариации. В одних случаях секции кода и данных помещаются в отдельные сегменты, в других – секции данных, доступные только на чтение, объединяются с секциями кода в единый сегмент. Соответственно и последняя секция кодового сегмента каждый раз будет иной.

Большинство файлов включает в себя более одной кодовой секции, и располагаются эти секции приблизительно так:

Листинг 10. Схема расположения кодовых секций типичного файла

```
.init      содержит инициализационный код
.plt       содержит таблицу связки подпрограмм
.text      содержит основной код программы
.fini      содержит термирующий код программы
```

Присутствие секции .finit делает секцию .text не последней секцией кодового сегмента файла, как чаще всего и происходит. Таким образом, в зависимости от стратегии распределения секций по сегментам, последней секцией файла обычно является либо секция .finit, либо .rodata.

Секция .finit в большинстве своем это такая крохотная секция, заражение которой трудно оставить незамеченным. Код, расположенный в секции .finit и непосредственно перехватывающий на себя нить выполнения программой, выглядит несколько странно, если не сказать – подозрительно (обычно управление на .finit передается косвенным образом как аргумент функции atexit). Вторжение будет еще заметнее, если последней секцией в заражаемом сегменте окажется секция .rodata (машинный код при нормальном развитии событий в данные никогда не попадает). Не остается незамеченным и вторжение в конец первой секции кодового сегмента (в последнюю секцию сегмента, предшествующему кодовому сегменту), поскольку кодовый сегмент практически всегда начинается с секции .init, вызываемой из глубины стартового кода и по обыкновению содержащей пару-тройку машинных команд. Вирусу здесь будет просто негде затеряться, и его присутствие сразу же становится заметным!

Более совершенные вирусы внедряются в конец секции .text, сдвигая все остальное содержимое файла вниз. Распознать такую заразу значительно сложнее, поскольку визуально структура файла выглядит неискаженной. Однако некоторые зацепки все-таки есть. Во-первых, оригинальная точка входа подавляющего большинства файлов расположена в начале кодовой секции, а не в ее конце. Во-вторых, зараженный файл имеет нетипичный стартовый код (подробнее об этом рассказывалось в предыдущей статье). И, в-третьих, далеко не все вирусы заботятся о выравнивании сегментов (секций).

Последний случай стоит рассмотреть особо. Системному загрузчику, ничего не знающему о существовании секций, степень их выравнивания некритична. Тем не менее, во всех нормальных исполняемых файлах секции тщательно выровнены на величину, указанную в поле sh_addralign. При заражении файла вирусом последний далеко не всегда оказывается так аккуратен, и некоторые секции могут неожиданно для себя очутиться по некратным адресам. Работоспособности программы это не нарушит, но вот факт вторжения вируса сразу же демаскирует.

Сегменты выравнивать тоже необязательно (при необходимости системный загрузчик сделает это сам), однако программистский этикет предписывает выравнивать секции, даже если поле p_align равно нулю (т.е. выравнивания не требуется). Все нормальные линкеры выравнивают сегменты по крайней мере на величину, кратную 32 байтам, хотя это происходит и не всегда. Тем не менее, если сегменты, следующие за сегментом кода, выровнены на меньшую величину – к такому файлу следует присмотреться повнимательнее.

Другой немаловажный момент: при внедрении вируса в начало кодового сегмента он может создать свой собственный сегмент, предшествующий данному. И тут вирус неожиданно сталкивается с довольно интересной проблемой. Сдвинуть кодовый сегмент вниз он не может, т.к. тот обычно начинается с нулевого смещения в файле, перекрывая собой предшествующие ему сегменты. Зараженная программа в принципе может и работать, но раскладка сегментов становится слишком уж необычной, чтобы ее не заметить.

Выравнивание функций внутри секций – это вообще вещь (в смысле: вещдок – вещественное доказательство). Кратность выравнивания функций нигде и никак не декларируется, и всякий программист склонен выравнивать функции по-своему. Одни используют выравнивание на адреса, кратные 04h, другие – 08h, 10h или даже 20h! Определить степень выравнивания без качественного дизассемблера практически невозможно. Требуется выписать стартовые адреса всех функций и найти наибольший делитель, на который все они делятся без остатка. Дописывая себя в конец кодового сегмента, вирус наверняка ошибется с выравниванием адреса пролога функции (если он вообще позаботится о создании функции в этом месте!), и он окажется отличным от степени выравнивания, принятой всеми остальными функциями (попутно заметим, что определять степень выравнивания при помощи дизассемблера IDA PRO – плохая идея, т.к. она определяет ее неправильно, закладываясь на наименьшее возможное значение, в результате чего вычисленная степень выравнивания от функции к функции будет варьироваться).

Классическим примером вируса, внедряющегося в файл путем расширения кодового сегмента, является вирус Linux.Vit.4096. Любопытно, что различные авторы по-разному описывают стратегии, используемые вирусом для заражения. Так, Евгений Касперский почему-то считает, что вирус Vit записывается в начало кодовой секции заражаемого файла (http://www.viruslist.com/viruslist.html?id=3276), в то время как он размещает свое тело в конце кодового сегмента файла (http://www.nai.com/common/media/vil/pdf/mvanvoers_VB_conf 202000.pdf). Ниже приведен фрагмент ELF-файла, зараженного вирусом Vit.

Рисунок 6. Фрагмент файла, зараженного вирусом Lin/Vit. Поля, модифицированные вирусом, выделены траурной рамкой

Многие вирусы (и в частности, вирус Lin/Obsidan) выдают себя тем, что при внедрении в середину файла «забывают» модифицировать заголовок таблицы секций (либо же модифицируют его некорректно). Как уже отмечалось выше, в процессе загрузки исполняемых файлов в память системный загрузчик считывает информацию о сегментах и проецирует их содержимое целиком. Внутренняя структура сегментов его совершенно не интересует. Даже если заголовок таблицы секций отсутствует или заполнен некорректно, запущенная на выполнение программа будет исправно работать. Однако несмотря на это, в подавляющем большинстве исполняемых файлов заголовок таблицы секций все-таки присутствует, и попытка его удаления оканчивается весьма плачевно – популярный отладчик gdb и ряд других утилит для работы с ELF-файлами отказываются признать «кастрированный» файл своим. При заражении исполняемого файла вирусом, некорректно обращающимся с заголовком таблицы секций, поведение отладчика становится непредсказуемым, демаскируя тем самым факт вирусного вторжения.

Перечислим некоторые наиболее характерные признаки заражения исполняемых файлов (вирусы, внедряющиеся в компоновочные файлы, обрабатывают заголовок таблицы секций вполне корректно, в противном случае зараженные файлы тут же откажут в работе и распространение вируса немедленно прекратится):

- поле e_shoff указывает «мимо» истинного заголовка таблицы секций (так себя ведет вирус Lin/Obsidan) либо имеет нулевое значение при непустом заголовке таблицы секций (так себя ведет вирус Linux.Garnelis);
- поле e_shoff имеет ненулевое значение, но ни одного заголовка таблицы секций в файле нет;
- заголовок таблицы секций содержится не в конце файла, этих заголовков несколько или заголовок таблицы секций попадает в границы владения одного из сегментов;
- сумма длин всех секций одного сегмента не соответствует его полной длине;
- программный код расположен в области, не принадлежащей никакой секции.

Следует сказать, что исследование файлов с искаженным заголовком таблицы секций представляет собой достаточно простую проблему. Дизассемблеры и отладчики либо виснут, либо отображают такой файл неправильно, либо же не загружают его вообще. Поэтому, если вы планируете заниматься исследованием зараженных файлов не день и не два, лучше всего будет написать свою собственную утилиту для их анализа.

Сдвиг кодовой секции вниз

Трудно объяснить причины, по которым вирусы внедряются в начало кодовой секции (сегмента) заражаемого файла или создают свой собственную секцию (сегмент), располагающуюся впереди. Этот прием не обеспечивает никаких преимуществ перед записью своего тела в конец кодовой секции (сегмента) и к тому же намного сложнее реализуется. Тем не менее, такие вирусы существуют и будут подробно здесь рассмотрены.

Наилучший уровень скрытности достигается при внедрении в начало секции .text и осуществляется практически тем же самым образом, что и внедрение в конец, с той лишь разницей, что для сохранения работоспособности зараженного файла вирус корректирует поля sh_addr и p_vaddr, уменьшая их на величину своего тела и не забывая о необходимости выравнивания (если выравнивание действительно необходимо). Первое поле задает виртуальный стартовый адрес для проекции секции .text, второе – виртуальный стартовый адрес для проекции кодового сегмента.

В результате этой махинации вирус оказывается в самом начале кодовой секции и чувствует себя довольно уверенно, поскольку при наличии стартового кода выглядит неотличимо от «нормальной» программы. Однако работоспособность зараженного файла уже не гарантируется, и его поведение рискует стать совершенно непредсказуемым, поскольку виртуальные адреса всех предыдущих секций окажутся полностью искажены. Если при компиляции программы компоновщик позаботился о создании секции перемещаемых элементов, то вирус (теоретически) может воспользоваться этой информацией для приведения впереди идущих секций в нормальное состояние, однако исполняемые файлы в своем подавляющем большинстве спроектированы для работы по строго определенным физическим адресам и потому неперемещаемы. Но даже при наличии перемещаемых элементов вирус не сможет отследить все случаи относительной адресации. Между секцией кода и секцией данных относительные ссылки практически всегда отсутствуют, и потому при вторжении вируса в конец кодовой секции работоспособность файла в большинстве случаев не нарушается. Однако внутри кодового сегмента случаи относительной адресации между секциями – скорее правило, нежели редкость. Взгляните на фрагмент дизассемблерного листинга утилиты ping, позаимствованный из UNIX Red Hat 5.0. Команду call, расположенную в секции .init, и вызываемую ею подпрограмму, находящуюся в секции .text, раз-

деляют ровно 8002180h – 8000915h == 186Bh байт, и именно это число фигурирует в машинном коде (если же вы все еще продолжаете сомневаться, загляните в Intel Instruction Reference Set: команда E8h это команда относительного вызова):

Листинг 11. Фрагмент утилиты ping, использующей, как и многие другие программы, относительные ссылки между секциями кодового сегмента

```
.init:08000910      _init          proc near
    ; CODE XREF: start+51?p
.init:08000910 E8 6B 18 00 00   call   sub_8002180
.init:08000915 C2 00 00  retn   0
.init:08000915      _init          endp

.text:08002180      sub_8002180    proc near
    ; CODE XREF: _init?p
```

Неудивительно, что после заражения файл перестает работать (или станет работать некорректно)! Но если это все-таки произошло, загрузите файл в отладчик/дизассемблер и посмотрите – соответствуют ли относительные вызовы первых кодовых секций пункту своего назначения. Вы легко распознаете факт заражения, даже не будучи специалистом в области реинжиниренга.

В этом мире ничего не дается даром! За скрытность вирусного вторжения последнему приходится расплачиваться разрушением большинства заражаемых файлов. Более корректные вирусы располагают свое тело в начале кодового сегмента – в секции .init. Работоспособность заражаемых файлов при этом не нарушается, но присутствие вируса становится легко обнаружить, т.к. секция .init редко бывает большой, и даже небольшая примесь постороннего кода сразу же вызывает подозрение.

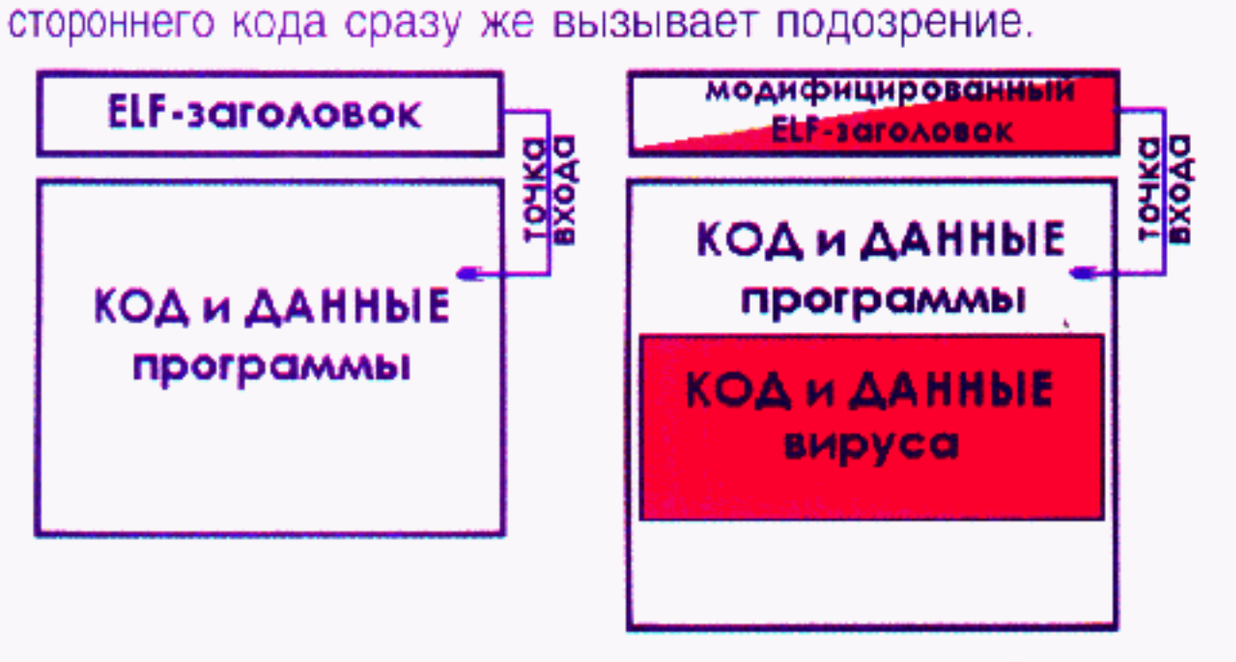

Рисунок 7. Типовая схема заражения исполняемого файла путем расширения его кодовой секции

Некоторые вирусы (например, вирус Linux.NuxBee) записывают себя поверх кодового сегмента заражаемого файла, перемещая затертую часть в конец кодовой секции (или, что более просто, в конец последнего сегмента файла). Получив управление и выполнив всю работу «по хозяйству», вирус забрасывает кусочек своего тела в стек и восстанавливает оригинальное содержимое кодового сегмента. Учитывая, что модификация кодового сегмента по умолчанию запрещена и разрешать ее вирусу не резон (в этом случае факт заражения очень легко обнаружить), вирусу приходится прибегать к низкоуровневым манипуляциям с атрибутами страниц памяти, вызывая функцию mprotect, практически не встречающуюся в «честных» приложениях.

Другой характерный признак: в том месте, где кончается вирус и начинается незатертая область оригинального тела программы, образуется своеобразный дефект. Скорее всего, даже наверняка, граница раздела двух сред пройдет посередине функции оригинальной программы, если еще не рассечет машинную команду. Дизассемблер покажет некоторое количество мусора и хвост функции с отсутствующим прологом.

Создание своей собственной секции

Наиболее честный (читай – «корректный») и наименее скрытный способ внедрения в файл состоит в создании своей собственной секции (сегмента), а то и двух секций – для кода и для данных соответственно. Разместить такую секцию можно где угодно. Хоть в начале файла, хоть в конце (вариант внедрения в сегмент с раздвижкой соседних секций мы уже рассматривали выше).

Листинг 12. Карта файла, зараженного вирусом, внедряющимся в собственноручно созданную секцию и этим себя демаскирующим (подробнее об этом рассказывалось в предыдущей статье этого цикла «Борьба с вирусами – опыт контртеррористических операций», в октябрьском номере журнала)

Name	Start	End	Align	Base	Type	Class	32	es	ss	ds	fs	gs
.init	08000910	08000918	para	0001	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.plt	08000918	08000B58	dword	0002	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.text	08000B60	080021A4	para	0003	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.fini	080021B0	080021B8	para	0004	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.rodata	080021B8	0800295B	byte	0005	publ	CONST	Y	FFFF	FFFF	0006	FFFF	FFFF
.data	0800295C	08002A08	dword	0006	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.ctors	08002A08	08002A10	dword	0007	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.dtors	08002A10	08002A18	dword	0008	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.got	08002A18	08002AB0	dword	0009	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.bss	08002B38	08013CC8	dword	000A	publ	BSS	Y	FFFF	FFFF	0006	FFFF	FFFF
.data1	08013CC8	08014CC8	qword	000A	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF

Внедрение между файлом и заголовком

Фиксированный размер заголовка a.out-файлов существенно затруднял эволюцию этого, в общем-то неплохого формата, и в конечном счете привел к его гибели. В последующих форматах это ограничение было преодолено. Так, в ELF-файлах длина заголовка хранится в двухбайтовом поле e_ehize, оккупировавшем 28h и 29h байты, считая от начала файла.

Увеличив заголовок заражаемого файла на величину, равную длине своего тела, и сместив оставшуюся часть файла вниз, вирус сможет безболезненно скопировать себя в образовавшееся пространство между концом настоящего заголовка и началом Program Header Table. Ему даже не придется увеличивать длину кодового сегмента, поскольку в большинстве случаев тот начинается с самого первого байта файла. Единственное, что будет вынужден сделать вирус, сдвинуть поля p_offset всех сегментов на соответствующую величину вниз. Сегмент, начинающийся с нулевого смещения, никуда перемещать не надо, иначе вирус не будет спроецирован в память. (Смещения сегментов в файле отсчитываются от начала файла, но не от конца заголовка, что нелогично и идеологически неправильно, зато упрощает программирование). Поле e_phoff, задающее смещение Program Head Table, также должно быть скорректировано.

Аналогичную операцию следует проделать и со смещениями секций, в противном случае отладка/дизассемблирование зараженного файла станет невозможной (хотя файл будет нормально запускаться). Существующие вирусы забывают скорректировать содержимое полей sh_offset, чем и выдают себя, однако следует быть гото-

вым к тому, что в следующих поколениях вируса этот недостаток будет устранен.

Впрочем, в любом случае такой способ заражения слишком заметен. В нормальных программах исполняемый код никогда не попадает в ELF-заголовок, и его наличие там красноречиво свидетельствует о вирусном заражении. Загрузите исследуемый файл в любой HEX-редактор (например, HIEW) и проанализируйте значение поля e_ehize. Стандартный заголовок, соответствующий текущим версиям ELF-файла, на платформе X86 (кстати, недавно переименованной в платформу Intel) имеет длину, равную 34 байтам. Другие значения в «честных» ELF-файлах мне видеть пока не доводилось (хотя я и не утверждаю, что таких файлов действительно нет – опыт работы с UNIX у меня небольшой). Только не пытайтесь загрузить зараженный файл в дизассемблер. Это бесполезно. Большинство из них (и IDA PRO в том числе) откажутся дизассемблировать область заголовка, и исследователь о факте заражения ничего не узнает!

Ниже приведен фрагмент файла, зараженного вирусом UNIX.inheader.6666. Обратите внимание на поле длины ELF-заголовка, обведенное квадратиком. Вирусное тело, начинающиеся с 34h байта, залито бордовым цветом. Сюда же направлена точка входа (в данном случае она равна 8048034h):

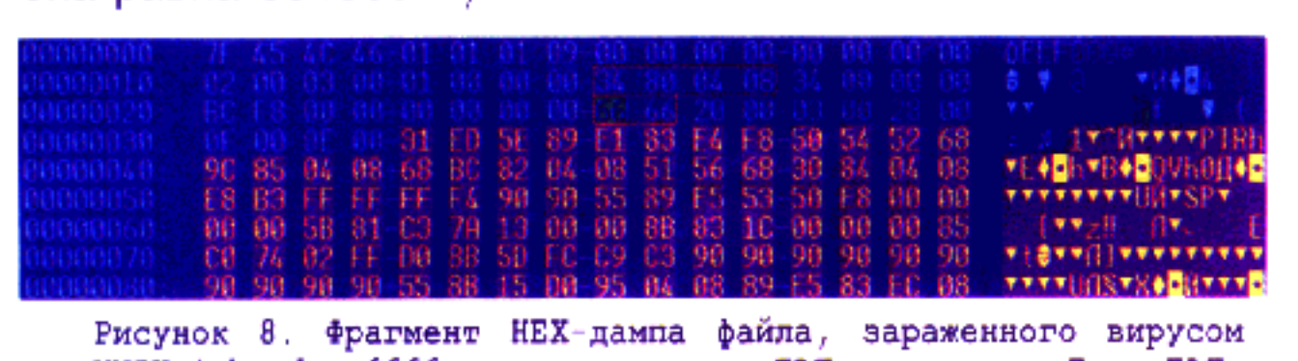

Рисунок 8. Фрагмент HEX-дампа файла, зараженного вирусом UNIX.inheader.6666, внедряющимся в ELF-заголовок. Поля ELF-заголовка, модифицированные вирусом, взяты в рамку, а само тело вируса залито бордовым цветом

Как вариант, вирус может вклиниться между концом ELF-заголовка и началом Program Header Table. Заражение происходит так же, как и предыдущем случае, однако длина ELF-заголовка остается неизменной. Вирус оказывается в «сумеречной» области памяти, формально принадлежащей одному из сегментов, но де-факто считающейся «ничейной» и потому игнорируемой многими отладчиками и дизассемблерами. Если только вирус не переустановит на себя точку входа, дизассемблер даже не сочтет нужным заругаться по этому поводу. Поэтому какой бы замечательной IDA PRO ни была, а просматривать исследуемые файлы в HIEW все-таки необходимо! Учитывая, что об этом догадываются далеко не все эксперты по безопасности, данный способ заражения рискует стать весьма перспективным. К борьбе с вирусами, внедряющимися в заголовок ELF-файлов, будьте готовы!

Перехват управления путем коррекции точки входа

Успешно внедриться в файл – это только полдела. Для поддержки своей жизнедеятельности всякий вирус должен тем или иным способом перехватить на себя нить управления. Классический способ, активно использовавшийся еще во времена MS-DOS, сводится к коррекции точки входа – одного из полей ELF/COFF/a.out заголовков файлов. В ELF-заголовке эту роль играет поле e_entry, в a.out – a_entry. Оба поля содержат виртуальный адрес (не смещение, отсчитываемое от начала файла) машинной инструкции, на которую должно быть передано управление.

При внедрении в файл вирус запоминает адрес оригинальной точки входа и переустанавливает ее на свое тело. Сделав все, что хотел сделать, он возвращает управление программе-носителю, используя сохраненный адрес. При всей видимой безупречности этой методики она не лишена изъянов, обеспечивающих быстрое разоблачение вируса.

Во-первых, точка входа большинства честных файлов указывает на начало кодовой секции файла. Внедриться сюда трудно, и все существующие способы внедрения связаны с риском необратимого искажения исполняемого файла, приводящего к его полной неработоспособности. Точка входа, «вылетающая» за пределы секции .text, – явный признак вирусного заражения.

Во-вторых, анализ всякого подозрительного файла начинается в первую очередь с окрестностей точки входа (и ею же обычно и заканчивается), поэтому независимо от способа вторжения в файл вирусный код сразу же бросается в глаза.

В-третьих, точка входа – объект пристального внимания легиона дисковых ревизоров, сканеров, детекторов и всех прочих антивирусов.

Использовать точку входа для перехвата управления – слишком примитивно и, по мнению большинства создателей вирусных программ, даже позорно. Современные вирусы осваивают другие методики заражения, и закладываться на анализ точки входа может только наивный (вот так и рождаются байки о неуловимых вирусах...).

Перехват управления путем внедрения своего кода в окрестности точки входа

Многие вирусы никак не изменяют точку входа, но внедряют по данному адресу команду перехода на свое тело, предварительно сохранив его оригинальное содержимое. Несмотря на кажущуюся элегантность этого алгоритма, он довольно капризен в работе и сложен в реализации. Начнем с того, что для сохранения оригинальной машинной инструкции, расположенной в точке входа, вирус должен определить ее длину, но без встроенного дизассемблера это сделать невозможно.

Большинство вирусов ограничивается тем, что сохраняет первые 16-байт (максимально возможная длина машинной команды на платформе Intel), а затем восстанавливает их обратно, так или иначе обходя запрет на модификацию кодового сегмента. Кто-то снабжает кодовый сегмент атрибутом Write, делая его доступным для записи (если не трогать атрибуты секций, то кодовый сегмент все равно будет можно модифицировать, но IDA PRO об этом не расскажет, т.к. с атрибутами сегментов она работать не умеет), кто-то использует функцию mprotect для изменения атрибутов страниц на лету. И тот, и другой способы слишком заметны, а инструкция перехода на тело вируса заметна без очереди!

Более совершенные вирусы сканируют стартовую процедуру заражаемого файла в поисках инструкций call или

jmp. А найдя таковую, подменяют вызываемый адрес на адрес своего тела. Несмотря на кажущуюся неуловимость, обнаружить такой способ перехвата управления очень легко. Первое и главное – вирус, в отличие от легально вызываемых функций, никак не использует переданные ему в стеке аргументы. Он не имеет никаких понятий об их числе и наличии (машинный анализ количества переданных аргументов немыслим без интеграции в вирус полноценного дизассемблера, оснащенного мощным интеллектуальным анализатором). Вирус тщательно сохраняет все изменяемые регистры, опасаясь, что функции могут использовать регистровую передачу аргументов с неизвестным ему соглашением. Самое главное – при передаче управления оригинальной функции вирус должен либо удалить с верхушки стека адрес возврата (в противном случае их там окажется два) либо вызывать оригинальную функцию не командой call, но командой jmp. Для «честных» программ, написанных на языках высокого уровня, и то и другое крайне нетипично, благодаря чему вирус оказывается немедленно разоблачен.

Вирусы, перехватывающие управление в произвольной точке программы (зачастую чрезвычайно удаленной от точки входа), выявить намного труднее, поскольку приходится анализировать довольно большие, причем заранее не определенные, объемы кода. Впрочем, с удалением от точки входа стремительно возрастает риск, что данная ветка программы никогда не получит управление, поэтому все известные мне вирусы не выходят за границы первого встретившегося им RET.

Основные признаки вирусов

Искажение структуры исполняемых файлов – характерный, но недостаточный признак вирусного заражения. Быть может, это защита хитрая такая или завуалированный способ самовыражения разработчика. К тому же некоторые вирусы ухитряются внедриться в файл практически без искажений его структуры. Однозначный ответ дает лишь полное дизассемблирование исследуемого файла, однако это слишком трудоемкий способ, требующий усидчивости, глубоких знаний операционной системы и неограниченного количества свободного времени. Поэтому на практике обычно прибегают к компромиссному варианту, сводящемуся к беглому просмотру дизассемблерного листинга на предмет поиска основных признаков вирусного заражения.

Большинство вирусов использует довольно специфический набор машинных команд и структур данных, практически никогда не встречающихся в «нормальных» приложениях. Конечно, разработчик вируса при желании может все это скрыть, и распознать зараженный код тогда не удастся. Но это в теории. На практике же вирусы обычно оказываются настолько тупы, что обнаруживаются за считанные доли секунды.

Ведь чтобы заразить жертву, вирус прежде должен ее найти, отобрав среди всех кандидатов только файлы «своего» типа. Для определенности возьмем ELF. Тогда вирус будет вынужден считать его заголовок и сравнить четыре первых байта со строкой «⌂FELF», которой соответствует ASCII-последовательность 7F 45 4C 46. Конечно, если тело вируса зашифровано, вирус использует хеш-сравнение или же другие хитрые приемы программирования, строки «ELF» в теле зараженного файла не окажется, но более чем в половине всех существующих UNIX-вирусов она все-таки есть, и этот прием, несмотря на свою изумительную простоту, очень неплохо работает.

Загрузите исследуемый файл в любой HEX-редактор и попробуйте отыскать строку «⌂ELF». В зараженном файле таких строк будет две – одна непосредственно в заголовке, другая – в кодовой секции или секции данных. Только не используйте дизассемблер! Очень многие вирусы преобразуют строку «⌂FELF» в 32-разрядную целочисленную константу 464C457Fh, которая маскирует присутствие вируса, но при переключении в режим дампа сразу же «проявляется» на экране. Ниже приведен внешний вид файла, зараженного вирусом VirTool.Linux.Mmap.443, который использует именно такую методику:

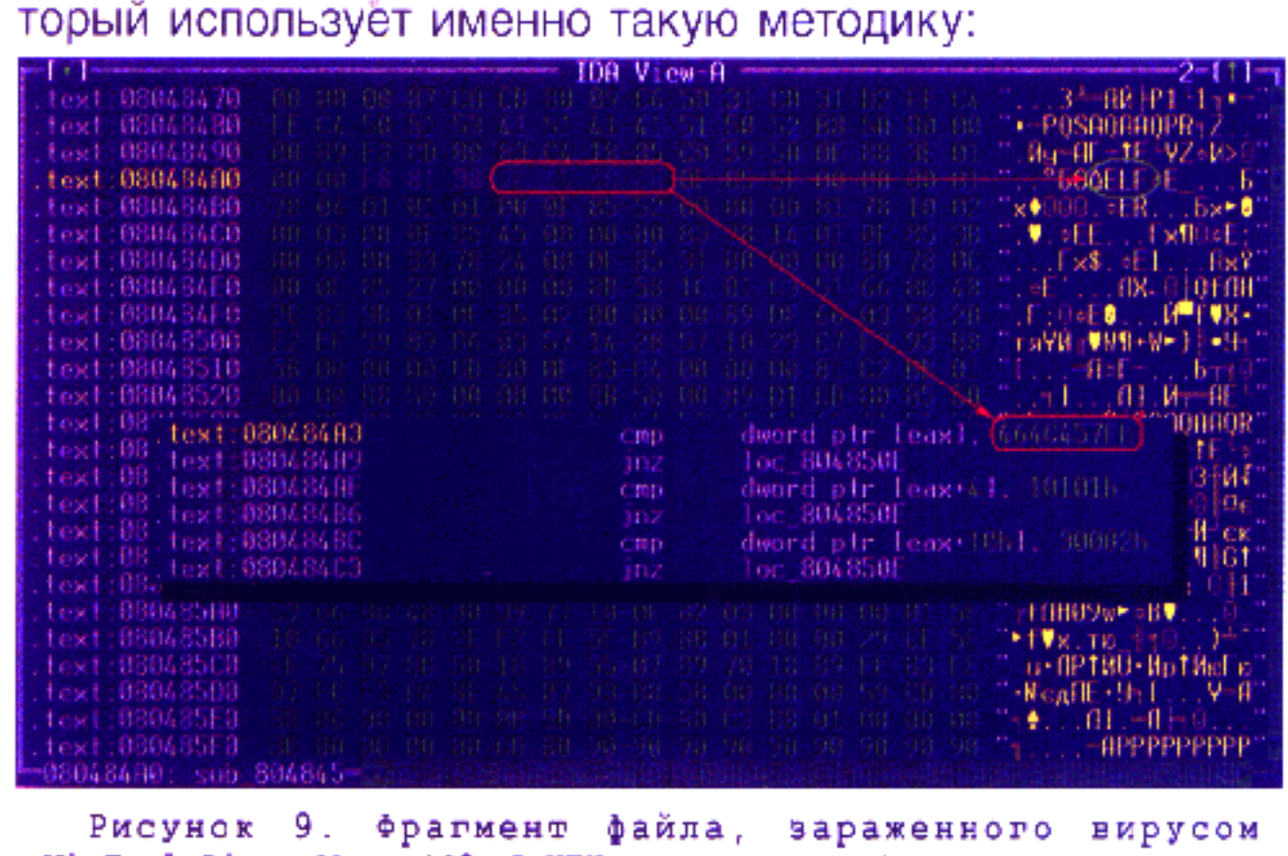

Рисунок 9. Фрагмент файла, зараженного вирусом VirTool.Linux.Mmap.443. В HEX-дампе легко обнаруживается строка «ELF», используемая вирусом для поиска жертв «своего» типа

Вирус Linux.Winter.343 (также известный под именем Lotek) по этой методике обнаружить не удается, поскольку он использует специальное математическое преобразование, зашифровывая строку «⌂ELF» на лету:

Листинг 13. Фрагмент вируса Lotek, тщательно скрывающего свой интерес к ELF-файлам

```
.text:08048473          mov     eax, 0B9B3BA81h ↵
    ; -"ELF" (минус "ELF")
.text:08048478          add     eax, [ebx] ↵
    ; первые четыре байта жертвы
.text:0804847A          jnz     short loc_804846E ↵↵
    ; → это не ELF
```

Непосредственное значение B9B3BA81h, соответствующее текстовой строке «ƃ‖|╢» (в приведенном выше листинге оно выделено жирным шрифтом), представляет собой не что иное, как строку «⌂ELF», преобразованную в 32-разрядную константу и умноженную на минус единицу. Складывая полученное значение с четырьмя первыми байтами жертвы, вирус получает ноль, если строки равны, и ненулевое значение в противном случае.

Как вариант, вирус может дополнять эталонную строку «⌂ELF» до единицы, и тогда в его теле будет присутствовать последовательность 80 BA B3 B9. Реже встречаются циклические сдвиги на одну, две, три... и семь позиций в различные стороны, неполные проверки (т.е. проверки на совпадение двух или трех байт) и некоторые другие операции – всех не перечислишь!

Более уязвимым с точки зрения скрытности является механизм реализации системных вызовов. Вирус не может позволить себе тащить за собой всю библиотеку LIBC, прилинкованную к нему статической компоновкой, поскольку существование подобного монстра трудно оставить незаметным. Существует несколько способов решения этой проблемы, и наиболее популярный из них сводится к использованию nativeAPI операционной системы. Поскольку последний является прерогативой особенностей реализации данной конкретной системы, создатели UNIX де-факто отказались от многочисленных попыток его стандартизации. В частности, в System V (и ее многочисленных клонах) обращение к системным функциям происходит через дальний call по адресу 0007:00000000, а в Linux это осуществляется через служебное прерывание INT 80h (перечень номеров системных команд можно найти в файле /usr/include/asm/unistd.h). Таким образом, использование nativeAPI существенно ограничивает ареал обитания вируса, делая его непереносимым.

«Честные» программы в большинстве своем практически никогда не работают через nativeAPI (хотя утилиты из комплекта поставки Free BSD 4.5 ведут себя именно так), поэтому наличие большого количества машинных команд INT 80h/CALL 0007:0000000 (CD 80/9A 00 00 00 00 07 00) с высокой степенью вероятности свидетельствует о наличии вируса. Для предотвращения ложных срабатываний (т.е. обнаружения вируса там, где и следов его нет), вы должны не только обнаружить обращения к nativeAPI, но и проанализировать последовательность их вызовов. Для вирусов характерна следующая цепочка системных команд: sys_open, sys_lseek, old_mmap/sys_munmap, sys_write, sys_close, sys_exit. Реже используются вызовы exec и fork. Их, в частности, использует вирус STAOG.4744. Вирусы VirTool.Linux.Mmap.443, VirTool.Linux.Elfwrsec.a, PolyEngine.Linux.LIME.poly, Linux.Winter.343 и ряд других обходятся без этого.

Ниже приведен фрагмент файла, зараженного вирусом VirTool.Linux.Mmap.443. Наличие незамаскированных вызовов INT 80h с легкостью разоблачает агрессивную природу программного кода, указывая на склонность последнего к саморазмножению:

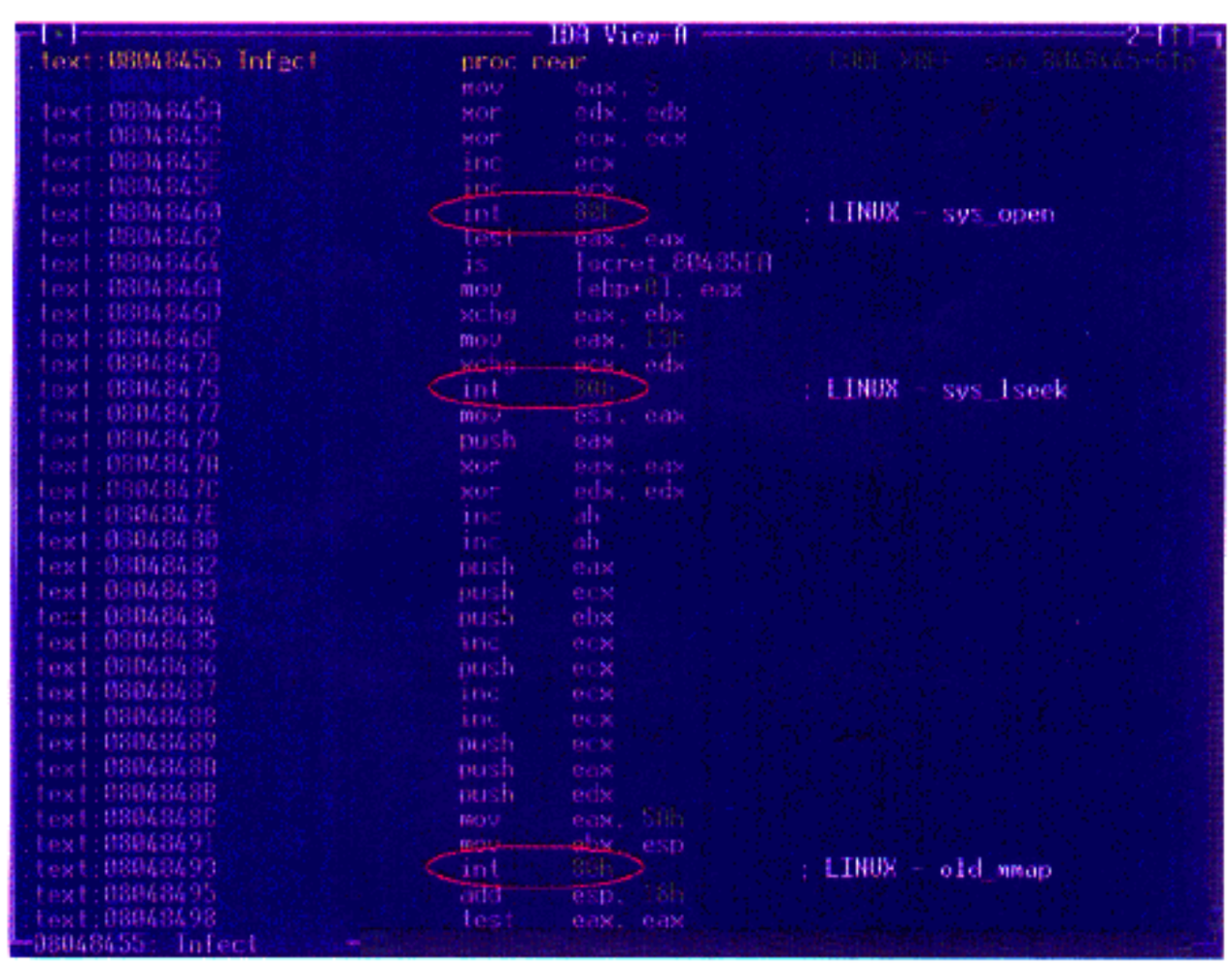

Рисунок 10. Фрагмент файла, зараженного вирусом VirTool.Linux.Mmap.443, демаскирующим свое присутствие прямым обращением к native API операционной системы

А вот так для сравнения выглядят системные вызовы «честной» программы – утилиты cat из комплекта поставки Free BSD 4.5 (см. рис. 11). Инструкции прерывания не разбросаны по всему коду, а сгруппированы в собственных функциях-обертках. Конечно, вирус тоже может «обмазать» системные вызовы слоем переходного кода, но вряд ли у него получится подделать характер оберток конкретного заражаемого файла.

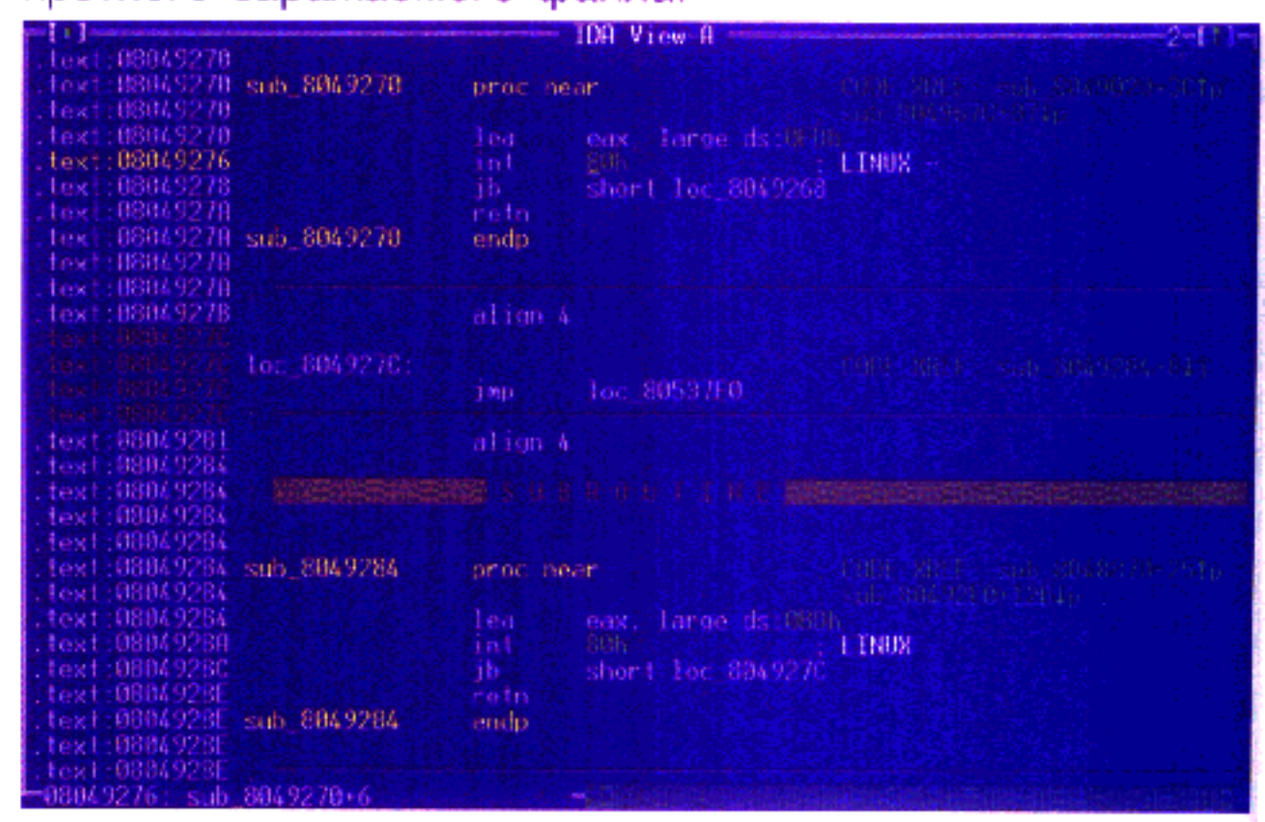

Рисунок 11. Фрагмент «честного» файла cat из комплекта поставки Free BSD, аккуратно размещающего native-API вызовы в функциях-обертках

Некоторые (впрочем, довольно немногочисленные) вирусы так просто не сдаются и используют различные методики, затрудняющие их анализ и обнаружение. Наиболее талантливые (или скорее прилежные) разработчики динамически генерируют инструкцию INT 80h/CALL 0007:00000000 на лету и, забрасывая ее на верхушку стека, скрытно передают ей управление. Как следствие – в дизассемблерном листинге исследуемой программы вызов INT 80h/INT 80h/CALL 0007:0000000 будет отсутствовать, и обнаружить такие вирусы можно лишь по многочисленным косвенным вызовам подпрограмм, находящихся в стеке. Это действительно нелегко, т.к. косвенные вызовы в изобилии присутствуют и в «честных» программах, а определение значений вызываемых адресов представляет собой серьезную проблему (во всяком случае при статическом анализе). С другой стороны, таких вирусов пока существует немного (да и те – сплошь лабораторные), так что никаких поводов для паники пока нет. А вот шифрование критических к раскрытию участков вирусного тела встречается гораздо чаще. Однако для дизассемблера IDA PRO это не бог весть какая сложная проблема, и даже многоуровневая шифровка снимается без малейшего умственного и физического напряжения.

Впрочем, на каждую старуху есть проруха, и IDA Pro тому не исключение. При нормальном развитии событий IDA Pro автоматически определяет имена вызываемых функций, оформляя их как комментарии. Благодаря этому замечательному обстоятельству, для анализа исследуемого алгоритма нет нужды постоянно лезть в справочник. Такие вирусы, как, например, Linux.ZipWorm, не могут смириться с подобным положением дел и активно используют специальные приемы программирования, сбивающие дизассемблер с толку. Тот же Linux.ZipWorm проталкивает номера вызываемых функций через стек, что

вводит IDA в замешательство, лишая ее возможности определения имен последних:

Листинг 14. Фрагмент вируса Linux.ZipWorm, активно и небезуспешно противостоящего дизассемблеру IDA Pro

```
.text:080483C0          push    13h
.text:080483C2          push    2
.text:080483C4          sub     ecx, ecx
.text:080483C6          pop     edx
; // EAX := 2. Это вызов fork
.text:080483C7          pop     eax
; LINUX - ← IDA не смогла определить имя вызова!
.text:080483C8          int     80h
```

С одной стороны, вирус действительно добился поставленной перед ним цели, и дизассемблерный листинг с отсутствующими автокомментариями с первого приступа не возьмешь. Но давайте попробуем взглянуть на ситуацию под другим углом. Сам факт применения антиотладочных приемов уже свидетельствует если не о заражении, то во всяком случае о ненормальности ситуации. Так что за противодействие анализу исследуемого файла вирусу приходится расплачиваться ослабленной маскировкой (в программистских кулуарах по этому случаю обычно говорят «из зараженного файла вирусные уши торчат»).

Уши будут торчать еще и потому, что большинство вирусов никак не заботится о создании стартового кода или хотя бы плохонькой его имитации. В точке входа «честной» программы всегда (ну или практически всегда) расположена нормальная функция с классическим прологом и эпилогом, автоматически распознаваемая дизассемблером IDA Pro, вот например:

Листинг 15. Пример нормальной стартовой функции с классическим прологом и эпилогом

```
text:080480B8 start     proc near
text:080480B8
text:080480B8           push    ebp
text:080480B9           mov     ebp, esp
text:080480BB           sub     esp, 0Ch
...
text:0804813B           ret
text:0804813B start     endp
```

В некоторых случаях стартовые функции передают бразды правления libc_start_main и заканчиваются по hlt без ret. Это вполне нормальное явление. «Вполне» потому что очень многие вирусы, написанные на ассемблере, получают в «подарок» от линкера такой же стартовый код. Поэтому присутствие стартового кода в исследуемом файле, не дает нам никаких оснований считать его здоровым.

Листинг 16. Альтернативный пример нормальной стартовой функции

```
.text:08048330          public start
.text:08048330  start   proc near
.text:08048330          xor     ebp, ebp
.text:08048332          pop     esi
.text:08048333          mov     ecx, esp
.text:08048335          and     esp, 0FFFFFFF8h
.text:08048338          push    eax
.text:08048339          push    esp
.text:0804833A          push    edx
.text:0804833B          push    offset sub_804859C
.text:08048340          push    offset sub_80482BC
.text:08048345          push    ecx
.text:08048346          push    esi
.text:08048347          push    offset loc_8048430
.text:0804834C          call    ___libc_start_main
.text:08048351          hlt
.text:08048352          nop
.text:08048353          nop
.text:08048353  start   endp
```

Большинство зараженных файлов выглядит иначе. В частности, стартовый код вируса PolyEngine.Linux.LIME.poly выглядит так:

Листинг 17. Стартовый код вируса PolyEngine.Linux.LIME.poly

```
; Alternative name is 'main'
.data:080499C1 LIME_END:
.data:080499C1          mov     eax, 4
.data:080499C6          mov     ebx, 1
; "Generates 50 [LiME] encrypted.."
.data:080499CB          mov     ecx, offset gen_msg
.data:080499D0          mov     edx, 2Dh
; LINUX - sys_write
.data:080499D5          int     80h
.data:080499D7          mov     ecx, 32h
```

Заключение

Несмотря на свой, прямо скажем, далеко не маленький размер, настоящая статья охватила далеко не весь круг изначально намеченных тем. Незатронутыми остались вопросы «прорыва» виртуальной машины интерпретатором, техника выявления Stealth-вирусов и противодействия им, жизненный цикл червей и комплекс мер, направленных на предотвращение возможного вторжения...

Обо всем этом и многом другом мы поговорим в следующий раз, если, конечно, к тому времени эта тема никому не надоест...