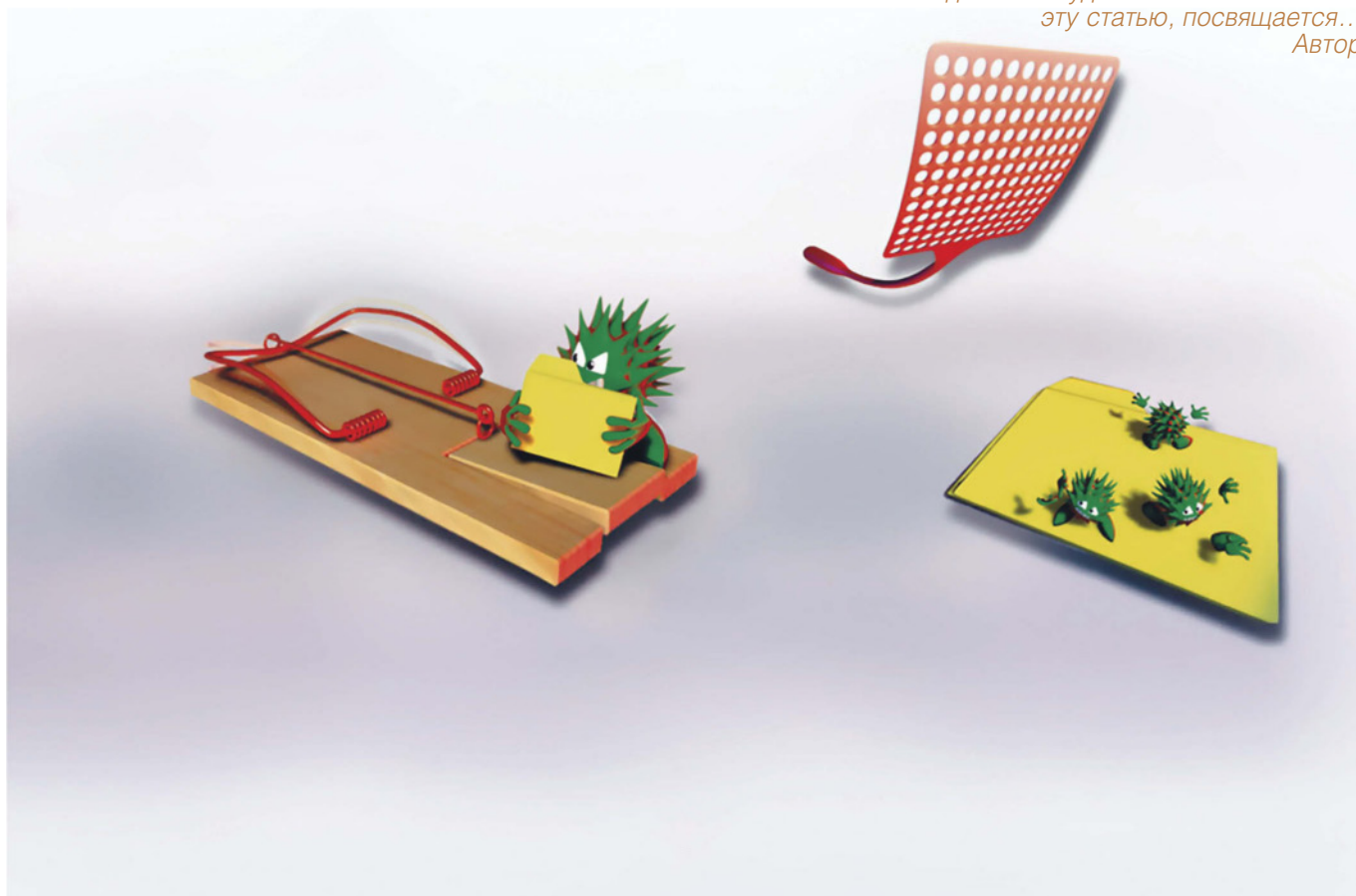


БОРЬБА С ВИРУСАМИ

ОПЫТ КОНТРТЕРРОРИСТИЧЕСКИХ ОПЕРАЦИЙ

Не рой яму другому, чтобы он не использовал
ее как окоп!
Солдатская мудрость

Червя LoveSan, разрушительной силой
своей эпидемии побудившего меня написать
эту статью, посвящается...
Автор



...знайте: когда вы читаете эти строки, какой-то парень на планете отлаживает очередной вирус, который не сегодня завтра нанесет удар и одной из жертв вирусного террора окажется вы. Не пытайтесь отмахнуться от проблемы и не надейтесь, что на этот раз вас «пронесет»! Вирусные атаки стали слишком интенсивными и никто не может чувствовать себя в безопасности. Использование антивирусов ничего не меняет: если вы администрируете локальную сеть крупной организации, персонально для вас может быть написан специальный вирус (троянская программа, шпион), проходящий сквозь антивирусные заслоны, как нож сквозь масло. Причем если до недавнего времени вирусы были нетехнической проблемой «грязных рук», которая решалась элементарным выламыванием дисководов и раздачей по ушам всем любителям левого «софта», то основная масса современных вирусов проникает в целевые компьютеры самостоятельно, не требует никаких действий со стороны пользователя. Всякий уважающий себя администратор должен уметь находить и нейтрализовать имеющие его вирусы самостоятельно. Тому, как именно это сделать, и посвящена настоящая статья. (P.S. речь идёт о ОС Windows NT/W2K/XP/9x/Me)

КРИС КАСПЕРСКИ

Расплата за бездумность

Ущерб, приносимый вирусами, троянскими конями и прочими зловредными программами, трудно переоценить. И дело здесь не только в разрушенной информации (при своевременном резервировании данных их всегда можно восстановить). Гораздо большие убытки приносит панический страх перед самой возможностью заражения, выливающийся в настоящую вирусную истерию.

А всякий страх зиждется на незнании. После изобретения громоотвода молнии по-прежнему продолжают убивать людей, однако сейчас их (молний) уже не так боятся и, оказавшись застигнутым молнией один на один, всякий грамотный человек знает, как свести риск поражения к минимуму. Напротив, поддавшись панике и действуя наобум, вы идете прямой тропой к своему кладбищу. И плачевные результаты попыток противостояния вирусным атакам – лучшее тому подтверждение. Лихорадочные переустановки операционной системы, чередующиеся с форматированием винчестера и отрубанием себя от Сети – ничуть не эффективнее омовения сервера святой водой или накачиванием его антибиотиками.

Использование антивирусов также не решает проблемы. Чтобы там ни говорила реклама, а качество антивирусных программ все еще оставляет желать лучшего. Зачастую вирусы не распознаются совсем или распознаются, но не удаляются. Мягкая переустановка системы (т.е. переустановка «поверх» ранее установленной версии) не гарантирует удаления заразы, и многие зловредные программы ее вполне благополучно переживают! Форматирование диска – вообще безумный способ лечения, сродни сжиганию больных на костре – жестокий и крайне неэффективный. До тех пор, пока не будут перекрыты все каналы проникновения вируса в систему, повторные заражения будут происходить вновь и вновь!

Основной недостаток подавляющего большинства антивирусов как раз и состоит в том, что удаляя вирус из системы, они даже и не пытаются заткнуть те дыры, которые вирус использует для своего распространения. Как следствие, «лечение» компьютера, подключенного к сети, превращается в перегон тараканов из одной казармы в другую, а потом обратно. Ладно, заражение локальной сети – это еще полбеды («останавливаем» сеть, лечим все машины, «запускаем» сеть), но вот проникновение вирусов в Интернет представляет собой весьма нетривиальную проблему. Вылечить все машины глобальной сети за раз просто нереально... Можно (и нужно!) установить очередное обновление от Microsoft, заткнув брешь в системе безопасности, но... кто даст голову на отсечение, что этот способ действительно сработает? Ряд обнаруженных дыр парням из Microsoft удалось заткнуть лишь со второго-третьего раза, а некоторые дыры остались не заткнутыми и до сих пор (или заплатки были выпущены не для всех ОС). Причем наблюдается ярко выраженная тенденция в ухудшении поддержки четвертой версии Windows NT. Хоть и древней, но до сих пор работающей.

В идеале каждый администратор должен быть готов к самостоятельному отражению вирусной атаки, не надеясь на помощь извне. Существование подобных отрядов са-

мообороны, рассредоточенных по всей Сети, сделало бы развитие глобальных эпидемий практически невозможным и снизило убытки от хакерских атак к разумному минимуму. В свое время существовала прекрасная книга «Компьютерные вирусы в MS-DOS» Евгения Касперского, доходчиво объясняющая методики рукопашной борьбы с вирусами, доступные для освоения всякому специалисту средней руки. Однако с появлением Windows и развитием глобальных сетей, стратегия заражения файлов существенно изменилась, и старые рецепты перестали работать, а новых книг по этой тематике с тех пор так и не выходило.

Данная статья рассказывает о наиболее типичных способах инфицирования исполняемых файлов и методах их выявления. Материал ориентирован на системных администраторов и прикладных программистов с минимальным уровнем подготовки.

Что нам потребуется?

Анализ вирусного кода требует обширных значений из различных областей программирования, а также специализированного инструментария, без которого исследовательская работа рискует превратиться в орудие средневековой пытки. Все это отпугивает новичков, которые порой даже и не пытаются взять в руки дизассемблерный меч, считая, что борьба с вирусами слишком сложна для них. Однако это предположение неверно. Бесспорно, наивно надеяться на то, что искусству дизассемблирования можно научиться за одну ночь, но вот пары недель упорного труда для достижения поставленной цели должно оказаться достаточно.

Знание языка ассемблера – древнейшего языка программирования – обязательно. И одних лишь учебников в стиле «ASSEMBLER» Юрова и «Программируем на языке ассемблера IBM PC» Рудакова для его освоения катастрофически недостаточно, поскольку всякий язык познается лишь при общении «вживую». Сходите на любой системно-ориентированный сайт (например, www.wasm.ru) и попытайтесь ухватить суть ассемблера извне, а не изнутри. На форумах, где дикие люди произносят непонятные слова, ругаются матом и обсуждают репродуктивные рецепторы вирусов, витает особый системный дух, делающий все сложное таким простым и понятным.

В конечном счете ассемблер – это всего лишь язык, причем очень и очень простой. Некоторые даже сравнивают его с эсперанто – десяток команд и вы уже можете сносно говорить. Единственная сложность состоит в том, что вирусы, в отличие от нормальных программ, содержат множество ассемблерных «извращений», смысл которых понятен только посвященным. Для непосвященных же это интеллектуальный вызов! Это увлекательные логические (и психологические!) головоломки; это бессонные ночи, горы распечаток, яркие озарения и ни с чем не сравнимые радости найденных вами решений! Хотя, если говорить честно... все уже украдено до нас, тьфу, все головоломки давным-давно разгаданы, а задачки решены. Ресурсы глобальной Сети к вашим услугам! Посетите сайт удивительного человека и исследователя программ Марка Русиновича – <http://www.sysinternals.com>, а также отыщите его книгу «Внутреннее устройство Windows 2000». Еще вам пригодится знаменитый Interrupt List Ральфа Брауна, – хорошо

структурированный справочник по портам, ячейкам памяти и прерываниям (включая недокументированные). Последние версии Platform SDK и DDK от Microsoft и Basic Architecture/Instruction Set Reference/System Programming Guide от Intel иметь обязательно. Русские переводы технической документации, которые заполнили книжные магазины, годятся разве что для студентов, работающих над очередным рефератом, который все равно не читают, а для реальной работы они не пригодны.

Из инструментария вам в первую очередь понадобится хороший отладчик и дизассемблер. Конечно, свой выбор каждый волен делать сам, но ничего лучше soft-ice от NuMega (www.numega.com) и IDA PRO от Ильфака Гуильфанова (www.idapro.com) еще никто не видел. Оба этих продукта относятся к классу тяжелой артиллерии и по сложности своего управления ничуть не уступают таким софтверным монстрам, как, например, Photoshop или Corel DRAW. Равно как изучение интерфейса, Photoshop не заменяет собой освоение техники рисования, так и искусство владения отладчиком/дизассемблером не ограничивается чтением штатной документации. Ищите в магазинах «Отладка Windows-приложений» Джона Роббинса, «Отладчик soft-ice» Романа Айрапетяна, «Образ мышления – дизассемблер IDA» Криса Касперски и «Фундаментальные основы хакерства – искусство дизассемблирования» его же.

Места наиболее вероятного внедрения вирусов

Объектом вирусного поражения могут выступать как исполняемые файлы (динамические библиотеки, компоненты ActiveX, плагины), драйвера, командные файлы операционной системы (bat, cmd), загрузочные сектора (MBR и BOOT), оперативная память, файлы сценариев (Visual

Источники угрозы

По данным сайта VX.NETLUX.ORG на начало сентября 2003 года рейтинг «популярности» вирусов и троянских коней выглядел так:

- I-Worm.Sobig.f
- Worm.Win32.Lovesan
- Worm.Win32.Welchia
- I-Worm.Sobig.a
- Worm.Win32.Ladex
- Win32.Parite
- I-Worm.FireBurn
- Trojan.Win32.Filecoder
- I-Worm.Mimail
- I-Worm.Klez.a-h
- 33.525
- Worm.P2P.Harex.a
- I-Worm.Tanatos.a
- TrojanProxy.Win32.Webber
- MBA.First
- AJ.family
- Worm.P2P.Tanked
- Andrey.932
- Worm.Win32.Opasoft
- Worm.Win32.Autorooter

Рисунок 1. TOP-20 вирусной гонки на выживание

Все двадцать вирусов из двадцати «сильнейших» чрезвычайно примитивные и неспособные к качественной мимикрии твари и все они легко обнаруживаются даже при поверхностном анализе исследуемых файлов по методикам, описанным ниже.

Basic Script, Java Script), файлы документов (Microsoft Word, Microsoft Excel) и... это далеко не все! Фантазия создателей вирусов поистине безгранична, и потому угрозы следует ожидать со всех сторон.

Поскольку, охватить все вышеперечисленные типы объектов в рамках одной-единственной статьи не представляется сколь-нибудь разрешимой задачей, автор остановил свой выбор на самых интересных вирусоносителях из всех – на исполняемых файлах. Во-первых, вирусы, поражающие исполняемые файлы (а также троянские программы, распространяющиеся через них же), лидируют по численности среди всех остальных типов вирусов вообще. Во-вторых, методология анализа new-EXE файлов на предмет их заражения не в пример скудно освещена. В-третьих, тема дизассемблирования достаточно интересна и сама по себе. Для многих она служит источником творческого вдохновения да и просто хорошим средством времяпрепровождения.

Так что не будем мешкать и совершим наш решительный марш-бросок, снося всех вирусов, встретившихся на нашем пути!

Основные признаки вирусного внедрения

Единственным гарантированным способом выяснения, является ли данный файл «плохим» файлом или нет, является его полное дизассемблирование. Не скрою, дизассемблирование – крайне кропотливая работа и на глубокую реконструкцию программы размером в пять – десять мегабайт могут уйти годы если не десятки человеко-лет! Чудовищные трудозатраты делают такой способ анализа чрезвычайно непривлекательным и бесперспективным. Давайте лучше отталкиваться от того, что подавляющее большинство вирусов и троянских коней имеют ряд характерных черт, своеобразных «родимых пятен», отличающих их от всякой «нормальной» программы. Надежность таких «индикаторов» зараженности существенно ниже и определенный процент зловредных программ при этом останется незамечен, но... как говорится, на безрыбье и слона из мухи сделаешь!

Количество всевозможных «родимых пятен», прямо или косвенно указывающих на зараженность файла, весьма велико и ниже перечислены лишь наиболее характерные из них. Но даже они позволяют обнаружить до 4/5 всех существующих вирусов, а по некоторым оценкам и более того (по крайней мере все «лауреаты» вирусного TOP-20 обнаруживаются).

Текстовые строки

Прежде чем приступать к тотальному дизассемблированию исследуемого файла, нелишне пролистать его дампы на предмет выявления потенциально небезопасных текстовых строк, к которым, в частности, относятся команды SMTP-сервера и командного интерпретатора операционной системы («HELO/MAIL FROM/MAIL TO/RCPT TO, DEL/COPY/RD/RMDIR соответственно), ветвей автозапуска реестра (RunServices, Run, RunOnce), агрессивные лозунги и высказывания («легализуем марихуану», «сам дурак») и т. д.


```

data:0040E048 aQuit          db 'QUIT',0Dh,0Ah,0
data:0040E050                db ' ',0Dh,0Ah,0
data:0040E058 aData          db 'DATA ',0Dh,0Ah,0
data:0040E060 aHeloS         db 'HELO %s',0Dh,0Ah,0
data:0040E06C asc_40          db '>',0Dh,0Ah,0
data:0040E070 aMailFrom      db 'MAIL FROM: <',0
data:0040E080 aRcptTo        db 'RCPT TO: <',0
data:0040F244 aSoftwareMicro db 'Software\Microsoft\Windows\CurrentVersion',0
data:0040F279 aRun            db 'Run',0
data:0040F27D aRunonce        db 'RunOnce',0
data:0040F285 aSystemCurrent db 'System\CurrentControlSet\Services',0
data:0040F2A7 aSoftwareMicr_ db 'Software\Microsoft\WAB\WAB4\Wab File Name',0
data:0040F2D1 aRunservices    db 'RunServices',0
data:0040F2DD aInternetSettin db 'Internet Settings\Cache\Paths',0
data:0040F302 aHi             db 'Hi',0
data:0040F306 aHello          db 'Hello',0
data:0040F30D aRe             db 'Re',0
data:0040F311 aFw             db 'Fw',0
data:0040F315 aUndeliverableM db 'Undeliverable mail-',0
data:0040F32E aReturnedMailS db 'Returned mail-',0

```

Рисунок 2. Фрагмент вируса I-Worm.Kiliez.e, текстовые строки содержащиеся в теле которого выдают агрессивные намерения последнего с головой

Конечно, все это еще не свидетельство наличия вируса (троянской программы), а отсутствие компрометирующих программу текстовых строк – не гарант ее лояльности, но... просто поразительно, какое количество современных вирусов ловится таким элементарным способом. Не иначе как снижение культуры программирования дает о себе знать? Действительно, подавляющее большинство программ (и зловредных программ в том числе) сегодня разрабатывается на языках высокого уровня, и программисты не дают себе труда хоть как-то скрыть «уши», торчащие из секции данных (не знают, как это сделать?). Откомпилированная программа просто шифруется статическими упаковщиками, которые легко поддаются автоматической/полуавтоматической распаковке, выдавая исследователю исходный дамп со всеми текстовыми строками на поверхности (см. раздел «Идентификация упаковщика и автоматическая распаковка»).

Выше в качестве примера приведен фрагмент вируса I-Worm.Kiliez.e, на малоизвестность которого жаловаться не приходится (вах! как трудно взглянуть на дамп того, что вы запускаете!).

Идентификация упаковщика и автоматическая распаковка

Упаковка исполняемых файлов «навесными» упаковщиками была широко распространена еще во времена гос-

подства MS-DOS и преследовала собой следующие цели:

- уменьшение размеров программы на диске;
- сокрытие текстовых строк от посторонних глаз;
- затруднение анализа программы;
- «ослепление» сигнатурного поиска.

Два последних пункта стоит отметить особо. Напрямую дизассемблировать упакованную программу нельзя. Прежде исследователю предстоит разобраться с упаковщиком, зачастую построенном по весьма не тривиальным алгоритмам, а также содержащим большое количество разнообразных приемов против отладчиков и дизассемблеров. Также существуют и полиморфные упаковщики, генерирующие машинный код распаковщика на «лету» и делающий зашифрованные экземпляры одной и той же программы не похожими друг на друга.

Для борьбы с упаковщиками было создано большое количество автоматических распаковщиков, работающих по принципу трассировки исполняемого кода и отслеживания момента передачи управления на оригинальный код. Для борьбы с антиотладочными приемами использовалась технология эмуляции процессора, обхитрить которую было не так-то просто, хотя и возможно, но на этот случай в некоторых из распаковщиков был предусмотрен режим ручной распаковки, в котором распаковывалось все, что только было можно распаковать.

С переходом на Windows многое изменилось. Количество упаковщиков резко возросло, но ни одного универсального распаковщика до сих пор так и не появилось, а потому анализ упакованных файлов представляет собой одну из актуальнейших проблем современной антивирусной индустрии.

Если при дизассемблировании исследуемого файла большую часть исполняемого кода дизассемблер представил в виде дампа или выдал на выходе бессмысленный мусор (неверные опкоды команд, обращения к портам ввода/вывода, привилегированные команды, несуществующие смещения и т. д.), то файл скорее всего

Критическая ошибка в SVCHOST.EXE

Если, работая под Windows 2000/Windows XP, вы поймаете сообщение о критической ошибке приложения в модуле SVCHOST.EXE и эта критическая ошибка с завидной регулярностью станет повторяться вновь и вновь, не торопитесь переустанавливать систему, не снесите ваш компьютер в ремонт! Источник ошибки сидит отнюдь не в нем, а приходит к вам по Сети своим шагом и имя ему – DCOM RPC bug.

Небрежное тестирование операционной системы вкупе с использованием потенциально опасных языков программирования привело к тому, что всякий нехороший человек получил возможность выполнять на вашей машине свой зловредный машинный код, управление которым передается путем нехитрого переполнения буфера. Однако современные хакеры в своей массе настолько тупы, что даже переполнить буфер, не уронив при этом машину, оказываются не в состоянии!

Немедленно кликните мышом по Windows Update и скачайте все критические обновления, которые вы по своей лени не скачали до сих пор! Поймите же наконец, что антивирусы против этой беды вам все равно не помогут, поскольку осуществляют лечение пост-фактум, когда зачастую лечить уже нечего... На худой конец закройте 135 порт – тогда вирусы и троянские кони не смогут распространяться.

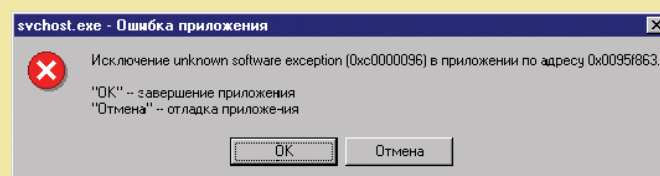


Рисунок 3. Это не признак нестабильности системы. Это – признак вирусной атаки! Если нажать на «ОК» – перестанет работать буфер обмена и некоторые другие функции системы, если нажать на «Отмену» – запустится отладчик (если он у вас есть) и система полностью встанет. Если же не делать ни того, ни другого – система успешно продолжит свою работу...

упакован и/или зашифрован. Зачастую расшифровщик крайне примитивен и состоит из десятка-другого машинных команд, смысл которых понятен с первого взгляда. В таком случае распаковать файл можно и самостоятельно. Вам даже не придется выходить из дизассемблера, всю работу можно выполнить и на встроенном языке (если, конечно, ваш дизассемблер поддерживает такой язык). Для расшифровки простейших «ксорок» хорошо подходит HIEW, а задачи посложнее решаются с помощью IDA. Подробное изложение методики расшифровки исполняемых файлов вы найдете в книге «Образ мышления – дизассемблер IDA» Криса Касперски.

Листинг 1. Пример типичного «ксороного» расшифровщика с комментариями

```
; CODE XREF: sub_401090+58?j
.text:004010DA loc_4010DA:
; загрузить в DL следующий байт
.text:004010DA mov dl, [esp+ecx+0Ch]
; расшифровать по XOR 66h
.text:004010DE xor dl, 66h
; положить на место
.text:004010E1 mov [esp+ecx+0Ch], dl
; увеличить счетчик на единицу
.text:004010E5 inc ecx
; еще есть что расшифровывать?
.text:004010E6 cmp ecx, eax
; ..если да, то мотаем цикл
.text:004010E8 jl short loc_4010DA
; CODE XREF: sub_401090+48?j
.text:004010EA loc_4010EA:
```

Если же код расшифровщика по своей дремучести напоминает непроходимый таежный лес, у исследования есть все основания считать, что исследуемая программа упакована одним из навесных упаковщиков, к которым, в частности, принадлежат ASPack, UPX, NeoLite и другие. Отождествить конкретный упаковщик при наличии достаточного опыта можно и самостоятельно (даже полиморфные упаковщики легко распознаются визуально, стоит только столкнуться с ними три-пять раз кряду), а во всех остальных случаях вам помогут специальные сканеры, самым известным (и мощным!) из которых является бесплатно распространяемый PESCAN (<http://k-line.cjb.net/tools/pe-scan.zip>). Давайте возьмем файл с вирусом Worm.Win32.Lovesan (также известный под именем MSblast) и натравим на него PE-SCAN. Сканер тут же сообщит, что вирус упакован упаковщиком UPX, который можно скачать с сервера upx.sourceforge.net, а при нажатии на кнопку «ОЕР» определит и адрес оригинальной точки в файл (в данном случае она равна 0x00011CB). Ну коль скоро мы знаем имя упаковщика, найти готовый распаковщик не составит больших проблем («UPX» + «unpack» в любом поисковике)¹. С другой стороны, знание оригинальной точки входа в файл позволяет установить на этот адрес точку останова и тогда в момент передачи управления только что распакованному файлу, отладчик немедленно всплывет (внимание! установка программной точки останова с кодом CSh в подавляющем большинстве случаев приводит к краху распаковщика, для предотвращения которого следует воспользоваться аппаратными точками останова; за подробностями обращайтесь к руководству пользо-

вателя вашего отладчика, в частности, в soft-ice установка аппаратной точки останова осуществляется командой BPM адрес X).

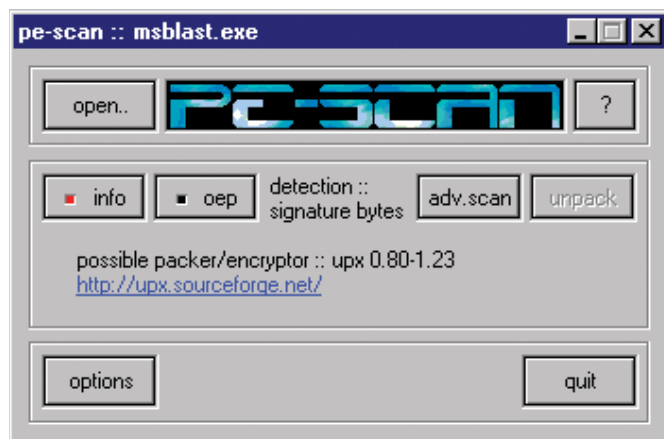


Рисунок 4. PE-SCAN в действии

А как быть, если PE-SCAN не сможет определить оригинальную точку входа или ни один из найденных вами распаковщиков не справляется с данным файлом? Если исследуемый файл хотя бы однократно запускался (то есть ваша машина уже потенциально заражена), можно взять ProcDump (<http://Procdump32.cjb.net>) и, запустив распаковываемый файл еще раз, снять с него полный дамп памяти (Task → имя процесса → Dump Fill). Конечно, чтобы полученный дамп превратился в полноценный PE-файл, над ним придется как следует поработать «руками», но для дизассемблирования сойдет и так. Шансы распаковать файл без его запуска средствами ProcDump относительно невелики, да и качество распаковки оставляет желать лучшего. Зачастую распакованный файл не пригоден даже для дизассемблирования, не то что запуска!

На худой конец можно попробовать перехватить передачу управления распакованному коду, просто поставив на соответствующие API-функции точки останова. При определенных навыках работы с двоичным кодом мы имеем все шансы осуществить такой перехват еще до того, как вирус успеет внедриться в систему или что-то испортить в ней, однако никаких гарантий на этот счет у нас нет и вирус в любой момент может вырваться из-под контроля, поэтому исследования такого рода лучше всего проводить на отдельной машине.

Итак, вызываем soft-ice и устанавливаем точки останова на все потенциально опасные функции, а также все те функции, которые обычно присутствуют в стартовом коде (см. раздел «Стартовый код»). Если вирус написан на языке высокого уровня, мы захватим выполнение еще до начала выполнения функции main. В противном случае отладчик всплывет при первой попытке выполнения потенциально опасной функции.

Отладчики, устанавливающие глобальные точки останова (и soft-ice в их числе), всплывают независимо от того, какое приложение их вызывает, поэтому всегда обращайте внимание на правый нижний угол экрана, в котором soft-ice выводит имя процесса, «потревожившего» отладчик, и, если это не исследуемый вами процесс, а что-то еще, вы можете смело покинуть отладчик, дожидаясь его очередного всплытия.

Таблица 1. Функции, помогающие перехватить управление у распакованного вирусного кода, получившего управление

основные функции стартового кода	основные потенциально опасные функции
GetVersion	CreateFileA,
GetVersionExA	RegOpenKeyA
GetCommandLineA	RegOpenKeyExA
GetModuleHandleA	LoadLibraryA
GetStartupInfoA	FindFirstFileA
SetUnhandledExceptionFilter	FindFirstFileExA

Давайте продемонстрируем технику ручной распаковки на примере анализа вируса I-Worm.Sobig.f. PE-SCAN показывает, что он упакован полиморфным упаковщиком TeLock 0.98 (<http://egoiste.cjb.net/>), однако найти готовый распаковщик в Интернете для этой версии не удастся (те, что есть, не распаковывают файл совсем или распаковывают его неправильно). Пошаговая трассировка распаковщика (равно как и попытки анализа алгоритма распаковки) заводят нас в никуда, ибо код оказывается слишком сложен для начинающих (а вот опытные программисты получают от его реконструкции настоящее удовольствие, ибо упаковщик весьма неплох). Просмотр дампа в HEX-редакторе также не показывает ничего подозрительного. Тупик...

А теперь на сцену выходит наш прием с контрольными точками, и исследуемая программа тотчас ловится на GetModuleHandleA и CreateFileA. На момент вызова последних весь код и все данные зараженного файла уже полностью распакованы и просмотр содержимого сегмента данных немедленно разоблачает вирус по агрессивным текстовым строкам:

```
001B:00419561 62 6C 65 00 00 00 00 62-61 73 65 36 34 00 00 53 ble....base64.S
001B:00419571 4D 54 50 00 00 00 00 74-63 70 00 74 65 78 74 2F MTP....tcp.txt/
001B:00419581 70 6C 61 69 6E 00 00 69-73 6F 2D 38 38 35 39 2D plain...iso-8859-
001B:00419591 31 00 00 51 55 49 54 0D-0A 00 00 45 48 4C 4F 20 L..QUIT....EHLO
001B:004195A1 25 73 0D 0A 00 00 00 50-61 73 77 6F 72 64 3A %s....Password:
001B:004195B1 00 00 00 55 73 65 72 6E-61 6D 65 3A 00 00 00 41 ...Username:...A
001B:004195C1 55 54 48 20 4C 4F 47 49-4E 0D 0A 00 00 00 4D UTH LOGIN.....M
001B:004195D1 41 49 4C 20 46 52 4F 4D-3A 20 3C 25 73 3E 0D 0A AIL FROM: <%s>..
001B:004195E1 00 00 00 52 43 50 54 20-54 4F 3A 20 3C 25 73 3E ...RCPT TO: <%s>
001B:004195F1 00 00 00 24 5C 00 00 53-4F 46 54 57 41 52 45 5C ...S\SOFTWARE\
001B:00419601 4D 69 63 72 6F 73 6F 66-74 5C 57 69 6E 64 6F 77 Microsoft/Window
001B:00419611 73 5C 43 75 72 72 65 6E-74 56 65 72 73 69 6F 6E s/CurrentVersion
001B:00419621 5C 52 75 6E 00 00 00 20-2F 73 69 6E 63 00 00 64 \Run.../sinc.d
001B:00419631 62 78 00 68 6C 70 00 6D-68 74 00 77 61 62 00 68 bx.hlp.mht.wab.h
```

Рисунок 5. Распакованный вручную I-Worm.Sobig.f сразу же выдает агрессивность своих намерений характерными текстовыми строками

Стартовый код

В девяностых годах двадцатого века, когда вирусы создавались преимущественно на ассемблере и писались преимущественно профессионалами, а коммерческие программисты в своем подавляющем большинстве полностью отказались от ассемблера и перешли на языки высокого уровня, для разработчиков антивирусов наступили золотые дни, ибо распознать зараженный файл зачастую удавалось с одного взгляда. Действительно, любя нормально откомпилированная программа начинается с так называемого стартового кода (Startup code), который легко отождествить визуально (как правило, он начинается с вызова функций GetVersion, GetModuleHandleA и т. д.), а дизассемблер IDA и вовсе идентифицирует стартовый код по обширной библиотеке сигнатур, выдавая тип и версию компилятора. Ассемблерные же программы стартового кода лишены, и потому, когда ассемблерный вирус внедряется в программу, написанную на языке высокого уровня, стартовый код отодвигается как бы

«вглубь» файла, демаскируя тем самым факт своего заражения.

Сегодня, когда ассемблерные вирусы становятся музейной редкостью, такой способ отождествления малопомалу перестает работать, однако до полного списывания в утиль дело не дошло.

Строго говоря, никаких формальных признаков «нормального» start-up не существует и каждый разработчик волен реализовывать его по-своему. Однако редкий разработчик цепляет к программе свой собственный start-up и все чаще для этих целей используется библиотечный стартовый код, поставляемый вместе с компилятором. Несмотря на то что даже в рамках одного-единственного компилятора существует множество разновидностей стартового кода, все они легко узнаваемы, и факт отсутствия стартового кода надежно обнаруживается даже самыми начинающими из исследователей!

Приблизительная структура типичного стартового кода такова: сначала идет пролог, затем настройка обработчика структурных исключений (для Си++ программ), обнаруживающая себя по обращению к сегментному регистру FS. Далее следует вызов функций GetVersion (GetVersionEx), GetModuleHandleA и GetStartupInfoA. Подробнее об идентификации стартового кода можно прочитать в книге «Фундаментальные основы хакерства» Криса Касперски или в «Hacker Disassembling Uncovered» его же.

Здесь же мы не можем позволить себе подробно останавливаться на этом обширном вопросе и просто сравним start-up код нормальной программы с кодом вируса Win2K.Inta.1676:

Листинг 2. Так выглядит нормальный start-up от Microsoft Visual C++ 6.0...

```
.text:00401670 start proc near
.text:00401670 push ebp
.text:00401671 mov ebp, esp
.text:00401673 push 0FFFFFFFh
.text:00401675 push offset stru_420218
.text:0040167A push offset _except_handler3
.text:0040167F mov eax, large fs:0
.text:00401685 push eax
.text:00401686 mov large fs:0, esp
; Get current version number of Windows
.text:00401696 call ds:GetVersion
.text:004016EC push 0
.text:004016EE call _heap_init
.text:00401704 mov [ebp+var_4], 0
.text:0040170B call _ioint
.text:00401710 call ds:GetCommandLineA
.text:00401716 mov dword_424F44, eax
.text:0040171B call _crtGetEnvironmentStringsA
.text:00401720 mov dword_4235C0, eax
.text:00401725 call _setargv
.text:0040172A call _setenvp
.text:0040172F call _cinit
.text:00401754 call _main
.text:00401763 call _exit
```

Листинг 3. ...а так выглядят окрестности точки входа вируса Win2K.Inta.1676

```
.text:00011000 start proc near
.text:00011000 mov eax, [esp+arg_0]
.text:00011004 lea edx, loc_11129
.text:0001100A mov [eax+34h], edx
.text:0001100D lea edx, dword_117A0
.text:00011013 lea eax, aHh ; "HH"
.text:00011019 mov [edx+8], eax
.text:0001101C mov [eax+4], aSystemrootSyst
.text:0001101C ; "\\SystemRoot\\system32\\drivers\\inf.sys"
```



```
.text:00011023      push    1200h
.text:00011028      push    0
.text:0001102D      call    ExAllocatePool
.text:00011032      or      eax, eax
.text:00011034      jnz     short loc_1103E
.text:00011036      mov     eax, 0C0000001h
.text:0001103B      retn    8
```

Смотрите! В то время как «хорошая» программа лениво опрашивает текущую версию операционной системы и иже с ней, зловерный вирус сломя голову несется в объятья драйвера `inf.sys`. Правильные программы так себя не ведут и коварность вирусных планов разоблачается с первого взгляда!

Разумеется, отсутствие стартового кода еще не есть свидетельство вируса! Быть может, исследуемый файл был упакован или разработчик применил нестандартный компилятор или набор библиотек. Ну с упаковкой мы уже разобрались, а с идентификацией компилятора поможет справиться IDA и наш личный опыт (замечание: текущие версии IDA PRO определяют версию компилятора непосредственно по стартовому коду, и, если он отсутствует или был изменен, механизм распознавания деактивируется и поиском подходящей библиотеки сигнатур нам приходится заниматься вручную, через меню `File → Load file → FLIRT signature file`). И, если обнаружится, что нормальный `start-up` у файла все-таки есть, но выполнение программы начинается не с него, шансы на присутствие вируса значительно возрастут!

Троянские программы, в большинстве своем написанные на языках высокого уровня, имеют вполне стандартный `Start-Up` и потому на такую наживку не обнаруживаются. Взять, например, того же `Kilez`, стартовый код которого выглядит так:

Листинг 4. Червь `I-Worm.Kilez.h` имеет стандартный стартовый код

```
.text:00408458 start      proc near
.text:00408458      push    ebp                ; sub_408458
.text:00408459      mov     ebp, esp
.text:0040845B      push    0FFFFFFFFh
.text:0040845D      push    offset stru_40D240
.text:00408462      push    offset __except_handler3
.text:00408467      mov     eax, large fs:0
.text:0040846D      push    eax
.text:0040846E      mov     large fs:0, esp
.text:0040847B      mov     [ebp+var_18], esp
; Get current version number of Windows
.text:0040847E      call    ds:GetVersion
.text:004084AF      xor     esi, esi
.text:004084B1      push    esi
.text:004084B2      call    __heap_init
.text:004084C4      mov     [ebp+var_4], esi
.text:004084C7      call    __ioint
.text:004084CC      call    ds:GetCommandLineA
.text:004084D2      mov     dword_494E68, eax
.text:004084D7      call    crtGetEnvironmentStringsA
.text:004084DC      mov     dword_493920, eax
.text:004084E1      call    __setargv
.text:004084E6      call    __setenvp
.text:004084EB      call    __cinit
.text:004084F0      mov     [ebp+StartupInfo.dwFlags], esi
.text:004084F3      lea     eax, [ebp+StartupInfo]
.text:004084F6      push    eax                ; lpStartupInfo
.text:004084F7      call    ds:GetStartupInfoA
.text:004084FD      call    __wincmdln
```

Даже «невооруженным» глазом видно, что стартовый код червя идентичен стартовому коду Microsoft Visual C++ 6.0, что и неудивительно, так именно на нем червь и написан.

Точка входа

При внедрении вируса в файл точка входа в него неизбежно изменяется. Лишь немногие из вирусов ухитряются заразить файл, не прикасаясь к точке останова (вирус может вписать по адресу оригинальной точки останова `JUMP` на свое тело, слегка подправить таблицу перемещаемых элементов или вклиниться в массив RVA-адрес таблицы импорта, внедриться в незанятые области кодовой секции файла и т. д.), однако ареал обитания таких особей ограничен преимущественно стенами лабораторий, а в дикой природе они практически не встречаются. Не тот уровень подготовки у вирусописателей, не тот...

И если «нормальные» точки входа практически всегда находятся в кодовой секции исполняемого файла («.text»), точнее, в гуще библиотечных функций (навигатор IDA PRO по умолчанию выделяет их голубым цветом), непосредственно предшествуя секции данных, то точка входа зараженного файла традиционно располагается между секцией инициализированных и неинициализированных данных, практически у самого конца исполняемого файла.

Так происходит потому, что дописывая свое тело в конец файла, «вирусная» секция оказывается самой последней секцией инициализированных ячеек памяти, за которой простирается обширный регион неинициализированных данных, без которого не обходится ни одна программа. Это-то его (вируса) и демаскирует!

Ни один из известных автору упаковщиков исполняемых файлов так себя не ведет, и потому ненормальное расположение точки входа с высокой степенью вероятности свидетельствует о заражении файла вирусом!

Нестандартные секции

При заражении исполняемого файла методом дозаписи своего тела в его конец у вируса есть две альтернативы: либо увеличить размер последней секции файла (а это, как правило, секция `.data`), присвоив ей атрибут `Executable`², либо создать свою собственную секцию. Оба этих способа легко распознаются визуально.

Код, расположенный в конце секции данных, — весьма характерный признак вируса, равно как и секция `.text`, замыкающая собой файл и после недолгих мытарств передающая управление «вперед» — на нормальную точку входа. То же самое относится и к секциям с нестандартными именами, зачастую совпадающими с именем самого вируса или маскирующимися под секции, создаваемые упаковщиками исполняемых файлов (но сам файл при этом остается неупакованным!).

Достаточно часто вирусы внедряются в создаваемую ими секцию `.reloc`, штатно предназначенную для хранения перемещаемых элементов. Быть может, вирусописатели думают, что разработчики антивирусов все сплошь круглые идиоты и не знают, для чего эта секция предназначена?

Но так или иначе, встретив в исследуемом файле секцию `.reloc`, содержащую исполняемый код, знайте: с вероятностью близкой к единице, вы имеете дело с вирусом.

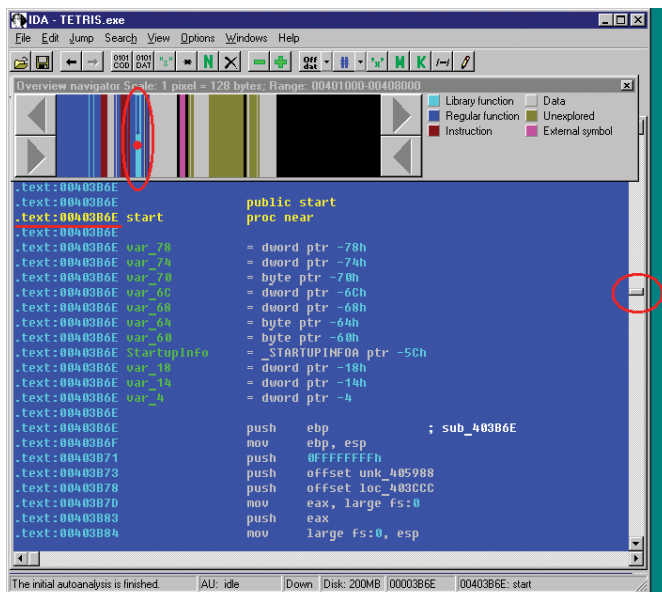


Рисунок 6. Так выглядит дизассемблерный листинг нормального файла. Точка входа расположена внутри секции .text в гуще библиотечных функций, приходится приблизительно на середину файла

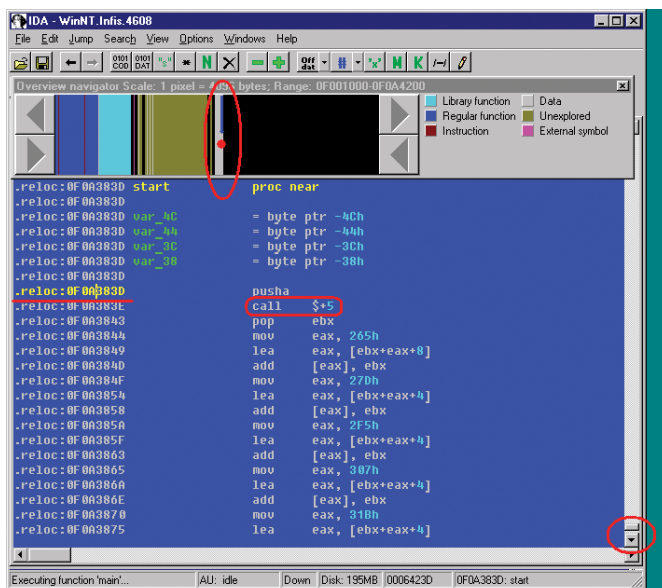
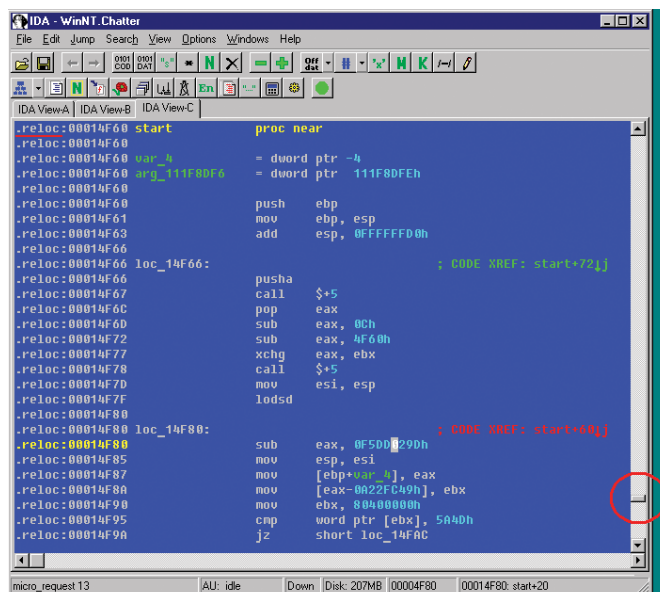


Рисунок 7. Так выглядит дизассемблерный листинг файла, зараженного вирусом WinNT.Infis.4608: точка входа расположена в секции .reloc, помещенной непосредственно за концом инициализированных данных у самой «кромки» исполняемого файла

Таблица импорта

Операционные системы семейства Windows поддерживают два основных способа компоновки: статический и динамический. При статической компоновке имена (ординалы) вызываемых API-функций выносятся в специальную таблицу – таблицу импорта, изучение которой дает более или менее полное представление о природе исследуемой программы и круге ее интересов. К потенциально опасным функциям в первую очередь относятся сетевые функции, функции поиска, вызова и удаления файлов, TOOLHELP-функции, используемые для просмотра списка активных процессов и внедрения в них...

Конечно, зловредной программе ничего не стоит загрузить все эти функции и самостоятельно, путем динамической компоновки, в простейшем случае опирающейся



Листинг 5. Так выглядит дизассемблерный листинг файла, зараженного вирусом WinNT.Chatter: исполняемый код, расположенный в секции .reloc, предназначенной для хранения перемещаемых данных, сигнализирует о ненормальности ситуации

ся на вызов LoadLibrary/GetProcAddress, а то и вовсе на «ручной» поиск API-функций в памяти (адрес системного обработчика структурных исключений дает нам адрес, принадлежащий модулю KERNEL32.DLL, базовый адрес которого определяется сканированием памяти на предмет выявления сигнатур «MZ» и «PE» с последующим разбором PE-заголовка), но в этом случае текстовые строки с именами соответствующих функций должны так или иначе присутствовать в теле программы (если только они не зашифрованы и не импортируются по ординалу).

Однако статистика показывает, что таблица импорта троянских программ носит резко полярный характер. Либо она вообще практически пуста, что крайне нетипично для нормальных, неупакованных, программ, либо содержит обращения к потенциально опасным функциям в явном виде. Конечно, сам факт наличия потенциально опасных функций еще не свидетельствует о троянской природе программы, но без особой нужды ее все-таки лучше не запускать.

Анализ таблицы импорта позволяет выявить также и ряд вирусных заражений. Собственно, у вируса есть два пути: использовать таблицу импорта файла-жертвы или создавать свою. Если необходимых вирусу API-функций у жертвы нет и она не импортирует функции LoadLibrary/GetProcAddress, вирус должен либо отказаться от ее заражения, либо тем или иным образом импортировать недостающие функции самостоятельно (некоторые вирусы используют вызов по фиксированным адресам, но это делает их крайне нежизнеспособными, ограничивая ареал обитания лишь теми версиями ОС, на которые явно закладывались вирусом-писателем; другие же определяют адреса функций самостоятельно: по сигнатурному поиску или ручным анализом таблицы импорта: первое – громоздко и ненадежно, второе – слишком сложно в реализации для начинающих).

И вот тут-то и начинается самое интересное. Разберем два варианта: использование готовой таблицы импорта и внедрение своей. На первый взгляд кажется, что отследить «левые» обращения к импорту жертвы просто нереально,

так как они ничем не отличаются от «нормальных». Теоретически все так и есть. Практически же не все так безнадежно. Большинство сред разработки компилирует программы с инкрементной линковкой и вместо непосредственного вызова всякой импортируемой функции, вызывает «переходник» к ней. Таким образом, каждая импортируемая функция вызывается лишь однажды и IDA генерирует лишь одну перекрестную ссылку. При заражении файла картина меняется и к API-функциям, используемым вирусом, теперь ведут две и более перекрестных ссылок.

Это вернейший признак вирусного заражения! Вернее и быть не может!

Листинг 6. «Заглушка», представляющая собой переходник к импортируемой функции и оттягивающая все перекрестные ссылки на себя

```
; CODE XREF: sub_432A58+C0?p
BRAT0:00648310 CreateFileA      proc near
; sub_432BC0+C0?p ...
BRAT0:00648310
BRAT0:00648310 FF 25 48 44+      jmp     ds:__imp_CreateFileA
BRAT0:00648310 CreateFileA      endp
```

Листинг 7. Таблица импорта исследуемого приложения: наличие «паразитной» ссылки на CreateFileA указывает на факт вирусного заражения

```
; DATA XREF: CreateDirectoryA?r
.idata:006A4440 extrn __imp_CreateDirectoryA:dword
; DATA XREF: CreateEventA?r
.idata:006A4444 extrn __imp_CreateEventA:dword
; DATA XREF: CreateFileA?r
.idata:006A4448 extrn __imp_CreateFileA:dword
; DATA XREF: sub_6A4140?r
.idata:006A4448
; DATA XREF: CreateProcessA?r
.idata:006A444C extrn __imp_CreateProcessA:dword
; DATA XREF: CreateThread?r
.idata:006A4450 extrn __imp_CreateThread:dword
```

А что, если вирус захочет создать собственную секцию импорта или как вариант попытается расширить уже существующую? Ну две секции импорта для операционных систем семейства Windows – это слишком! Хотя... Почему, собственно, нет? Вирус создает еще одну секцию импорта, дописывая ее в конец файла, копирует туда содержимое оригинальной таблицы импорта, добавляет недостающие API-функции и затем направляет поле Import Table на «свою» таблицу импорта. По факту загрузки файла операционной системой вирус проделывает обратную операцию, перетягивая таблицу импорта «назад» (необходимость последней операции объясняется тем, что система находит таблицу импорта по содержимому поля Import Table, а

непосредственно сам исполняемый файл работает с ней по фиксированным адресам). Наличие двух таблиц импорта в файле – верный признак его заражения!

Как вариант вирус может добавить к файлу секцию BOUND_IMPORT, что очень просто реализуется и, что самое интересное, обнаруживается далеко не всеми дизассемблерами! Откройте исследуемый файл в HIEW и посмотрите на 12-й элемент Header Data Directories. Если такой элемент действительно присутствует и хранит в себе нечто отличное от нуля, – вероятность присутствия вируса в файле становится весьма велика (хотя некоторые легальные программы, в частности, линкер ULINK Юрия Харона также содержат в себе секцию BOUND_IMPORT, но вирусами, очевидно, не являются).

Расширение уже существующей таблицы импорта менее наглядно, но при наличии опыта работы с PE-файлами его все-таки можно разоблачить. Так, большинство линкеров упорядочивают импортируемые функции по алфавиту и функции, дописанные вирусами в конец таблицы импорта, сразу же обращают на себя внимание. Даже если импорт и не отсортирован, повышенная концентрация характерных для вируса API-функций все равно не может не броситься в глаза. Действительно, перечисление имен всех импортируемых функций обычно идет сплошным потоком от первой до последней используемой DLL, причем библиотека KERNEL32.DLL (которая вирусу и нужна!) оказывается в конце списка достаточно редко и вирусу ничего не остается, как дописывать импорт из KERNEL32.DLL в хвост другой библиотеки, в результате чего ссылка на модуль KERNEL32.DLL в таблице импорта зараженного файла присутствует дважды!

Заключение

Вот мы и рассмотрели основные способы выявления зараженных файлов и теперь смело можем приступать к расширению и углублению полученных знаний и навыков. Чем больше вирусов пройдет через ваши руки, тем легче будет справиться с каждым из них. В конце концов не так страшны вирусы, как люди...

¹ Упаковщик UNPX также содержит в себе и распаковщик, хотя это скорее исключение, чем правило.

² Впрочем, под Windows 9x/NT этого можно и не делать, так как она разрешает выполнение кода в секции данных.

