вирусы в мире UNIX

крис касперски ака мыщъх no-email

трудно представить себе более простую штуку, чем компьютерный вирус. тетрис и тот посложнее будет, однако, программирование вирусов вызывает у начинающих большие трудности: как внедрить свой код в файл, какие поля необходимо изменять, а какие лучше не трогать, чем отлаживать вирусы и можно ли использовать языки высокого уровня? тесные рамки журнальной статьи не позволяют рассмотреть все аспекты конструирования вирусов целиком, но по крайней мере общее представление о вопросе могут дать.

оперативная обстановка

Первые вирусы, поражающие ELF-файлы (основной формат исполняемых файлов под UNIX), были зарегистрированы в конце девяностых, а теперь их популяция насчитывает свыше полусотни представителей (см. коллекцию вирусов на http://vx.netlux.org). Антивирусная Энциклопедия Евгения Касперского (см. http://www.viruslist.com/viruslist.html?id=3166) сообщает лишь о четырнадцати из них, что наводит на серьезные размышления о качестве AVP и добросовестности его создателей.

По умолчанию UNIX запрещает модификацию исполняемых файлов и успешное распространение вирусов возможно только на уровне root, который либо присваивается зараженному файлу администратором, либо самостоятельно захватывается вирусом через дыры в ядре системы. При правильной политике разграничения доступа и оперативном наложении заплаток, угроза вирусного заражения сводится к минимуму. К тому же времена тотального обмена софтом давно позади. Сейчас уже никто не копирует исполняемые файлы друг у друга, скачивая их напрямую с Интернета. Даже если вирус ухитриться поразить центральный сервер, дальше первого поколения его распространение не пойдет и вторичные заражения будут носить единичный характер.

Файловые вирусы уже неактуальны и отсутствие крупных эпидемий наглядно подтверждает этот факт. Тем не менее, накопленные методики внедрения отнюдь не стали бесполезными – без них жизнь троянов и систем удаленного администрирования была бы весьма недолгой. Захватить управление атакуемым компьютером и заполучить права root'a – все равно, что бросить зернышко на раскаленный асфальт. Хакер должен укорениться в системе, цепляясь за все исполняемые файлы, что встретятся ему на пути, но и тогда он не может быть ни в чем уверен, поскольку существует такое понятие как резервное копирование, позволяющее восстановить пораженную систему, как бы глубоко вирус ни был внедрен.

Считается, что вирусы, внедряющиеся в исходные тексты, более живучи, однако, в действительности это не так. Исходные тексты требуется небольшому числу пользователей, а девелоперы активно используют системы контроля версий, отслеживающих целостность программного кода и позволяющих делать многоуровневый "откат". Было зарегистрировано несколько попыток заражения исходных текстов операционной системы LINUX и сервера Apache, однако, все они с треском провалились.

Тоже самое относится и к вирусам, обитающих в интерпретируемых скриптах, таких, например, как sh, perl, php и все-всех-всех. В UNIX скрипты вездесущи и их модификация по умолчанию разрешена, что создает благоприятные условия для размножения вирусов. Если бы пользователи обменивались скриптами, юниксоидный мир погрузился бы в эпоху ранней MS-DOS, когда новые вирусы выходили едва ли не каждый день, а так – вирусы остаются внутри пораженного компьютера, не в силах вырваться наружу.

Разумеется, вирус может распространяется и через Интернет, но тогда это будет не вирус, а червь. Некоторые исследователи считают червей самостоятельными организмами, некоторые – разновидностью вирусов, но как бы там ни было, черви – тема отдельного разговора.

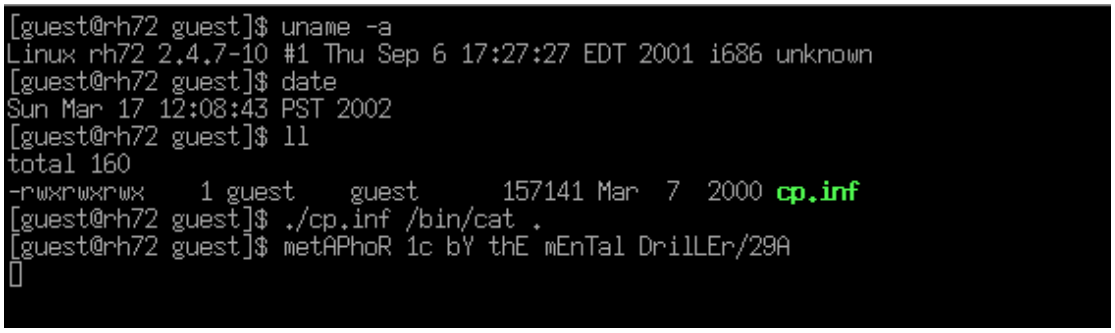

Рисунок 1 вирусных разгул под UNIX

язык разработки

Настоящие хакеры признают только один, ну максимум два языка – Си и Ассемблер, причем последний из них стремительно утрачивает свои позиции, уступая место Бейсику, DELPHI и прочей дряни, на которой элегантный вирус невозможно создать в принципе.

А что на счет Си? С эстетической точки зрения – это чудовищный выбор и вирусописатели старой школы вас за него не простят. С другой стороны, будучи низкоуровневым системно-ориентированным языком, Си неплохо походит для разработки вирусов, хотя от знания ассемблера все равно не освобождает.

К коду, генерируемому компилятором, предъявляются следующие требования: он должен быть полностью перемещаемым (т. е. независимым от базового адреса загрузки), не модифицировать никакие ячейки памяти, за исключением стекового пространства, и не использовать стандартные механизмы импорта функций, либо подключая все необходимые библиотеки самостоятельно, либо обращаясь в native-API. Этим требованиям удовлетворяет подавляющее большинство компиляторов, однако, от программиста тоже кое-что потребуется.

Не объявляйте главную функцию программы как main – встретив такую, линкер внедрит в файл start-up код, который вирусу на хрен не нужен! Не используйте глобальных или статических переменных – компилятор принудительно размещает их в сегменте данных, но у вирусного кода не может быть сегмента данных! Даже если вирус захочет воспользоваться сегментом пораженной программы, он будет должен во-первых, самостоятельно определить адрес его "хвоста", а во-вторых, растянуть сегмент до необходимых размера. Все это тривиально реализуется на ассемблере, но для компилятора оказывается через чур сложной задачей. Храните все данные только в локальных переменных, задавая строковые константы в числовом виде. Если вы напишите `char x[] = "hello, world"`, коварный компилятор сбросит "hello, world" в сегмент данных, а затем динамические скопирует его в локальную переменную x. Делайте так: `x[0]='h', x[1]='e', x[2]='l'`… или преобразуйте char в int осуществляйте присвоение двойными словами, не забывая о том, что младший байт должен располагаться по наименьшему адресу, что разворачивает строку задом наперед.

Не используйте никаких библиотечных функций, если только вы не уверены, что они полностью удовлетворяют всем вышеперечисленным требованиям. Системные функции обычно вызываются через интерфейс native-API, так же известный под именем sys-call (в LINUX-подобных системах за это отвечает прерывание INT 80h, другие системы обычно используют дальний вызов по селектору семь, смещение ноль). Поскольку, системные вызовы варьируются от одной системы к другой, это ограничивает среду обитания вируса и при желании он может прибегнуть к внедрению в таблицу импорта.

Откомпилировав полученный файл, вы получите объективник и ругательство компилятора по поводу отсутствия main. Остается только слинковать его в двоичный 32.64-разрядный файл. Естественно, внедрять его в жертву придется вручную, т. к. системный загрузчик откажется обрабатывать такой файл.

средства анализа, отладки и плагиата

Ну какой вирусописатель может удержаться от соблазна, чтобы не пополнить свой заплечный рюкзак за чужой счет, выдирая идеи и алгоритмы из тел попавших к нему вирусов? Чаще всего вирусами обмениваются тет-а-тет. Коллекции, найденные в сети, для опытных хакеров не представляют никакого интереса, поскольку набираются из открытых источников, но для начинающих исследователей это настоящий клад.

Если исходные тексты вируса отсутствуют (кривые дизассемблерные листинги, выдаваемые за божественное откровение, мы в расчет не берем), препарировать двоичный код вируса приходится самостоятельно. Тут-то нас и поджидает одна большая проблема. Дизассемблер всех времен и народов – **IDA PRO** – не приспособлен для работы с ELF-вирусами, поскольку отказывается загружать файлы с искаженным section header'ом (а большинство вирусов никак не корректируют его после заражения!). Других достойных дизассемблеров, переваривающих ELF-формат, мне обнаружить так и не удалось (а самому писать лень). За неимением лучших идей, приходится трахаться с HEX-редакторами (например, тем же HIEW'ом), разбираясь со служебными структурами файла вручную.

С отладчиками дело обстоит еще хуже. Фактически под UNIX существует всего один более или менее самостоятельный отладчик прикладного уровня – **gdb** (GNU Debugger), являющийся фундаментом для большинства остальных. Простейшие антиотладочные приемы, нарытые в хакерских мануалах времен первой молодости MS-DOS, пускают gdb в разнос или позволяют вирусу вырваться из-под его контроля, поэтому отлаживать вирусный код на рабочей машине категорически недопустимо и лучше использовать для этой цели эмулятор, такой например, как **BOCHS**. Особенно предпочтительны эмуляторы, содержащие интегрированный отладчик, обойти который вирусу будет очень тяжело, а в идеале вообще невозможно (BOCHS такой отладчик содержит). Кстати говоря, совершенно необязательно для исследования ELF-вирусов устанавливать UNIX. Эмулятора для этих целей будет более чем достаточно.

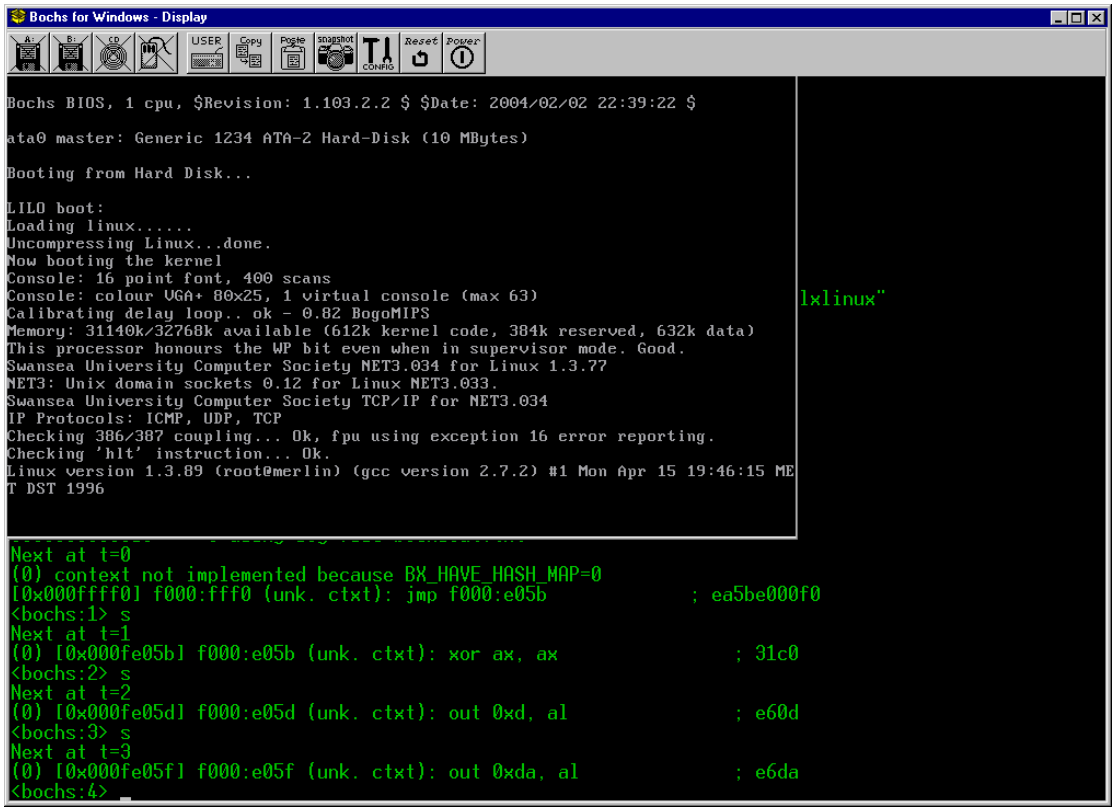

Рисунок 2 отладка вирусного кода на интегрированном отладчике эмулятора BOCHS, запущенного под управлением Windows 2000

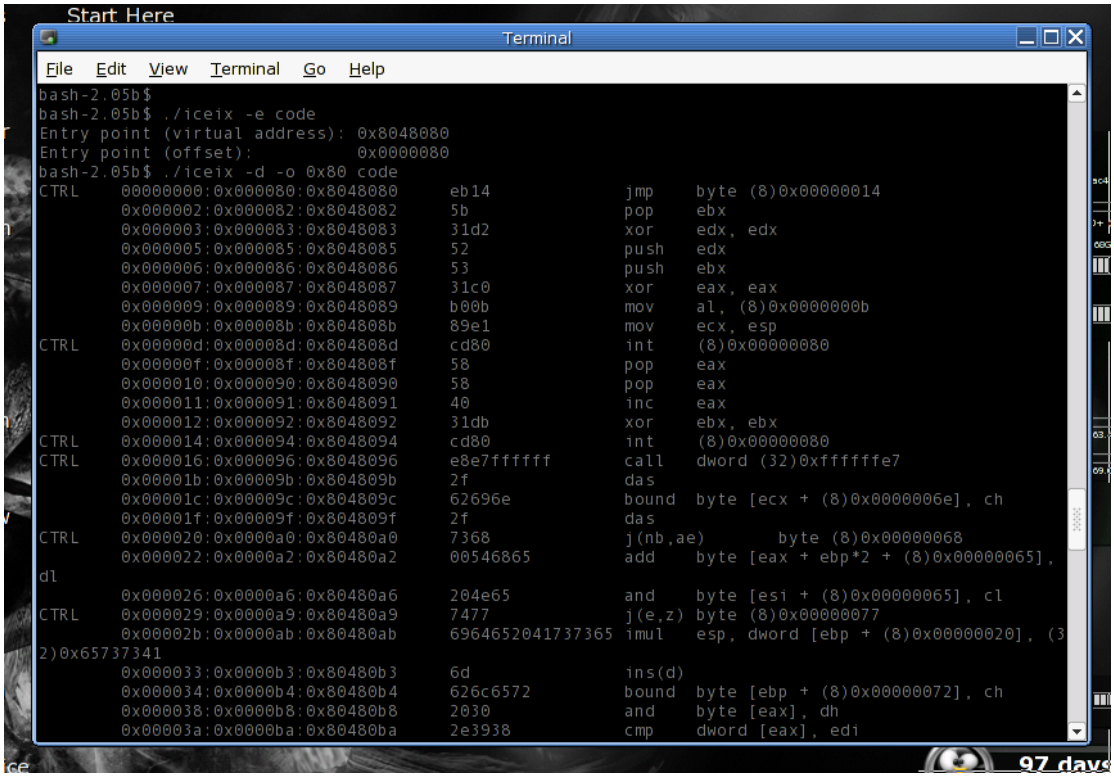

Рисунок 3 исследование вирусов под UNIX с помощью дизассемблера iceix

>>>> врезка кривое зеркало, или как антивирусы стали плохой идеей

Антивирусные программы, в том виде, в котором они есть сейчас, категорически не справляются со своей задачей, да и не могут с ней справиться в принципе. Это не означает, что они полностью бесполезны, но надеяться на их помощь было бы по меньшей мере неразумно. Как уже отмечалось выше, в настоящий момент жизнеспособных UNIX-вирусов практически нет. И, стало быть, антивирусным сканерам сканировать особо и нечего. Эвристические анализаторы так и не вышли из ясельной группы детского сада и к реальной эксплуатации в промышленных масштабах явно не готовы.

Ситуация усугубляется тем, что в скриптовых вирусах крайне трудно выделить устойчивую сигнатуру – такую, чтобы не встречалась в "четных" программах и выдерживала хотя бы простейшие мутации, отнюдь не претендующие на полиморфизм. Антивирус Касперского ловит многие из существующих скриптовых вирусов, но… как-то странно он их ловит. Во-первых, вирусы обнаруживаются не во всех файлах, а во-вторых, простейшее переформатирование зараженного файла приводит к тому, что вирус остается незамеченным.

Все скрипты, позаимствованные из потенциально ненадежных источников, следует проверять вручную, поскольку "*...самый дурацкий троян может за несколько* ***секунд*** *парализовать жизнь сотен контор, которые напрасно надеются на разные антивирусы*" (с) Игорь Николаев. Вы либо безоговорочно доверяете своему поставщику, либо нет. В полученном вами файле может быть все, что угодно (и просто некорректно работающая программа в том числе!).

С двоичными файлами ситуация обстоит более плачевно. Отчасти потому, что их ручной анализ требует глубоких знаний системы и нереальных затрат времени. Отчасти потому, что автоматизированному анализу нормальные вирусы не поддаются в принципе. Поэтому, лучшим средством борьбы по прежнему остается правильная политика разграничения доступа, своевременная установка свежих заплаток и резервное копирование.

структура ELF-файлов

Структура **ELF**-файлов (сокращение от ***E****xecution & **L**inkable **F**ormat*) имеет много общих черт с **PE** (***P****ortable **E**xecution*) – основным исполняемым форматом платформы

Window 9x и NT, а потоку концепции их заражения весьма схожи, хотя и реализуются различным образом.

С высоты птичьего полета ELF-файл состоит из *ELF-заголовка* (*ELF-header*), описывающего основные особенности поведения файла, *заголовка программной таблицы* (*program header table*) и одного или нескольких *сегментов* (*segment*), содержащих код, инициализированные/неинициализированные данные и прочие структуры. (см. рис. 4). Каждый сегмент представляет собой непрерывную область памяти со своими атрибутами доступа (кодовый сегмент обычно доступен только на исполнение, сегменты данных как минимум доступны на чтение, а при необходимости еще и на запись). Пусть слово "сегмент" не вводит вас в заблуждение, ничего общего с сегментной моделью памяти тут нет. Большинство 32-битных реализаций UNIX'a помещают все сегменты ELF-файла в один 4х-гигабайтный "процессорный" сегмент. В памяти все ELF-сегменты должны выравниваться по величине страницы (на x86 равной 4 Кб), но непосредственно в самом ELF-файле хранятся в невыровненном виде, вплотную прижимаясь друг к другу. Сам ELF-заголовок и program header в первый сегмент не входят (ну, формально не входят), но совместно грузятся в память, при этом начало сегмента следует непосредственно за концом program header'a и по границе страницы не выравнивается!

Последим из всех идет *заголовок таблицы секций* (*section header table*). Для исполняемых файлов он необязателен и реально используется только в объективниках. Еще в нем нуждаются отладчики – исполняемый файл с изуродованным section header table не отлаживается ни gdb, ни производными от него отладчиками, хотя нормально обрабатывается операционной системой.

Сегменты естественным образом делятся на *секции*. Типичный кодовый сегмент состоит из секций .init (процедуры инициализации), .plt (секция связок), .text (основой код программы) и .finit (процедуры финализации), атрибуты которых описываются в section header'e. Загрузчик операционной системы ничего не знает о секциях, игнорируя их атрибуты и загружая весь сегмент целиком. Тем не менее, для сохранения работоспособности зараженного файла под отладчиком, вирус должен корректировать оба заголовка сразу как program header, так и section header.

```
ELF Header
        Program header table
        Segment 1
        Segment 2
        Section header table (optional)
```

Рисунок 4 структура исполняемого ELF-файла

Основные структуры ELF-файла задается в файле /usr/include/elf.h и выглядят следующим образом:

```
typedef struct
{
        unsigned char  e_ident[EI_NIDENT];     /* идентифкатор ELF-файла: 7F 45 4C  */
        Elf32_Half     e_type;         /* тип файла                                 */
        Elf32_Half     e_machine;      /* архитектура                               */
        Elf32_Word     e_version;      /* версия объективного файла                 */
        Elf32_Addr     e_entry;        /* виртуальный адрес точки входа             */
        Elf32_Off      e_phoff;        /* физическое смещение program header в файле */
        Elf32_Off      e_shoff;        /* физическое смещение section header в файле */
        Elf32_Word     e_flags;        /* флаги                                     */
        Elf32_Half     e_ehsize;       /* размер ELF-заголовка в байтах             */
        Elf32_Half     e_phentsize;    /* размер элемента program header'a в байтах */
        Elf32_Half     e_phnum;        /* кол-во элементов в program header'e       */
        Elf32_Half     e_shentsize;    /* размер элемента section header'a в байтах */
        Elf32_Half     e_shnum;        /* кол-во элементов в section header'e       */
        Elf32_Half     e_shstrndx;     /* индекс string table в section header'e    */
} Elf32_Ehdr;
```

Листинг 1 структура ELF-заголовка

```
typedef struct
{
        Elf32_Word     p_type;         /* тип сегмента                              */
        Elf32_Off      p_offset;       /* физическое смещение сегмента в файле      */
        Elf32_Addr     p_vaddr;        /* виртуальный адрес начала сегмента         */
```

```
        Elf32_Addr      p_paddr;        /* физические адрес сегмента                */
        Elf32_Word      p_filesz;       /* физические размер сегмента в файле       */
        Elf32_Word      p_memsz;        /* размер сегмента в памяти                 */
        Elf32_Word      p_flags;        /* флаги                                    */
        Elf32_Word      p_align;        /* кратность выравнивания                   */
} Elf32_Phdr;
```

Листинг 2 структура program segment header'a

```
typedef struct
{
        Elf32_Word      sh_name;        /* имя секции (tbl-index)                   */
        Elf32_Word      sh_type;        /* тип секции                               */
        Elf32_Word      sh_flags;       /* флаги секции                             */
        Elf32_Addr      sh_addr;        /* виртуальный адрес начала секции          */
        Elf32_Off       sh_offset;      /* физическое смещение секции в файле       */
        Elf32_Word      sh_size;        /* размер секции в байтах                   */
        Elf32_Word      sh_link;        /* связка с другой секций                   */
        Elf32_Word      sh_info;        /* дополнительная информация о секции       */
        Elf32_Word      sh_addralign;   /* кратность выравнивая секции              */
        Elf32_Word      sh_entsize;     /* размер вложенного элемента если есть     */
} Elf32_Shdr;
```

Листинг 3 структура Section header'a

За более подробной информацией обращайтесь к оригинальной спецификации на ELF-файла "Executable and Linkable Format – Portable Format Specification", составленной естественно на английском языке.

методы заражения

Простейший и наиболее универсальный метод заражения сводится к поглощению оригинального файла вирусом. Вирус просто дописывает оригинальный файл к своему телу как оверлей, а для передачи управления жертве проделывает обратный процесс: пропускает первые virus_size байт своего тела (что обычно осуществляется функцией seek), считывает оставшийся хвост и записывает его во временный файл. Присваивает атрибут исполняемого и делает ему exec, предварительно расщепив материнский процесс функцией fork. После завершения работы файла-жертвы, вирус удаляет временный файл с диска.

Описанный алгоритм элементарно реализуется на любом языке программирования вплоть до Бейсика, и пригоден как для исполняемых файлов, так и для скриптов. Однако, ему присущие следующие недостатки: он медлителен и неэлегантен, требует возможности записи на диск и прав установки атрибута "исполняемый", кроме того появление посторонних файлов на диске не может долго оставаться незамеченным и участь вируса заранее предрешена. Поэтому большинство вирусов не используют такую методику, а предпочитают внедряться в конец последнего сегмента файла, расширяя его на необходимую величину.

Под "последним" здесь подразумевается последний подходящий сегмент файла, которым как правило является сегмент инициализированных данных, за которым следует сегмент неинициализированных данных, занимающий ноль байт дисковой памяти. Конечно, можно внедриться и в него, но это будет выглядеть как-то странно.

Приблизительный алгоритм внедрения в конец ELF-файла выглядит следующим образом:

- вирус открывает файл и, считывая его заголовок, убеждается, что это действительно ELF;
- просматривая *Program Header Table*, вирус отыскивает последний сегмент с атрибутом PL_LOAD;
- 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).

Теоретически, вирус может внедриться в середину файла, дописав свое тело в конец кодового сегмента и сдвинув все последующие сегменты вниз, однако, при этом ему потребуется скорректировать все указатели на ячейки сегмента данных, поскольку после заражения они будут располагаться по совершенно другим адресам. Как вариант, перед передачей управления программе-носителю, вирус может "подтянуть" опущенные сегменты вверх, вернув их на свое законное место, однако, если файл содержит перемещаемые элементы или прочие служебные структуры данных вирусу их придется скорректировать тоже, в противном случае, системный загрузчик необратимо исказит зараженный файл и тот откажет в работе. Все это слишком сложно для начинающих, а потому вирусы подобного типа не получили большого распространения.

Теоретически возможно внедриться в область, образованную выравниванием сегментов в памяти. Поскольку, границы сегментов всегда выравниваются на величину 4 Кб, между концом кодового сегмента и началом сегмента данных обычно можно наскрести некоторое количество незанятого пространства, однако, никаких гарантий на этот счет у нас нет и потому для заражения подходят далеко не все файлы.

- вирус открывает файл и, считывая его заголовок, убеждается, что это действительно ELF;
- просматривая program header table, вирус находит сегмент с атрибутом PL_LOAD и (PAGE_SIZE % p_filesz) >= virus_size; если же такого сегмента нет, вирус отказывается от заражения;
- поля p_filez (размер на диске) и p_memsz (размер в памяти) соответствующего сегмента увеличиваются на длину тела вируса;
- поле p_offset и факультативно sh_offset всех последующих сегментов/секций увеличивается на длину тела вируса;
- поля e_phoff и факультативно e_shoff ELF-заголовка увеличивается на величину тела вируса;
- вирус внедряем себя в конец выбранного сегмента;
- для перехвата управления вирус корректирует точку входа в файл (e_entry), либо же внедряет в истинную точку входа jmp на свое тело;

Некоторые вирусы внедряются в область памяти между заголовком и началом первого сегмента (ну или во всяком случае пытаются это сделать). Однако, большинство файлов "приклеивают" свой первый сегмент к заголовку и потому для внедрения просто не остается свободного места.

Смотрите:

```
 Name      Start    End      Align Base Type Class 32 es   ss   ds   fs   gs
.init     080480AC 080480B7 dword 0001 publ CODE  Y  FFFF FFFF 0005 FFFF FFFF
.text     080480B8 08053ABB dword 0002 publ CODE  Y  FFFF FFFF 0005 FFFF FFFF
.fini     08053ABC 08053AC2 dword 0003 publ CODE  Y  FFFF FFFF 0005 FFFF FFFF
.rodata   08053AE0 08055460 32byt 0004 publ CONST Y  FFFF FFFF 0005 FFFF FFFF
.data     08056460 08057530 32byt 0005 publ DATA  Y  FFFF FFFF 0005 FFFF FFFF
```

Рисунок 5 структура файла echo из комплекта поставки FreeBSD 4.5, между секциями .finit и .rodata расположено всего лишь 1Eh байт данных, что недостаточно для размещения даже крошечного вируса

```
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
```

Рисунок 6 структура файла ls из комплекта поставки RedHat 5.0, между секциями .rodata и .data имеется расположено 1000h байт, что с лихвой достаточно для размещения даже высокотехнологичного вируса

общая структура и стратегия вируса

Конкретная структура вирусного кода зависит от фантазии его разработчика и выглядит приблизительно так же, как и в Windows-вирусах. Обычно вначале вируса находится расшифровщик, за ним расположен модуль поиска подходящих жертв, инжектор вирусного кода и процедура передачи управления файлу-носителю.

Для большинства ELF-вирусов характерная следующая последовательность системных вызовов: sys_open (mov eax, 05h/int 80h)[1] открывает файл; sys_lseek (mov eax,13h) перемещает файловый указатель на нужное место; old_mmap (mov eax, 5Ah/int 80h) проецирует файл в память; sys_unmap (mov eax, 5Bh/int 80h) удаляет образ из памяти, записывая на диск все изменения, а sys_close (mov eax, 06/int 80h) закрывает сам файл (см. рис. 1)

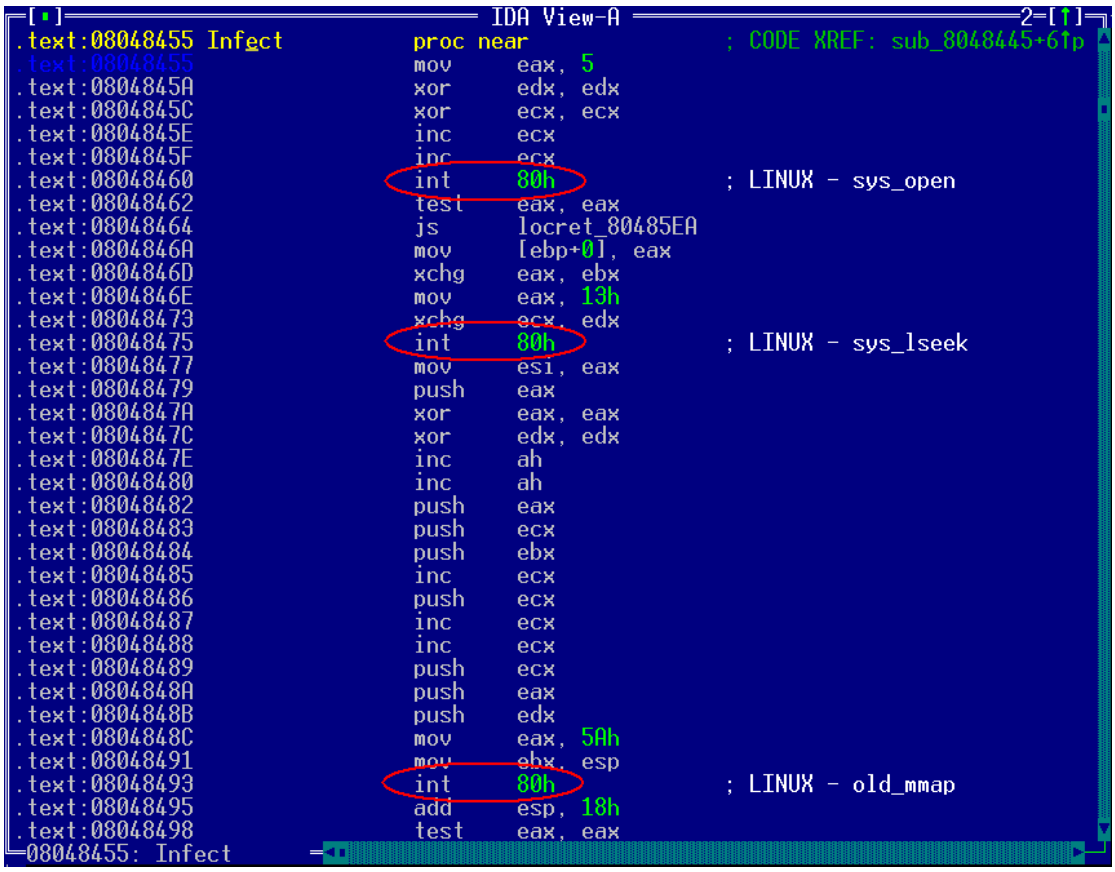

Рисунок 7 типичная структура вирусного кода

Техника проецирования (или, выражаясь забугорной технологией mapping), значительно упрощает работу с файлами большого объема. Теперь уже не нужно выделять буфер, копируя туда файл по кускам и всю черную работу можно переложить на плечи операционной системы, сосредоточив свои усилия непосредственно на процессе заражения. Правда, при заражении файла протяженностью в несколько гигабайт (например, самораспаковывающегося дистриьютива какого-то программного продукта), вирусу придется либо просматривать файл через "окно", проецируя в 4х гигабайтное адресное пространство различные его части, либо попросту отказаться от заражения, выбрав файл поприличнее. Подавляющее большинство вирусов именно так и поступают.

>>>>врезка перехват управления путем модификации таблицы импорта

Классический механизм импорта внешних функций из/в ELF-файлов в общем виде выглядит так: на первом этапе вызова импортируемой функции из секции .text вызывается "переходник", расположенный в секции .plt (*Procedure Linkable Table*) и ссылающийся в свою

[1] приведенные номера системных функций относятся к LINUX.

очередь на указатель на функцию printf, расположенный в секции .got ("Global Offset Tables"), ассоциированной с таблицей строк, содержащей имена вызываемых функций (или их хеши).

Ниже приведена схема вызова функции printf утилитой ls, позаимствованной из комплекта поставки Red Hat 5.0:

```
.text:08000E2D         call    _printf
…
.plt:08000A58 _printf  proc near
.plt:08000A58
.plt:08000A58          jmp     ds:off_800628C
.plt:08000A58 _printf  endp
…
.got:0800628C off_800628C     dd offset printf
…
extern:8006580 extrn printf:near ; weak
…
0000065B:  FF 00 6C 69-62 63 2E 73-6F 2E 35 00-73 74 70 63  y libc.so.5 stpc
0000066B:  70 79 00 73-74 72 63 70-79 00 69 6F-63 74 6C 00  py strcpy ioctl
0000067B:  70 72 69 6E-74 66 00 73-74 72 65 72-72 6F 72 00  printf strerror
```

Листинг 4 схема вызова функции printf утилитой ls

В какое место этой цепочки может внедриться вирус? Ну, прежде всего он может создать подложную таблицу строк, перехватывая вызовы всех интересующих его функций. Чаще всего заражению подвергается функция printf/fprintf/sprintf (поскольку, без этой функции не обходится практически ни одна программа) и функции файлового ввода/вывода, что автоматически обеспечивает прозрачный механизм поиска новых жертв для заражения.

Вирусы-спутники, создав специальную библиотеку-перехватчик, во всех заражаемых файлах. Поскольку IDA Pro при дизассемблировании ELF-файлов не отображает имя импортируемой библиотеки, заподозрить что-то неладное в этой ситуации нелегко. К счастью, HEX-редакторы еще никто не отменял. Другие же вирусы склонны манипулировать полями глобальной таблицы смещений, переустанавливая их на свое тело.

>>>> врезка ссылки по теме

bochs

Качественный эмулятор ПК с интегрированным отладчиком внутри. Хорошо подходит для экспериментов в вирусами непосредственно на вашей рабочей машине без риска уничтожения информации. Бесплатен, распространяется с исходными текстами. http://bochs.sourceforge.net

Executable and Linkable Format – Portable Format Specification

"Родная" спецификация на ELF-формат. Настоятельно рекомендуется к изучению всем вирусописателям, пробующим свои силы на платформе UNIX. www.ibiblio.org/pub/historic-linux/ftp-archives/sunsite.unc.edu/Nov-06-1994/GCC/ELF.doc.tar.gz.

The Linux Virus Writing And Detection HOWTO

Пошаговое руководство по проектированию и реализации вирусов под LINUX с кучей готовых примеров (на английском языке). http://www.creangel.com/papers/writingvirusinlinux.pdf

"Unix viruses" от Silvio Cesare

Статья, описывающая основные принципы функционирования UNIX-вирусов и способы их детектирования (на английском языке). http://vx.netlux.org/lib/vsc02.html

LINUX VIRUSES - ELF FILE FORMAT Marius Van Oers

Блестящий обзор современных UNIX-вирусов и анализ используемых ими методик внедрения в ELF-файлы (на английском языке).
www.nai.com/common/media/vil/pdf/mvanvoers_VB_conf%25202000.pdf&e=747

заключение

В ближайшее время по-видимому следует ожидать лавинообразного роста численности ELF-вирусов, ибо для этого имеются все условия. Всплеск интереса к LINUX пошел не пользу этой операционной системы. В погоне за улучшениями превратили ее в решето, дырявое как дуршлаг, прикрутили "интуитивно-понятный" графический интерфейс, но не предупредили пользователей, что прежде чем начать работать с системой, следует перелопатить тысячи страниц технической документации и прочитать хотя бы пару умных книжек, в противном случае зараза не заставит себя долго ждать.

Чем больше народу перейдет на UNIX, тем больше среди них окажется хакеров и вирусописателей и тогда с UNIX произойдет тоже, что в свое время произошло с MS-DOS. Будут ли эти вирусы добродушными или злобными – это зависит от вас.

>>>> выноски

- некоторые администраторы полагают, что под UNIX вирусов нет. вирусы же придерживаются иного мнения;
- некоторые пользователи, в желании почувствовать себя богом, подолгу работают в системе на уровне root. вирусы любят таких пользователей;
- малочисленность вирусов в мире UNIX, компенсируется отсутствуем нормальных антивирусов;
- Осел и IRC – вот основные источники для пополнения вашей коллекции вирусов;
- открытость ELF-формата вкупе с доступностью исходных текстов системного загрузчика значительно упрощает конструирование вирусов под UNIX;
- создание вирусов не преследуется по закону. по закону преследуется создание вредоносных программ;
- из десятка возможных методов внедрения в ELF-файлы, вирусописателям удалось освоить лишь два-три, так что на отсутствие творческого простора жаловаться не приходится;
- UNIX- и Windows-вирусы строятся по одним и тем же принципам, причем UNIX-вирусы даже проще;
- Антивирусная Энциклопедия Касперского содержит большое кол-во фактических ошибок в описании UNIX-вирусов;
- многие UNIX-вирусы зависят от версии операционной системы, поэтому всякий исследователь вынужден держать на своей машине зоопарк осей;
- огромная коллекция UNIX-вирусов на http://vx.netlux.org.