техника внедрения и удаления кода из PE-файлов

крис касперски

настоящая статья предпринимает попытку систематизации и классификации существующих алгоритмов внедрения в PE32/PE64-файлы, способов визуальной идентификации потенциально опасного кода и методов его удаления, а так же дает некоторые рекомендации по предотвращению вирусного вторжения.
публикация рассчитана на широкий круг читателей, знакомых с языком Си, и имеющих опыт системного программирования на операционных системах семейства Windows 9x и Windows NT.

введение

Опасаясь яростных нападок пуристов, склонных рассматривать добрую половину аспектов системного программирования как покушение на авторское право, подрывающие фундамент личной безопасности, и всячески препятствующих публикациям подобного рода, автор вынужден начинать статью с оправданий и деклараций.

Свободу слова еще никто не отменял и сами по себе технологии внедрения в исполняемые файлы не могут быть ни хорошими, ни плохими. Вирусы (как компьютерные, так и биологические) – это неотъемлемая часть мироздания, несущая не только вред, но и пользу. С кишечной палочкой у нас установился взаимовыгодный симбиоз еще миллионы лет назад, компьютерные вирусы успешно "симбиотят" с протекторами, упаковщиками исполняемых файлов, поисковыми машинами, операционными системами и многими другими объектами виртуального мира, окружающими нас.

Механизмы внедрения в PE-файлы весьма разнообразны, но довольно поверхностно описаны в доступной литературе. Имеющиеся источники либо катастрофически неполны, либо откровенно неточны, да к тому же рассеянны по сотням различных FAQ'ax и tutorial'ов. Приходится, выражаясь словами Маяковского, перелопачивать тонны словесной руды, прежде чем обнаружится нечто полезное. Данная работа представляет собой попытку систематизации и классификации всех известных способов внедрения. Это самая полная коллекция из имеющихся! Во всяком случае, в открытой печати ничего подобно со времен MS-DOS ни разу не было опубликовано.

Статья будет интересна не только специалистам по информационной безопасности, специализирующихся на идентификации и удалении вирусов, но и разработчикам навесных защит конвертного типа и упаковщиков. Что же касается вирусописателей… Господа пуристы, да поймите же вы, наконец, что если человек задался целью написать вирус, то он его напишет! Публикации подобного рода на это *никак* не влияют! Автор ни к чему не призывает, и ни от чего не отговаривает. Это – прерогатива карательных органов власти, религиозных деятелей, ну и моралистов наконец. Моя же задача намного скромнее – показать какие пути внедрения существуют, на что обращать внимание при поиске постороннего кода и как отремонтировать файл, угробленный некорректным внедрением.

понятие x-кода и другие условные обозначения

Код, внедряющийся в файл, мы будем называть **X-кодом.** Под это определение попадает любой код, внедряемый нами в файл-носитель (он же подопытной файл или файл-хозяин от английского host-file), например, инструкция **NOP**. О репродуктивных способностях X-кода в общем случае ничего не известно и для успокоения будем считать, X-код несаморазмножающимся кодом. Всю ответственность за внедрение берет на себя ***человек***, запускающий программу-внедритель (нет, не "вредитель", а "внедритель" от слова "внедрить"), которая предварительно убедившись в наличии прав записи в файл-носитель (а эти права опять-таки дает человек) и его совместимости с выбранной стратегией внедрения, записывает X-код внутрь файла и осуществляет высокоманевренный перехват управления, так что подопытная программа не замечает никаких изменений.

Для экономии места в статье используются ряд общепринятых сокращений, перечисленных ниже:

- FA : File Alignment – физическое выравнивание секций;
- SA, OA : Section Alignment или Object Alignment – виртуальное выравнивание секций;

- RVA : Relative Virtual Address – относительный виртуальный адрес;
- FS : First Section – первая секция файла;
- LS : Last Section – последняя секция файла;
- CS : Current Section – текущая секция файла;
- NS : Next Section – следующая секция файла;
- v_a : Virtual Address – виртуальный адрес;
- v_sz : Virtual Size – виртуальный размер;
- r_off : raw offset – физический адрес начала секции;
- f_sz : raw size – физический размер секции;
- DDIR : DATA DIRECTORY – нет адекватного перевода;
- EP : Entry Point – точка входа;

Под **Windows NT** если не оговорено обратное, подразумевается вся линейка NT – подобных операционных систем: Windows NT 4.0/Windows 2000/Windows XP, а под Windows 9x – Windows 95, Windows 98 и Windows Me.

Под ***системным загрузчиком*** понимается компонент операционной системы, ответственный за загрузку исполняемых файлов и динамических библиотек.

Выше, левее, ***западнее –*** соответствует меньшим адресам, что совпадает с естественной схемой отображения дампа памяти отладчиком или дизассемблером.

Рисунок 1 условные графические обозначения принятые в статье

цели и задачи x-кода

Перед X-кодом стоят по меньшей мере три серьезных задачи: а) разместить свое тело внутри подопытного файла (или, как бы сказал Пелевин, слиться в ним в алхимическом браке); б) перехватить управление до начала выполнения основной программы или в процессе оного; в) определить адреса API-функций, жизненно важных для собственного функционирования.

Методология перехвата управления и определения адресов API-функций уже рассматривалась нами ранее в статьях "борьба с вирусамиопыт контртеррористических операций ", "вирусы в UNIX, или Гибель Титаника II" и "ошибки переполнения буфера извне и

изнутри как обобщенный опыт реальных атак" и поэтому здесь не описывается. Ограничимся тем, что напомним читателю основные моменты.

Перехват управления обычно осуществляется следующими путями:

- переустановкой точки входа на тело X-кода;
- внедрением в окрестности оригинальной точки входа команды перехода на X-код (естественно, перед передачей управления, X-код должен удалить команду, восстановив исходное содержимое EP);
- переустановкой произвольно взятой команды JMP/CALL на тело X-кода с последующей передачей управления по оригинальному адресу (этот прием не гарантирует, что X-коду вообще удастся заполучить управление, но зато обеспечивает ему феноменальную скрытность и максимальную защищенность от антивирусов);
- модификацией одного или нескольких элементов таблицы импорта с целью подмены вызываемых функций своими собственными (этой технологией в основном пользуются stealth-вирусы, умело скрывающие свое присутствие в системе);

Определение адресов API-функций обычно осуществляется следующими путями:

- поиском необходимых функций в таблице импорта файла-хозяина (будьте готовы к тому, что там их не окажется, либо отказываясь от внедрения, либо используя другую стратегию поиска);
- поиском LoadLibrary/GetProcAddress в таблице импорта файла-хозяина с последующим импортированием всех необходимых функций вручную (будьте готовы к тому, что и этих функций в таблице импорта также не окажется);
- прямым вызовом API-функций по их абсолютным адресам, жестко прописанным внутри X-кода (адреса функций KERNEL32.DLL/NTDLL.DLL непостоянны и меняются от одной версии к системы к другой, а адреса USER32.DLL и всех остальных пользовательских библиотек непостоянны даже в рамках одной конкретной системы и варьируются в зависимости от Image Base остальных загружаемых библиотек, поэтому при всей популярности данного способа пользоваться им допустимо только в образовательно-познавательных целях);
- добавлением в таблицу импорта необходимых X-коду функций, которыми как правило, являются LoadLibrary/GetProcAddress, с помощью которых можно вытащить из недр системы и все остальные (достаточно надежный, хотя и слишком заметный способ);
- непосредственным поиском функций LoadLibrary/GetProcAddress в памяти – поскольку KERNEL32.DLL проецируется на адресное пространство всех процессов, а ее базовый адрес всегда выровнен на границу в 64 Кбайта, от нас всего лишь требуется просканировать первую половину адресного пространства процесса, на предмет поиска сигнатуры "MZ". Если такая сигнатура найдена – убеждаемся в наличии сигнатуры "PE", расположенной по смещению e_lfanew от начала базового адреса загрузки. Если она действительно присутствует, анализируем DATA DIRECTORY и определяем адрес таблицы экспорта, в которой требуется найти LoadLibraryA и GetProcAddress. Если же хотя бы одно из этих условий не совпадает, уменьшаем указатель на 64 Кб и повторяем всю процедуру заново. Пара соображений в подмогу: прежде чем что-то читать из памяти, вызовете функцию IsBadReadPtr, убедившись, что вы вправе это делать; помните, что Windows 2000 Advanced Server и Datacenter Server поддерживают загрузочный параметр /3GB, предоставляющий в распоряжение процесса 3 Гбайта оперативной памяти и сдвигающий границу сканирования на 1 Гбайт вверх; для упрощения отождествления KERNEL32.DLL можно использовать поле Name RVA, содержащееся в Export Directory Table и указывающее на имя динамической библиотеки, однако, оно может быть и подложным (системный загрузчик его игнорирует);
- определением адреса функции KERNEL32!_except_handler3, на которую указывает обработчик структурных исключений по умолчанию. Эта функция не экспортируется ядром, однако, присутствует в отладочной таблице символов, которую можно скачать со следующего сервера http://msdl.microsoft.com/download/symbols (внимание! сервер не поддерживает просмотр браузером и с ним работают только последние версии Microsoft Kernel Debugger и NuMega Soft-Ice). Это делается так: mov esi, fs:[0]/lodsd/lodsd. После выполнения кода, регистр EAX содержит адрес, лежащий где-то в глубине KERNEL32. Выравниваем его по границе 64 Кб и ищем MZ/PE сигнатуры, как показано в

предыдущем пункте (это наиболее корректный и надежный способ поиска, всячески рекомендуемый к употреблению);

- определением базового адреса загрузки KERNEL32.DLL через PEB: mov eax, fs:[30h]/mov eax, [eax + 0Ch]/mov esi, [eax + 1Ch]/lodsd/mov ebx, [eax + 08h], – базовый код возвращается в регистре EBX; (это очень простой, хотя и ненадежный прием, т.к. структура PEB в любой момент может измениться, и за все время существования Windows она уже менялась по меньшей мере три раза, к тому же PEB есть только в NT);
- использованием native API операционной системы, взаимодействие с которым осуществляется либо через прерывание INT 2Fh (Windows 3.x, Windows 9x), либо через прерывание INT 2Eh (Windows NT, Windows 2000), либо через машинную команду syscall (Windows XP). Краткий перечень основных функций можно найти в Interrupt List'e Ральфа Брауна, бесплатно распространяемого через Интернет http://www.pobox.com/~ralf/files.html (это наиболее трудоемкий и наименее надежный способ из всех, мало того, что native API-функции не только недокументированны и подвержены постоянным изменениям, так они еще и до безобразия примитивны, в смысле реализуют простейшие низкоуровневые функции, непригодные к непосредственному использованию);

Принципы внедрения X-кода в PE-файлы с технической точки зрения практически ничем не отличаются от ELF, разве что именами служебных полей и стратегией их модификации. Однако, детальный анализ спецификаций и дизассемблирование системного загрузчика выявляет целый пласт тонкостей, неизвестных даже профессионалов (во всяком случае, ни один известный мне протектор/упаковщик не избежал грубых ошибок проектирования и реализации).

Существуют следующие способы внедрения: а) размещение X-кода поверх оригинальной программы (так же называемое затиранием); б) размещение X-кода в свободном месте программы (интеграция); в) дописывание X-кода в начало, середину или конец файла с сохранением оригинального содержимого; г) размещение X-кода вне основного тела файла-носителя (например, в динамической библиотеке или NTFS-потоке), загружаемого "головой" X-кода, внедренной в файл способами (а), (б) или (в).

Поскольку, способ (а) приводит к необратимой потере работоспособности исходной программы и реально применяется только в вирусах, здесь он не рассматривается. Все остальные алгоритмы внедрения полностью или частично обратимы.

требования, предъявляемые к X-коду

X-код следует проектировать с учетом всей жесткости требований, предъявляемых неизвестной и под час очень агрессивной средой чужеродного кода, в которое он будет заброшен, в смысле внедрен.

Во-первых, X-код должен быть полностью перемещаем, т. е. сохранять свою работоспособность независимо от базового адреса загрузки. Это достигается использованием относительной адресации: определив свое текущее расположение вызовом команды CALL $+5/POP EBP, X-код сможет преобразовать смещения внутри своего тела в эффективные адреса простым сложением их с EBP. Разумеется, это не единственная схема. Существуют и другие, однако, мы не будем на них останавливаться, поскольку к PE-файлам они не имеет не малейшего отношения.

Во-вторых, грамотно сконструированный X-код никогда не модифицирует свои ячейки, поскольку не знает: имеется ли у него правда на запись или нет. Стандартная секция кода лишена атрибута IMAGE_SCN_MEM_WRITE и присваивать его крайне нежелательно, т. к. это не только демаскирует X-код, но и снижает иммунитет программы-носителя. Разумеется, при внедрении в секцию данных, это ограничение теряет свою актуальность, однако, далеко не во всех случаях, запись в секцию данных разрешена. Оптимизм – это прекрасно, но программист должен закладываться на наихудший вариант развития событий. Разумеется, это еще не обозначает, что X-код не может быть самомодифицирующимся или не должен модифицировать никакие ячейки памяти вообще! К его услугам и стек (автоматическая память), и динамическая память (куча), и кольцевой стек сопроцессора наконец!

В третьих, X-код должен быть предельно компактным, поскольку объем пространства, пригодного для внедрения под час очень даже ограничен (можно даже сказать "драконичен").

Имеет смыл разбить X-код на две части: крошечный загрузчик и протяжный хвост. Загрузчик лучше всего разместить в PE-заголовке или регулярной последовательности внутри файла, а хвост сбросить в оверлей или NTFS-поток, комбинируя тем самым различные методы внедрения.

Наконец, X-код не может позволить себе задерживать управление более чем на несколько сотых, ну от силы десятых долей секунд, в противном случае, факт внедрения станет слишком заметным и будет сильно нервировать пользователя, чего допускать ни в коем случае нельзя.

внедрение

Перед внедрением в файл необходимо убедиться, что он не является драйвером, не содержит нестандартных таблиц в DATA DIRECTORY и доступен для модификации. Присутствие оверлеев крайне нежелательно и без особой необходимости в оверлейный файл лучше ничего не внедрять, а если и внедрять то придерживаться наиболее безболезненной стратегии внедрения – стратегии A.

Ниже все эти требования разобраны подробнее:

- если файл расположен на носителе, защищенном от записи, или у нас недостаточно прав для его записи/чтения (например, файл заблокирован другим процессом), отказывается от внедрения;
- если файл имеет атрибут, запрещающий модификацию, либо снимаем этот атрибут, либо отказываемся от внедрения;
- если поле Subsystem > 2h или Subsystem < 3h отказываемся от внедрения;
- если FA < 200h или SA < 1000h это, вероятнее всего, драйвер и в него лучше ничего не внедрять;
- если файл импортирует одну или несколько функций из hal.dll и/или ntoskrnl.exe, отказываемся от внедрения;
- если файл содержит секцию INIT, он, возможно, является драйвером устройства, а возможно и нет, но без особой нужны лучше сюда ничего не внедрять;
- если DATA DIRECTIRY содержит ссылки на таблицы, использующие физическую адресацию, либо отказываемся от внедрения, либо принимаем на себя обязательства корректно "распотрошить" все иерархию структур данных и скорректировать физические адреса;
- если ALIGN_UP(LS.r_off + LS.r_sz, A) > SizeOfFile, файл скорее всего содержит оверлей и внедряться в него можно только по методу A.
- если физический размер одной или нескольких секций превышает виртуальный на величину большую или равную FA и при этом виртуальный размер не равен нулю, подопытный файл содержит оверлей, допуская тем самым использовать внедрения только типа A.

Следует помнить о необходимости восстановления атрибутов файла и времени его создания, модификации и последнего доступа (большинство разработчиков ограничивается одним лишь временем модификации, что демаскирует факт внедрения).

Если поле контрольной суммы не равно нулю, следует либо оставить такой файл в покое, либо рассчитать новую контрольную сумму самостоятельно, например, путем вызова API-функции CheckSumMappedFile. Обнулять контрольную сумму, как это делают некоторые, категорически недопустимо, т. к. при активных сертификатах безопасности, операционная система просто откажет файлу в загрузке!

Еще несколько соображений общего типа. В последнее время все чаще и чаще приходится сталкиваться с исполняемыми файлами чудовищного объема, неуклонно приближающегося к отметке в несколько гигабайт. Обрабатывать таких монстров по кускам – нудно и сложно. Загружать весь файл целиком – слишком медленно, да и позволит Windows выделить такое количество памяти! Поэтому имеет смысл воспользоваться файлами проецируемыми в память (Memory Mapped File), управляемыми функциями CreateFileMapping и MapViewOfFile/UnmapViewOfFile. Это не только увеличивает производительность, упрощает программирование, но и ликвидирует все ограничения на предельно допустимый объем, который теперь может достигать 18 экзобайтов, что соответствует 1.152.921.504.606.846.976 байтам). Как вариант, можно ограничить размер обрабатываемых файлов несколькими мегабайтами, легко копируемыми в оперативный буфер и сводящими

количество "обвязочного" кода к минимуму (кто работал с файлами от 4х Гбайт и выше, тот поймет).

предотвращение повторного внедрения

В то время как средневековые алхимики пытались создать Алмогест – универсальный растворитель, растворяющий вся и все – их оппоненты язвительно замечали: задумайтесь, в чем вы его будете хранить? И хотя алмогест так и не был изобретен, его идея не умерла и до сих пор будоражит умы вирусописателей, вынашивающих идею принципиально недетектируемого вируса. Может ли существовать такой вирус хотя бы в принципе? И если да, то как он сможет отличать уже инфицированные файлы от еще не зараженных? В противном случае заражения одного и тоже файла будут происходить многократно и навряд ли многочисленные копии вирусов смогут мирно соседствовать с друг другом.

X-код, сохраняющий работоспособность даже при многократном внедрении, называют рентабельным. Рентабельность предъявляет жесткие требования как к алгоритмам внедрения в целом, так и к стратегии поведения X-кода в частности. Очевидно, что X-код, внедряющийся в MS-DOS заглушку, рентабельным не является и каждая последующая копия затирает собой предыдущую. Протекторы, монополизирующие системные ресурсы с целью противостояния отладчикам (например, динамически расшифровывающие/зашифровывающие защищаемую программу путем перевода страниц памяти в сторожевой режим с последующим перехватом прерываний), будут конфликтовать друг с другом, вызывая либо зависание, либо сбой программы. Классическим примером рентабельности является X-код, дописывающий себя в конец файла и после совершения всех запланированных операций возвращающий управление программе-носителю. При многократном внедрении, X-коды как бы "разматываются", передавая управление словно по эстафете, однако, если нить управления запутается, все немедленно рухнет. Допустим, X-код привязывается к своему физическому смещению, отсчитывая его относительно конца файла. Тогда, при многократном внедрении по этим адресам будут расположены совсем другие ячейки, принадлежащие чужому X-коду и поведение обоих станет неопределенным.

Перед внедрением в файл нерентабельного X-кода необходимо предварительно убедиться, что в файл не был внедрен кто-то еще. К сожалению, универсальных путей решения не существует и приходится прибегать к различным эвристическим приемам, распознающим присутствие инородного X-кода по косвенным признакам.

Родственные X-коды всегда могут "договориться" друг с другом, отмечая свое присутствие уникальной сигнатурой. Например, если файл содержит строку "x-code ZANZIBAR here", отказываемся от внедрения на том основании, что здесь уже есть "свои". К сожалению, этот трюк очень ненадежен и при обработке файла любым упаковщиком/протектором, сигнатура неизбежно теряется. Ну, разве что, внедрить сигнатуру в ту часть секции ресурсов, которую упаковщики/протекторы предпочитают не трогать (иконка, информация о файле и т. д.). Еще надежнее внедрять сигнатуру в дату/время последней модификации файла (например, в десятые доли секунды). Упаковщики/протекторы ее обычно восстанавливают, однако, короткая длина сигнатуры вызывает большое количество ложных срабатываний, что тоже нехорошо.

Не родственным X-кодам приходится намного хуже. Чужих сигнатур они не знают и потому не могут наверняка утверждать – возможно ли осуществить корректное внедрение в файл или нет? Поэтому, X-код, претендующий на корректность, обязательно должен быть рентабельным, в противном случае, сохранение работоспособности файлам уже не гарантированно.

Упаковщики оказываются в довольно выигрышном положении: дважды один файл не сожмешь и если коэффициент сжатия окажется исчезающее мал, упаковщик вправе отказаться обрабатывать такой файл. Протекторам – другое дело. Протектор, отказывающийся обрабатывать уже упакованные (зашифрованные) файлы мало кому нужен. Если протектор монополизирует ресурсы, отказываясь их предоставлять кому-то еще, он должен обязательно контролировать целостность защищенного файла и обнаружив внедрение посторонних, выводить соответствующее предупреждение на экран, возможно, прекращая при этом работу. В противном случае, защищенный файл могут упаковать и попытаться защитить повторно. Последствия такой защиты не заставят себя ждать…

классификация механизмов внедрения

Механизмы внедрения можно классифицировать по-разному: по месту (начало, конец, середина), по "геополитике" (затирание исходных данных, внедрение в свободное пространство, переселение исходных данных на новое место обитания), по надежности (предельно корректное, вполне корректное и крайне некорректное внедрение), по рентабельности (рентабельное или нерентабельное) и т. д. Мы же будем отталкиваться от *характера воздействия на физический и виртуальный образ подопытной программы*, разделив все существующие механизмы внедрения на четыре категории, обозначенных латинскими буквами A, B, C и Z.

- К **категории A** относятся механизмы, не вызывающие изменения адресации ни физического, ни виртуального образов. После внедрения в файл ни его длина, ни количество выделенной при загрузке памяти не изменяется и все базовые структуры остаются на своих прежних адресах. Этому условию удовлетворяют: внедрение в пустое место файла (PE-заголовок, хвосты секций, регулярные последовательности), внедрение путем сжатия части секции и создание нового NTFS-потока внутри файла[1];
- К **категории B** относятся механизмы, вызывающие изменения адресации только физического образа. После внедрения в файл его длина увеличивается, однако количество выделенной при загрузке памяти не изменяется и все базовые структуры проецируется по тем же самым адресам, однако, их физические смещения изменяются, что требует полной или частичной перестройки структур, привязывающихся к своим физическим адресам и, если хотя бы одна из них останется не скорректированной (или будет скорректирована неправильно), файл-носитель с высокой степенью вероятности откажет в работе. Категории B соответствуют: раздвижка заголовка, сброс части оригинального файла в оверлей и создание своего собственного оверлея;
- К **категории C** относятся механизмы, вызывающие изменения адресации как физического, так и виртуального образов. Длина файла и выделяемая при загрузке память увеличиваются. Базовые структуры могут либо оставаться на своих местах (т.е. изменяются лишь смещения, отсчитываемые от конца образа/файла), либо перемещаться по страничному имиджу произвольным образом, требуя обязательной коррекции. Этой категории соответствует: расширение последней секции файла, создание своей собственной секции и расширение серединных секций.
- К "засекреченной" **категории Z** относятся механизмы, вообще не дотрагивающиеся до файла-носителя и внедряющиеся в его адресное пространство косвенным путем, например, модификацией ключа реестра, ответственного за автоматическую загрузку динамических библиотек. Этой технологией интересуются в первую очередь сетевые черви и шпионы. Вирусы к ней равнодушны.

Категория A наименее конфликтна и приводит к отказу лишь тогда, когда файл контролирует свою целостность. Сфера применений категорий B и C гораздо более ограничена, в частности, она не неспособна обрабатывать файлы с отладочной информацией, поскольку отладочная информация практически всегда содержит большое количество ссылок на абсолютные адреса. Ее формат недокументирован и к тому же различные компиляторы используют различные форматы отладочной информации, поэтому скорректировать ссылки на новые адреса нереально. Помимо отладочной информации еще существуют сертификаты безопасности и прочие структуры данных, нуждающиеся в неприкосновенности своих смещений. К сожалению, механизмы внедрения категории A, налагают достаточно жесткие ограничения на предельно допустимый объем X-кода, определяемый количеством свободного пространства, имеющегося в программе, и достаточно часто здесь не находится места даже для крохотного загрузчика, поэтому приходится идти на вынужденный риск, используя другие категории внедрения.

Кстати говоря, различные категории можно комбинировать друг с другом, осуществляя "гибридное" внедрение, наследующее худшие качества всех используемых механизмов, но и аккумулирующее их лучшие черты. Короче, делайте свой выбор, господа!

---

[1] как вариант, X-код может внедриться в хвост кластера, оккупируя один или несколько незанятых секторов (если они там есть), однако, здесь этот вариант не рассматривается, т.к. никакого отношения к PE-файлов он имеет.

категория A: внедрение в пустое место файла

Проще всего внедриться в пустое место файла. На сегодняшний день таких мест известно три: а) PE-заголовок; б) хвостовые части секций; в) регулярные последовательности. Рассмотрим их поподробнее.

внедрение в PE-заголовок

Типичный PE-заголовок вместе с MS-DOS заголовком и заглушкой занимает порядка ~300h байт, а минимальная кратность выравнивания секций составляет – 200h. Таким образом, между концом заголовка на началом первой секции практически всегда имеется ~100h бесхозных байт, которые можно использовать для "производственных целей", размещая здесь либо всю внедряемую программу целиком, либо только загрузчик X-кода, считывающий свое продолжение из дискового файла или реестра.

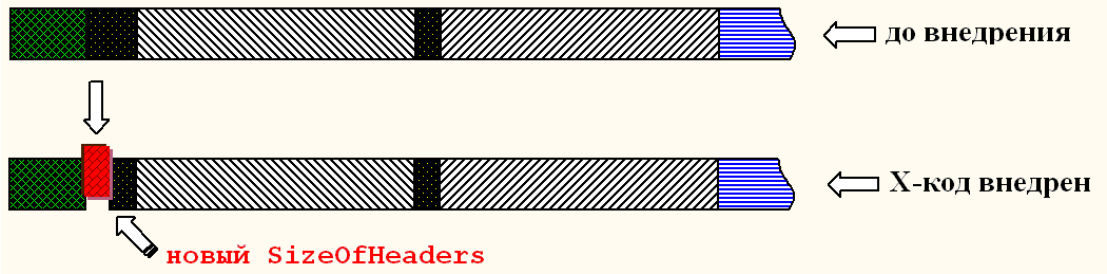

Рисунок 2 внедрение X-кода в свободное пространство хвоста PE-заголовка

Внедрение. Перед внедрением в заголовок X-код должен убедиться, что хвостовая часть заголовка (ласково называемая "предхвостием"), действительно свободна, т. е. SizeOfHeadres < FS.r_off. Если же SizeOfHeadres == FS.r_off вовсе не факт, что свободного места в конце заголовка нет. "Подтягивать" хвост заголовка к началу первой секции – обычная практика большинства линкеров, усматривающих в этом гармонию высшего смысла. Сканирование таких заголовков обычно выявляет длинную цепочку нулей, расположенных в его хвосте и, очевидно, никак и никем не используемых. Может ли X-код записать в них свое тело? Да, может, но только с предосторожностями. Необходимо отсчитать по меньшей мере 10h байт от последнего ненулевого символа, оставляя этот участок нетронутым (в конце некоторых структур присутствует до 10h нулей, искажение которых ни к чему хорошему не приведет).

Некоторые программисты пытаются проникнуть в MS-DOS заголовок и заглушку. Действительно, загрузчик Windows NT реально использует всего лишь шесть байт: сигнатуру "MZ" и указатель e_lfanew. Остальные же его никак не интересует и могут быть использованы X-кодом. Разумеется, о последствиях запуска такого файла в голой MS-DOS, лучше не говорить, но… MS-DOS уже давно труп. Правда, некоторые вполне современные PE-загрузчики дотошно проверяют все поля MS-DOS заголовка (в особенности это касается win32-эмуляторов), поэтому без особой нужды лучше в них не лезть, а вот использовать для своих нужд MS-DOS заглушку – можно, пускай и не без ограничений. Достаточно многие системные загрузчики не способны транслировать виртуальные адреса, лежащие к западу от PE-заголовка, что препятствует размещению в MS-DOS-заголовке/заглушке служебных структур PE-файла. Даже и не пытайтесь внедрять сюда таблицу импорта или таблицу перемещаемых элементов! А вот тело X-кода внедрять можно.

Кстати говоря, при обработке файла популярным упаковщиком UPX, X-код, внедренный в PE-заголовок, не выживает, поскольку, UPX полностью перестраивает заголовок, выбрасывая оттуда все "ненужное" (MS DOS-заглушку он, к счастью, не трогает) Упаковщики ASPack и tElock ведут себя более корректно, сохраняя и MS DOS-заглушку, и оригинальный PE-заголовок, однако, X-код должен исходить из худшего варианта развития событий.

В общем случае, внедрение в заголовок осуществляется так:

- считываем PE-заголовок и приступаем к его анализу;
- если SizeOfHeaders < FS.r_off и (SizeOfHeaders + sizeof(X-code))< FS.r_off то:
  - увеличиваем SizeOfHeaders на sizeof(X-code) или же просто подтягиваем его к raw offset'y первой секции;
  - записываем X-код на образовавшееся место;
- иначе:

- сканируем PE-заголовок на предмет поиска непрерывной цепочки нулей, и, если таковая будет действительно найдена, внедряем свое тело начиная с 10h байта от ее начала;
- внедряем X-код в MS-DOS заглушку, не сохраняя ее старого содержимого;

- если внедрение прошло успешно, перехватываем управление на X-код;

Идентификация пораженных объектов. Внедрение в PE-заголовок в большинстве случаев можно распознать и визуально. Рассмотрим как выглядит в hex-редакторе типичный исполняемый файл (см. рис. 3): вслед за концом MS-DOS заголовка, обычно содержащим в себе строку "This program cannot be run in DOS mode" (или что-то подобное), расположена "PE" сигнатура, за которой следует немного мусора, щедро разбавленного нулями, и плавно перетекающего в таблицу секций, содержащую легко узнаваемые имена .text, .rsrc и .data (если файл упакован, названия секций скорей всего будут другими).

Иногда за таблицей секций присутствует таблица BOUND импорта с перечнем имен загружаемых динамических библиотек. Дальше, вплоть до начала первой секции, не должно быть ничего, кроме нулей, использующихся для выравнивания. (отождествить начало первой секции легко, hiew ставит в этом месте точку). Если же это не так, то исследуемый файл содержит X-код (см. рис. 4).

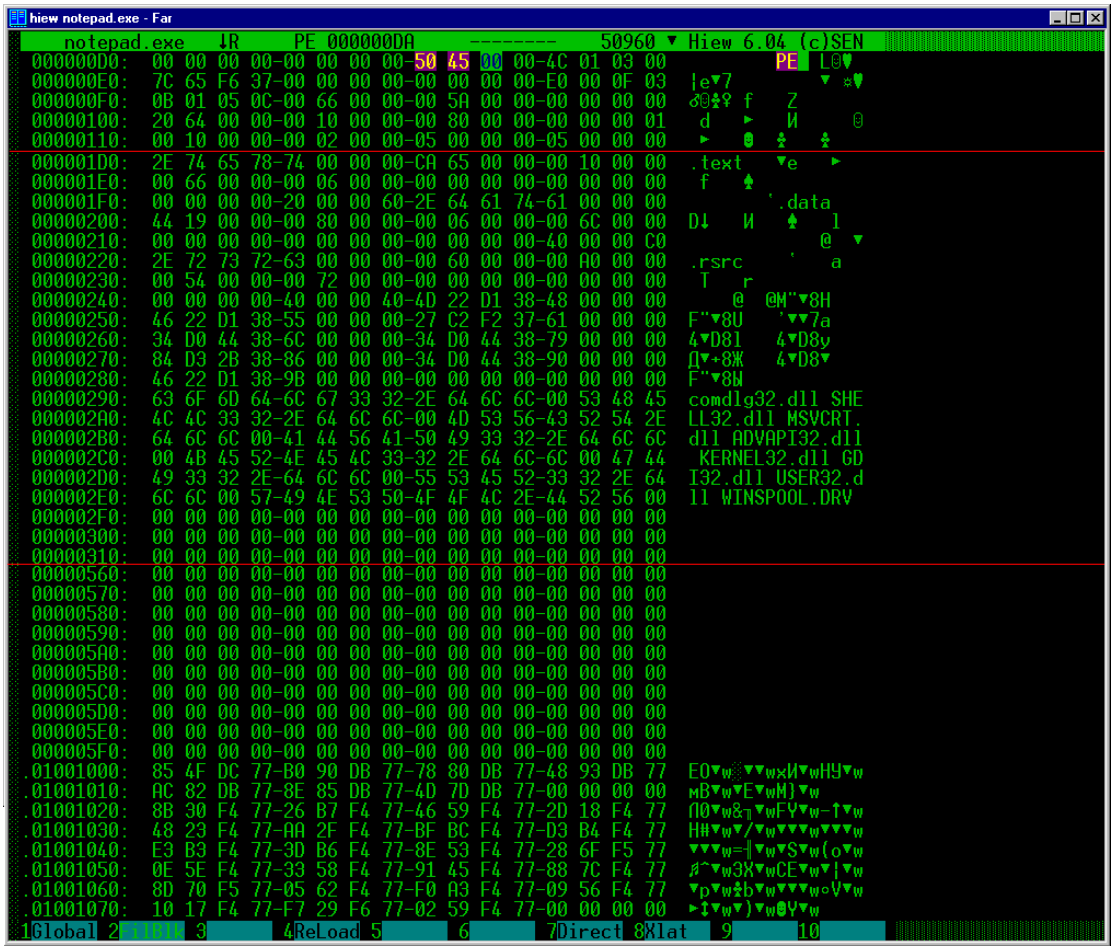

Рисунок 3 так выглядит типичный PE-заголовок незараженного файла

Рисунок 4 так выглядит заголовок файла после внедрения X-кода

Восстановление пораженных объектов. Не все дизассемблеры позволяют дизассемблировать PE-заголовок. IDA PRO относится к числу тех, что позволяют, но делает это только в случаях крайней необходимости, когда точка входа указывает внутрь заголовка. Заставить же ее отобразить заголовок вручную, судя по всему, невозможно. HIEW в этом отношении более покладист, но RVA-адреса и переходы внутри заголовка он не транслирует и их приходится вычислять самостоятельно. Дизассемблировав X-код и определив характер и стратегию перехвата управления, восстановите пораженный файл в исходный вид или потрассируйте X-код в отладчике, позволив ему сделать это самостоятельно, а в момент передачи управления оригинальной программе, сбросьте дамп (разумеется, прогон активного X-кода под отладчиком всегда таит в себе угрозу и отлаживаемая программа в любой момент может вырваться из-под контроля, поэтому если вы хотя бы чуточку не уверены в себе пользуйтесь дизассемблером, так будет безопаснее).

Если X-код оказался утрачен, например вследствие упаковки UPX'ом, распакуйте файл и постарайтесь идентифицировать стартовый код оригинальной программы (в этом вам поможет IDA PRO), переустановив на него точку входа. Возможно, вам придется реконструировать окрестности точки входа, разрушенные командой перехода на X-код. Если исходный стартовый код начинался с пролога (а в большинстве случаев это так), то на ремонт файла уйдет совсем немного времени (первые 5 байт пролога стандартны и легко предсказуемы, обычно это 55 8B EC 83 EC, 55 8B EC 83 C4, 55 8B EC 81 EC или 55 8B EC 81 C4, правильный вариант определяется по правдоподобности размера стекового фрейма, отводимого под локальные переменные). При более серьезных разрушениях, алгоритм восстановления становится неоднозначен и вам, возможно, придется перебрать большое количество вариантов. Попробуйте отождествить компилятор и изучить поставляемый вместе с ним стартовый код – это существенно упрощает задачу. Хуже, если X-код внедрился в произвольное место программы, предварительно сохранив оригинальное содержимое в заголовке (которого теперь с нами нет). Возвратить испорченный файл из небытия скорее всего будет невозможно, во всяком случае, никаких универсальных рецептов его реанимации не существует.

Некорректно внедренный X-код может затереть таблицу диапазонного импорта, обычно располагающуюся позади таблицы секций и тогда система откажет файлу в загрузке. Это происходит когда разработчик определяет актуальный конец заголовка по следующей формуле: e_lfanew + SizeOfOptionalHeader + 14h + NumberOfSections*40, которая, к сожалению, неверна.

Как уже говорилось выше, любой компилятор/линкер вправе использовать все SizeOfHeaders байт заголовка.

Если таблица диапазонного импорта дублирует стандартную таблицу импорта (а чаще всего это так), то простейший способ ремонта файла сводится к обнулению 0x11'го элемента DATA DIRECTORY, а точнее ссылки на структуру IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT. Если же таблица диапазонного импорта содержит (точнее содержала) уникальные динамические библиотеки, отсутствующие во всех остальных таблицах, то для восстановления достаточно знать базовый адрес их загрузки. При отключенном диапазонном импорте эффективные адреса импортируемых функций, жестко прописанные в программе, будут ссылаться на невыделенные страницы памяти и операционная система немедленно выбросит исключение, сообщая виртуальный адрес ячейки, к которой произошло обращение. Остается лишь найти динамическую библиотеку (и этой библиотекой скорее всего будет собственная библиотека восстанавливаемого приложения, входящая в комплект поставки), содержащую по данному адресу более или менее осмысленный код, совпадающий с точкой входа в функцию. Зная имена импортируемых библиотек, восстановить таблицу диапазонного импорта не составить никакого труда.

Для приличия (чтобы не ругались антивирусы) можно удалить неактивный X-код из файла, установив SizeOfHeaders на последний байт таблицы секций (или таблицы диапазонного импорта, если она есть) и вплоть до FS.r_off заполнив остальные байты нулями, символом "*" или любым другим символом по своему вкусу. Например, "посторонним вирусам вход воспрещен!".

```
HEADER:01000300 ; The code at 01000000-01000600 is hidden from normal disassembly
HEADER:01000300 ; and was loaded because the user ordered to load it explicitly
HEADER:01000300 ;
HEADER:01000300 ;<<<< IT MAY CONTAIN TROJAN HORSES, VIRUSES, AND DO HARMFUL THINGS >>>
HEADER:01000300 ;
HEADER:01000300         public start
HEADER:01000300 start:
HEADER:01000300         call    $+5
HEADER:01000305         pop     ebp
HEADER:01000306         mov     esi, fs:0
HEADER:0100030C         lodsd
HEADER:0100030D         push    ebp
HEADER:0100030E         lodsd
HEADER:0100030F         push    eax
```

Листинг 1 дизассемблерный фрагмент X-кода, внедренного в заголовок (все комментарии принадлежат Иде)

внедрение в хвост секции

Операционная система Windows 9x требует, чтобы физические адреса секций были выровнены по меньшей мере на 200h байт (Windows NT – на 002h), поэтому между секциями практически всегда есть некоторое количество свободного пространства, в котором легко затеряться.

Рассмотрим структуру файла notepad.exe из поставки Windows 2000 (см. листинг 2). Физический размер секции .text превышает виртуальный на 6600h – 65CAh == 36h байт, а .rsec – аж на C00h! Вполне достаточный объем пространства для внедрения, не правда ли? Разумеется, такое везение выпадает далеко не всегда, но пару десятков свободных байт можно найти практически в любом файле.

Number	Name	v_size	RVA	r_size	r_offst	flag
1	.text	00065CA	0001000	0006600	0000600	60000020
2	.data	0001944	0008000	0000600	0006C00	C0000040
3	.rsrc	0006000	000A000	0005400	0007200	40000040

Листинг 2 так выглядит таблица секций файла notepad.exe

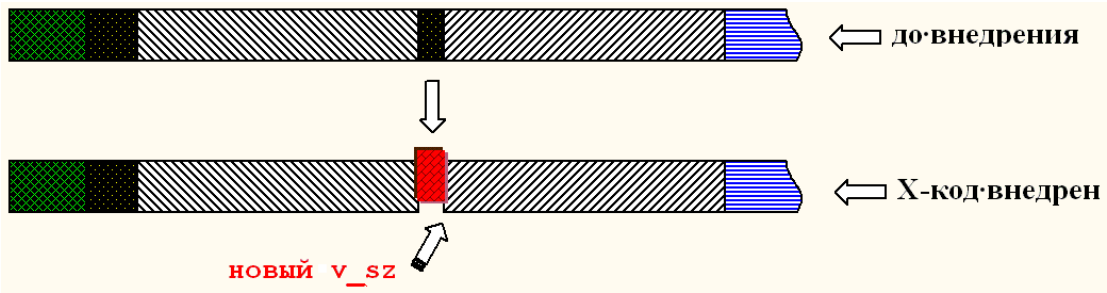

Рисунок 5 внедрение Х-кода в хвост секции, оставшийся от выравнивания

Внедрение. Перед внедрением необходимо найти секцию с подходящими атрибутами и достаточным свободным пространством в конце или рассредоточить Х-код в нескольких секциях. При этом необходимо учитывать, что виртуальный размер секции зачастую равен физическому или даже превышает его. Это еще не значит, что свободное пространство отсутствует, – попробуйте просканировать хвостовую часть секции на предмет наличия непрерывной цепочки нулей – если таковая там действительно присутствует (а куда бы она делась?), ее можно безбоязненно использовать для внедрения. Правда тут есть одно "но", почему-то не учитываемое подавляющее большинством разработчиков: если виртуальный размер секции меньше физического, загрузчик игнорирует физический размер (хотя и не обязан это делать) и он может быть любым, в том числе и заведомо бессмысленным! Если виртуальный размер равен нулю, загрузчик использует в качестве него физический, округляя его на величину Section Alignment. Поэтому, если r_off + r_sz некоторой секции превышает r_off следующей секции, следует либо отказаться от обработки такого файла, либо самостоятельно вычислить физический размер на основе разницы raw offset'ов двух соседних секций.

Некоторые программы хранят оверлеи внутри файла (да, именно внутри, а не в конце!), при этом разница физического и виртуального размеров как правило оказывается больше кратности физического выравнивания. Такую секцию лучше не трогать, т. к. внедрение Х-кода скорее всего приведет к неработоспособности файла. К сожалению, оверлеи меньшего размера данный алгоритм отловить не в состоянии, поэтому всегда проверяйте внедряемый участок на нули и отказывайтесь от внедрения, если здесь расположено что-то другое.

Большинство разработчиков Х-кода, проявляя преступную небрежность, пренебрегают проверкой атрибутов секции, что приводит к критических ошибкам и прочим серьезным проблемам. Внедряемая секция должна быть во-первых доступной (флаг IMAGE_SCN_MEM_READ установлен) и, во-вторых, невыгружаемой (флаг IMAGE_SCN_MEM_DISCARDABLE сброшен). Желательно, но необязательно, чтобы по крайней мере один флагов IMAGE_SCN_CNT_CODE, IMAGE_SCN_CNT_INITIALIZED_DATA был взведен. Если же эти условия не соблюдаются, и других подходящих секций нет, допустимо модифицировать флаги один или нескольких секций вручную, однако, работоспособность подопытного приложения в этом случае уже не гарантирована. Если флаги IMAGE_SCN_MEM_SHARED и IMAGE_SCN_MEM_WRITE установлены, в такую секцию может писать кто угодно и что угодно, а, во-вторых, адрес ее загрузки может очень сильно отличается от v_a, поскольку та же Windows 9x позволяет выделять разделяемую памяти только во второй половине адресного пространства.

Поскольку при внедрении в хвост секции, невозможно отличить данные, инициализированные нулями, от неинициализированных данных, перед передачей управления основному коду программы Х-код должен замести следы, аккуратно подчистив все за собой. Например, скопировать свое тело в стек или в буфер динамической памяти и вернуть нули на место. К сожалению, многие об этом забывают, в результате чего часть программ отказывает в работе.

Код, внедренный в конец секции, как правило выживает при упаковке или обработке файла протектором (т. к. внедренная область памяти теперь помечена как занятая). Исключение составляют служебные секции, такие, как секция перемещаемых элементов или секция импорта, сохранять которые упаковщик не обязан и вполне может реконструировать их, выбрасывая оттуда все "ненужное".

Обобщенный алгоритм внедрения выглядит приблизительно так:

- считываем PE-заголовок;
- анализируем Section Table, сравнивая физическую длину секций с виртуальной;

- ищем секции у которых r_sz > v_sz и записываем их в кандидаты на внедрение, предварительно убедившись, что в хвосте секции содержатся одни нули;
- если r_sz – v_sz >= FA не трогаем такую секцию, т. к. скорее всего она содержит оверлей;
- если кворума набрать не удалось, ищем секции у которых r_sz <= v_sz и пытаемся найти непрерывную цепочку нулей в их конце;
- из всех кандидатов отбираем секции с наибольшим количеством свободного места;
- находим секцию, атрибуты которой располагают к внедрению (IMAGE_SCN_MEM_SHARED, IMAGE_SCN_MEM_DISCARDABLE сброшены, IMAGE_SCN_MEM_READ или IMAGE_SCN_MEM_EXECUTE установлены, IMAGE_SCN_CNT_CODE или IMAGE_SCN_CNT_INITIALIZED_DATA установлены), а если таких среди оставшихся кандидатов нет, либо корректируем атрибуты самостоятельно, либо отказываемся от внедрения;
- если v_sz != 0 и v_sz < r_sz, увеличиваем v_sz на sizeof(X-code) или подтягиваем к v_a следующей секции;

Идентификация пораженных объектов. Распознать внедрения этого типа достаточно проблематично, особенно если X-код полностью помещается в первой кодой секции файла, которой как правило является секция .text.

Внедрение в секцию данных разоблачает себя наличием осмысленного дизассемблерного кода в ее хвосте, но если X-код перехватывает управление хитрым образом, дизассемблер может и не догадаться дизассемблировать этот код и нам придется сделать это вручную, самостоятельно отыскав точку входа. Правда, если X-зашифрован и расшифровщик находится вне кодовой секции, этот прием уже не сработает.

Внедрение во все служебные секции (например, секцию ресурсов или fixup'ов) распознается по наличию в них чужеродных элементов, которые не принадлежат никакой подструктуре данных.

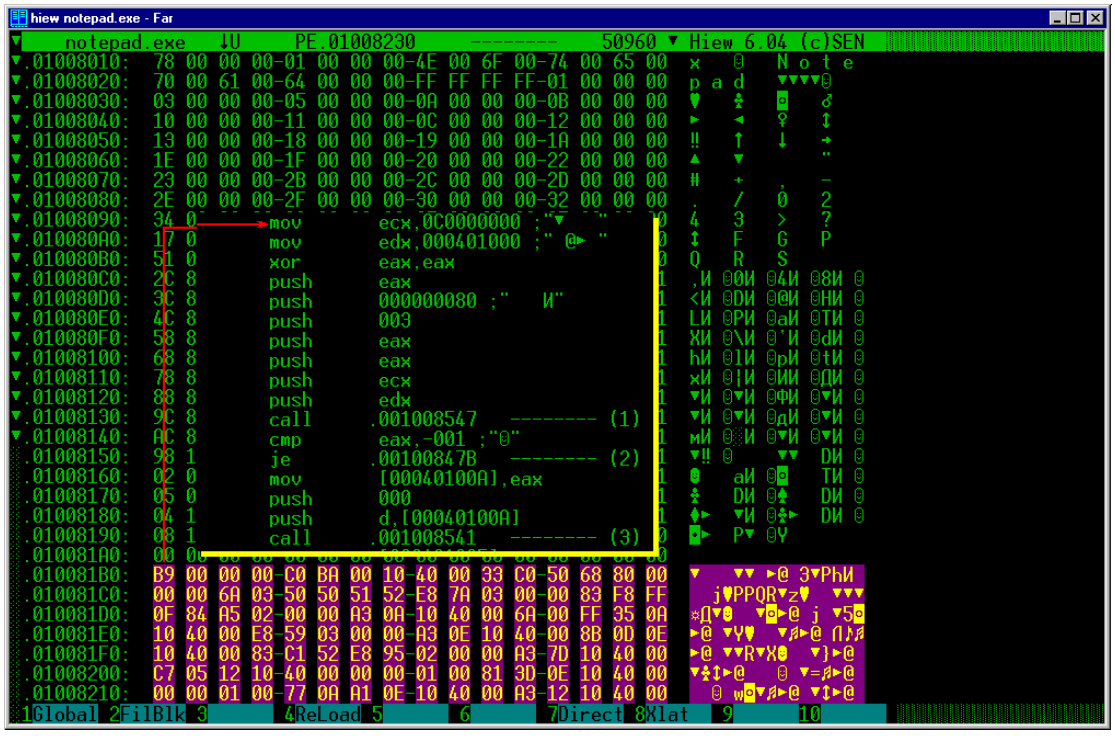

Рисунок 6 осмысленный машинный код в хвосте секции данных – признак внедрения

Восстановление пораженных объектов. Чаще всего приходится сталкивается с тем, что программист не предусмотрел специальной обработки для виртуального размера равного нулю и вместо того, чтобы внедриться в хвост секции, необратимо затер ее начало. Такие файлы восстановлению не подлежат и должны быть уничтожены. Реже встречается внедрение в секцию с "неудачными" атрибутами: секцию недоступную для чтения или DISCARDABLE-

секцию. Для реанимации файла, либо заберите у X-кода управление, либо отремонтируйте атрибуты секции.

Могут так же попасться файлы, с неправильно "подтянутым" виртуальным размером. Обычно вирусописатели устанавливают виртуальный размер внедряемой секции равным физическому, забывая о том, что если r_sz < v_sz, то виртуальный размер следует вычислять исходя из разницы адресов виртуальных адресов текущей и последующей секции. К счастью, ошибки внедрения этого типа не деструктивны и исправить виртуальный размер можно в любой момент.

внедрение в регулярную последовательность байт

Цепочки нулей необязательно искать в хвостах секций. Дался нам этом хвост, когда остальные части файла ничуть не хуже, а зачастую даже лучше конца! Скажем больше, необязательно искать именно нули, – для внедрения подходит любая регулярная последовательность (например, цепочка FF FF FF… или даже FF 00 FF 00…), которую мы сможем восстановить в исходный вид перед передачей управления. Если внедряемых цепочек большой одной, X-коду придется как бы "размазаться" по телу файла (а скорее всего так и будет). Соответственно, стартовые адреса и длины этих цепочек придется где-то хранить, иначе как потом прикажете их восстанавливать?

Регулярные последовательности чаще всего обнаруживаются в ресурсах, а точнее – в bitmap'ах и иконках. Технически внедриться сюда ничего не стоит, но пользователь тут же заметит искажение иконки, чего допускать ни в коем случае нельзя (даже если это и не главная иконка приложения, Проводник показывает остальные по нажатию кнопки "сменить значок" в меню свойств ярлыка). Существует и другая проблема: если регулярная последовательность относится к служебным структурам данным, анализируемых загрузчиком, то файл "упадает" еще до того, как X-код успеет восстановить эту регулярную последовательность в исходный вид. Соответственно, если регулярная последовательность содержит какое-то количество перемещаемых элементов или элементов таблицы импорта, то в исходный вид ее восстанавливать ни в коем случае нельзя, т. к. это нарушит работу загрузчика. Поэтому, поиск подходящей последовательности существенно усложняется, но отнюдь не становится принципиально невозможным!

Правда, некоторые программисты исподтишка внедряются в таблицу перемещаемых элементов, необратимо затирая ее содержимое, поскольку по их мнению исполняемым файлам она не нужна. Варвары! Хоть бы удостоверились сначала, что 01.00.00.00h >= Image Base >= 40.00.00h, в противном случае таблица перемещаемых элементов реально нужна файлу! К тому же не все файлы с расширением EXE – исполняемые. Под их личной вполне может прятаться и динамическая библиотека, а динамическим библиотекам без перемещения – никуда. Кстати говоря, вопреки распространенному мнению, установка атрибута IMAGE_FILE_RELOCS_STRIPPED вовсе не запрещает системе перемещать файл и для корректного отключения таблицы перемещаемых элементов необходимо обнулить поле IMAGE_DIRECTORY_ENTRY_BASERELOC в DATA DIRECTORY.

Автор знаком с парой лабораторных вирусов, умело интегрирующих X-код в оригинальную программу и активно использующих строительный материал, найденный в теле файла-хозяина. Основной интерес представляют библиотечные функции, распознанные по их сигнатуре (например, sprintf, rand), а если таковых не обнаруживается, X-код либо ограничивает свою функциональность, либо реализует их самостоятельно. В дело идут и одиночные машинные команды, такие как CALL EBX или JMP EAX. Смысл этого трюка заключается в том, что подобное перемешивание команд X-кода с командами основной программы, не позволяет антивирусам отодрать X-код от файла. Однако, данная техника еще не доведена до ума и все еще находится в стадии разработки…

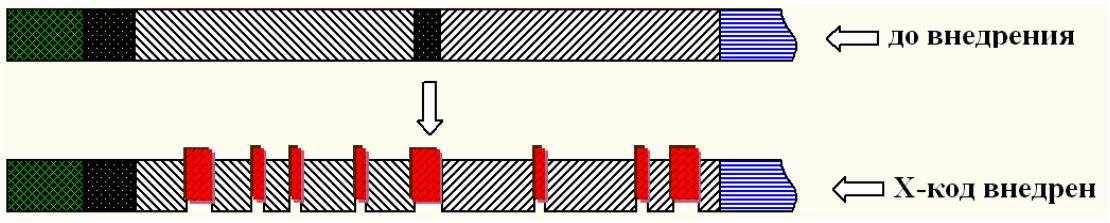

Рисунок 7 внедрение X-кода в регулярные цепочки

Внедрение. Алгоритм внедрения выглядит приблизительно так:

- сканируем файл на предмет поиска регулярных последовательностей и отбираем среди них цепочки наибольшей длины, причем сумма их длин должна несколько превышать размеры X-кода, т. к. на каждую цепочку в среднем приходится 11 байт служебных данных: четыре байта на стартовую позицию, один байт – на длину, один – на оригинальное содержимое и еще пять байт на машинную команду перехода к другой цепочке;
- убеждаемся, что никакая часть цепочки не принадлежит ни одной из подструктур, перечисленных в DATA DIRECTORY (именно ***под***структур, а не структур! поскольку таблицы экспорта/импорта, ресурсов, перемещаемых элементов образуют многоуровневые древовидные иерархии, произвольным образом рассеянные по файлу, ограничиться одной лишь проверкой к принадлежности, IMAGE_DATA_DIRECTORY.VirtualAddress и IMAGE_DATA_DIRECTORY.Size категорически недостаточно);
- проверяем атрибуты секции, которым принадлежит цепочка (IMAGE_SCN_MEM_SHARED, IMAGE_SCN_MEM_DISCARDABLE сброшены, IMAGE_SCN_MEM_READ или IMAGE_SCN_MEM_EXECUTE установлены, IMAGE_SCN_CNT_CODE или IMAGE_SCN_CNT_INITIALIZED_DATA установлены);
- "нарезаем" X-код на дольки, добавляя в конец каждой из них команду перехода на начало следующей, не забывая о том, что тот jmp, который соответствует машинному коду EBh, работает с относительными адресами, и это те самые адреса, которые образуются после загрузки программы в память. С "сырыми" смещениями внутри файла они вправе не совпадать. Как правильно вычислить относительный адрес перехода? Определяем смещение команды перехода от физического начала секции, добавляем к нему пять байт (длина команды вместе с операндом). Полученную величину складываем в виртуальным адресом секции и кладем полученный результат в переменную a1. Затем определяем смещение следующей цепочки, отсчитываемое от начала той секции, к которой она принадлежит и складываем его с виртуальным адресом, записывая полученный результат в переменную a2. Разность a2 и a1 и представляет собой операнд инструкции jmp;
- запоминаем начальные адреса, длины и исходное содержимое всех цепочек в импровизированном хранилище, сооруженном либо внутри PE-заголовка, либо внутри одной из цепочек. Если этого не сделать, тогда X-код не сможет извлечь свое тело из файла-хозяина для внедрения во все последующие. Некоторые разработчики вместо команды jmp используют call, забрасывающий на вершину стека адрес возврата. Как нетрудно сообразить, совокупность адресов возврата представляет собой локализацию "хвостов" всех используемых цепочек, а адреса "голов" хранятся в операнде команды call! Извлекаем очередной адрес возврата, уменьшаем его на четыре и – относительный стартовый адрес следующей цепочки перед нами!

Идентификация пораженных объектов. Внедрение в регулярную последовательность достаточно легко распознать по длинной цепочке jmp'ов или call'ов, протянувшихся через одну или несколько секций файла и зачастую располагающихся в совсем не свойственных исполняемому коду местах, например, секции данных (см. листинг 3). А, если X-код внедриться внутрь иконки, она начинает характерно "шуметь" (см. рис.8). Хуже, если одна регулярная цепочка, расположенная в кодой секции, вмещает в себя весь X-код целиком – тогда для выявления внедренного кода приходится прибегать к его дизассемблированию и прочим хитроумным трюкам. К счастью, такие регулярные цепочки в живой природе практически не встречаются. Во всяком случае, проскандировав содержимое папок WINNT и Program Files я обнаружил лишь один такой файл, да и то деинсталлятор.

```
.0100A708: 9C               pushfd
.0100A709: 60               pushad
.0100A70A: E80B000000       call    .00100A71A   -------- (1)
.0100A70F: 64678B260000     mov     esp,fs:[00000]
.0100A715: 6467FF360000     push    d,fs:[00000]
.0100A71B: 646789260000     mov     fs:[00000],esp
.0100A721: E800000000       call    .00100A726   -------- (2)
.0100A726: 5D               pop     ebp
.0100A727: 83ED23           sub     ebp,023 ;"#"
```

```
.0100A72A: EB2B              jmps    .00100A757   -------- (3)
…
.0100A757: EB0E              jmps    .00100A767   -------- (1)
…
.0100A767: 8BC5              mov     eax,ebp
.0100A769: EB2C              jmps    .00100A797   -------- (1)
…
.0100A797: EB5E              jmps    .00100A7F7   -------- (1)
…
.0100A7F7: EB5E              jmps    .00100A857   -------- (1)
…
.0100A857: EB3E              jmps    .00100A897   -------- (1)
…
.0100A897: EB3D              jmps    .00100A8D6   -------- (1)
…
.0100A8D6: EB0D              jmps    .00100A8E5   -------- (1)
…
.0100A8E5: 2D00200000        sub     eax,000002000 ;"    "
.0100A8EA: 89857E070000      mov     [ebp][00000077E],eax
.0100A8F0: 50                push    eax
.0100A8F1: 0500100000        add     eax,000001000 ;"  ► "
.0100A8F6: 89857E070000      mov     [ebp][00000077E],eax
.0100A8FC: 50                push    eax
.0100A8FD: 0500100000        add     eax,000001000 ;"  ► "
.0100A902: EB31              jmps    .00100A935   -------- (1)
```

Листинг 3 внедрение X-кода в регулярные цепочки

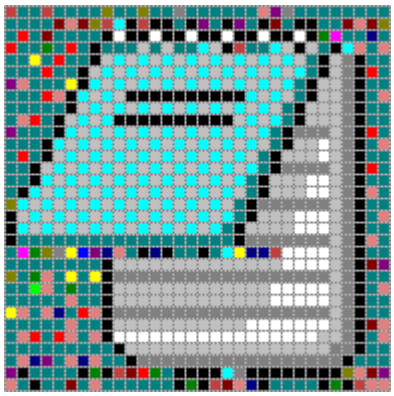

Рисунок 8 внедрение X-кода в главную иконку файла

Восстановление пораженных объектов. Отодрать от файла X-код, наглухо слившийся с ним интеграции алхимического брака (редактору – pardon, но по другому тут не скажешь), практически невозможно, поскольку отличить фрагменты X-кода от фрагментов оригинального файла практически нереально. Да и нужно ли? Ведь достаточно отобрать у него управление… К счастью, таких изощренных X-кодов в дикой природе практически не встречается и обычно они ограниваются внедрением в свободные с их точки зрения регулярные последовательности, которые вполне могут принадлежать буферам инициализированных данных и если X-код перед передачей управления оригинальной программе не подчистит их за собой, ее поведение рискует стать совершенно непредсказуемым (она ожидала увидеть в инициализированной переменной ноль, а ей что подсунули?).

Восстановление иконок и bitmap'ов не представляет большой проблемы и осуществляется тривиальной правкой ресурсов в любом приличном редакторе (например, в Visual Studio). Задачу существенно упрощает тот факт, что все иконки обычно хранятся в нескольких экземплярах, выполненных с различной цветовой палитрой и разрешением. К тому же, из всех регулярных последовательностей программисты обычно выбирают для внедрения нули, соответствующие прозрачному цвету в иконках и черному в bitmap'ах. Сама картинка остается неповрежденной, но окруженной мусором, который легко удаляется ластиком. Если после удаления X-кода файл отказывается запускаться, просто смените редактор ресурсов, либо воспользуйтесь hiew'ов, при минимальных навыках работы с котором иконки можно править и hex-режиме (считайте, что идете по стопам героев "Матрицы", рассматривающих окружающий мир через призму шестнадцатеричных кодов).

Отдельный случай представляет восстановление таблицы перемещаемых элементов, необратимо разрушенных внедренным X-кодов. Если Image Base < 40.00.00h, такой файл не

может быть загружен под Windows 9x, если в нем нет перемещаемых элементов. Причем, поле IMAGE_DIRECTORY_ENTRY_BASERELOC имеет приоритет над флагом IMAGE_FILE_RELOCS_STRIPPED и, если IMAGE_DIRECTORY_ENTRY_BASERELOC != 0, а таблица перемещаемых элементов содержит мусор, то попытка перемещения файла приведет к непредсказуемым последствиям – от зависания до отказа в загрузке. Если это возможно, перенесите поврежденный файл на Windows NT, минимальный базовый адрес загрузки которой составляет 1.00.00h, что позволяет ей обходится без перемещений даже там, где Windows 9x уже не справляется.

X-код, не проверяющий флага IMAGE_FILE_DLL может внедриться и в динамические библиотеки, имеющие расширение EXE. Вот это действительно проблема! В отличии от исполняемого файла, всегда загружающегося первым, динамическая библиотека вынуждена подстраивается под конкретную среду самостоятельно и без перемещаемых элементов ей приходится очень туго, поскольку ну один и тот же адрес могут претендовать множество библиотек. Если разрешить конфликт тасованием библиотек в памяти не удастся (это можно сделать утилитой EDITBIN из SDK, запущенной с ключом /REBASE), придется восстанавливать перемещаемые элементы вручную. Для быстрого отождествления всех абсолютных адресов можно использовать следующий алгоритм: проецируем файл в память, извлекаем двойное слово, присваиваем ее переменной X. Нет, X не годится, возьмем Y. Если Y >= Image Base и Y <= (Image Base + Image Size), объявляем текущий адрес кандидатом в перемещаемые элементы. Смещаемся на байт, извлекаем следующее двойное слово и продолжаем действовать в том же духе, пока не достигнем конца образа. Теперь загружаем исследуемый файл в ИДУ и анализируем каждого кандидата на "правдоподобность" – он должен представлять собой смещение, а не константу (отличие констант от смещений подробно рассматривалось в "Фундаментальных основах хакерства" Криса Касперски). Остается лишь сформировать таблицу перемещаемых элементов и записать ее в файл. К сожалению, предлагаемый алгоритм чрезвычайно трудоемок и не слишком надежен, т. к. смещение легко ступать с константой. Но других путей, увы, не существует. Останется надеяться лишь на то, что X-код окажется мал и затрет не всю таблицу, а только ее часть.

категория A: внедрение путем сжатия части файла

Внедрение в регулярные последовательности фактически является разновидностью более общей техники внедрением в файл путем сжатия его части, в данном случае осуществляемое по алгоритму RLE. Если же использовать более совершенные алгоритмы (например, Хаффмана или Лемпеля-Зива), то стратегия выбора подходящих частей значительно упрощается. Давайте сожмем кодовую секцию, а на освободившееся место запишем свое тело. Легко в реализации, надежно в эксплуатации! Исключение составляют, пожалуй, одни лиши упакованные файлы, которые уже не ужмешь, хотя… много ли X-коду нужно пространства? А секция кода упакованного файла по любому должна содержать упаковщик, хорошо поддающийся сжатию. Собственно говоря, разрабатывать свой компрессор совершенно необязательно, т. к. соответствующий функционал реализован и в самой ОС (популярная библиотека lz32.dll для наших целей непригодна, поскольку работает исключительно на распаковку, однако, в распоряжении X-кода имеются и другие упаковщики: аудио-/видео-кодеки, экспортеры графических форматов, сетевые функции сжатия и т. д.).

Естественно, упаковка оригинального содержимого секции (или ее части) не обходится без проблем. Во-первых, следует убедиться, что секция вообще поддается сжатию. Во-вторых, предотвратить сжатие ресурсов, таблиц экспорта/импорта и другой служебной информации, которая может присутствовать в любой подходящей секции файла и кодой секции в том числе. В-третьих, перестроить таблицу перемещаемых элементов (если, конечно, она вообще есть), исключая из нее элементы, принадлежащие сжимаемой секции и поручая настройку перемещаемых адресов непосредственно самому X-коду.

Возникают проблемы и при распаковке. Она должна осуществляться на предельной скорости, иначе время загрузки файла значительно возрастет и пользователь тут же почует что-то неладное. Поэтому, обычно сжимают не всю секцию целиком, а только ее часть, выбрав места с наибольшей степенью сжатия. Страницы кодовой секции от записи защищены и попытка их непосредственной модификации вызывает исключение. Можно, конечно, при внедрении X-кода присвоить кодовой секции атрибут IMAGE_SCN_MEM_WRITE, но красивым это решение никак не назовешь оно демаскирует X-код и снижает надежность программы. Это все равно, что сорвать с котла аварийный клапан – так и до взрыва недалеко. Лучше (и

правильнее!) динамически присвоить атрибут PAGE_READWRITE вызовом VirtualProtect, а после завершения распаковки возвратить атрибуты на место.

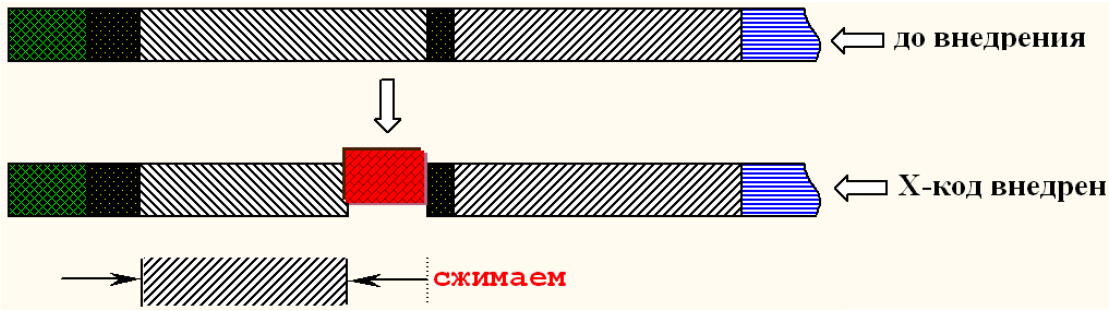

Рисунок 9 внедрение X-кода путем сжатия секции

Внедрение: Обобщенный алгоритм внедрения выглядит так:

- открываем файл, считываем PE-заголовок;
- находим в таблице секций секцию с атрибутом IMAGE_SCN_CNT_CODE (как правило, это первая секция файла);
- убеждаемся, что эта секция пригодна для внедрения (она сжимается, не содержит в себе ни никаких служебных таблиц, используемых загрузчиком, и не имеет атрибута IMAGE_SCN_MEM_DISCARDABLE);
- сжимаем секцию и записываем себя на освободившееся пространство, размещая X-код либо вначале секции, либо в ее конце;
- анализируем таблицу перемещаемых элементов и "выкусываем" оттуда все элементы, относящиеся к сжатой части секции и размещаем их внутри X-кода, а на выкушенные места записываем IMAGE_REL_BASED_ABSOLUTE – своеобразный аналог команды NOP для перемещаемых элементов;

Идентификация пораженных объектов. Распознать факт внедрения в файл путем сжатия части секции трудно, но все-таки возможно. Дизассемблирование сжатой секции обнаруживает некоторое количество бессмысленного мусора, настораживающего опытного исследователя, но зачастую ускользающего от новичка. Разумеется, речь не идет о внедрении в секцию данных – присутствие постороженного кода в которой не заметит только слепой (однако, если X-код перехватывает управление косвенным образом он не будет дизассемблированной ИДОЙ и может прикинуться овечкой невинного массива данных).

Обратите внимание на раскладку страничного имиджа. Если виртуальные размеры большинства секций много больше физический, файл по всей видимости сжат каким либо упаковщиков. В несколько меньшей степени это характерно для протекторов, вирусы же практически никогда не уменьшают физического размера секций т. к. для этого им пришлось бы перестраивать всю структуру заражаемого файла целиком, что не входит в их планы.

Восстановление пораженных объектов. Типичная ошибка большинства разработчиков – отсутствие проверки на принадлежность сжимаемой секции служебным структурам (или некорректно выполненная проверка). В большинстве случаев ситуация обратима, достаточно обнулив все поля DATA DIRECORY загрузить файл в дизассемблер, реконструировать алгоритм распаковщика и написать свой собственный, реализованный на любом симпатичном вам языке (например, ИДА-Си, тогда для восстановления файла даже не придется выходить из ИДЫ).

Если же файл запускается вполне нормально то для удаления X-кода достаточно немного потрассировать его в отладчике, дождавшись момента передачи управления оригинальной программе и немедленно сбросить дамп.

категория A: создание нового NTFS-потока внутри файла

Файловая система NTFS поддерживает множество потоков в рамках одного файла, иначе называемых **расширенными атрибутами (Extended Attributes)** или именованными разделами. Безымянный атрибут соответствует основному телу файла, атрибут $DATE – времени создания файла и т. д. Вы так же можете создавать и свои атрибуты практически

неограниченной длинны (точнее, до 64 Кбайт), размещая в них всякую всячину (например, X-код). Аналогичную технику использует и Mac OS, только там потоки именуются трудно переводимым словом forks. Подробнее об этом можно прочитать в "Основах Windows NT и NTFS" Хелен Кастер, "Недокументированных возможностях Windows NT" А.В.Коберниченко и "Windows NT File System Internals" Rajeev'a Nagar'a.

Сильной стороной этого алгоритма является высочайшая степень его скрытности, т. к. видимый объем файла при этом не увеличивается (под размером файла система понимает отнюдь не занимаемое им пространство, а размер основного потока), однако, список достоинства на этом и заканчивается. Теперь поговорим о недостатках. При перемещении файла на не NTFS-раздел (например, дискету, zip или CD-R/RW) все рукотворные потоки бесследно исчезают. Тоже самое происходит при копировании файла из оболочки наподобие Total Commander'a (в девичестве Windows Commander'a) или обработке архиватором. К тому же, полноценная поддержка NTFS есть только в Windows NT.

Внедрение. Ввиду хрупкости расширенных атрибутов, X-код необходимо проектировать так, чтобы пораженная программа сохраняла свою работоспособность даже при утрате всех дополнительных потоков. Для этого в свободное место подопытной программы (например, в PE-заголовок) внедряют крошечный загрузчик, который считывает свое продолжение из NTFS-потока, а если его там не окажется, передает управление программе-носителю.

Функции работы с потоками недокументированны и доступы только через Native-API. Это: NtCreateFile, NtQueryEaFile и NtSetEaFile, описание которых можно найти в частности в книге "The Undocumented Functions Microsoft Windows NT/2000" Tomasz'a Nowak'a, электронная копия которой может быть бесплатно скачена с сервера NTinterlnals.net.

Создания нового потока осуществляется вызовом функции NtCreateFile, среди прочих аргументов принимающей указатель на структуру FILE_FULL_EA_INFORMATION, передаваемый через EaBuffer. Вот она-то нам и нужна! Как вариант, можно воспользоваться функцией NtSetEaFile, передав ей дескриптор, возращенный NtCreateFile, открывающей файл обычным образом. Перечислением (и чтением) всех имеющихся потоков занимается функция NtQueryEaFile. Прототипы всех функций и определения структур содержатся в файле NTDDK.H, в котором присутствует достаточное количество комментариев, чтобы со всем этим хозяйством разобраться, однако, до тех пор пока Windows 9x не будет полностью вытеснена с рынка, подобная техника внедрения судя по всему останется невостребованной.

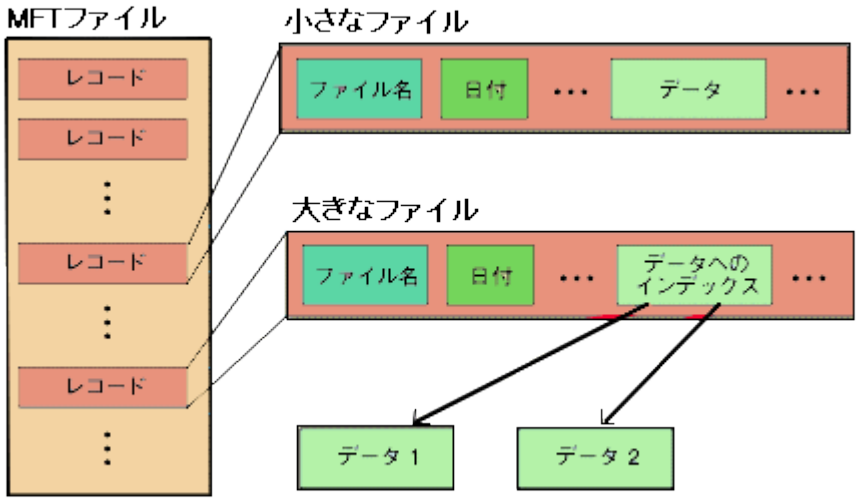

Рисунок 10 файловая система NTFS поддерживает несколько потоков в рамках одного файла (рисунок, к сожалению, на китайском – другой найти не удалось, но приблизительная структура понятна и без перевода)

Идентификация пораженных объектов. По непонятным маркетинговым соображениям штатные средства Windows не позволяют просматривать расширенные атрибуты файлов, мне так же неизвестна ни одна утилита сторонних производителей, способная справится с этой задачей, поэтому необходимый минимум программного обеспечения приходится разрабатывать самостоятельно. Наличие посторонних потоков внутри файла однозначно свидетельствует о его зараженности.

Другой, не менее красноречивый признак внедрения, обращение к функциям NtQueryEaFile/NtSetEaFile, которое может осуществляется как непосредственным импортом из NTDLL.DLL, так и прямым вызовом INT 2Eh.EAX=067h/INT 2Eh.EAX = 9Ch, а в Windows XP еще и машинной командой syscall. Возможет так же вызов по прямым адресам NTDLL.DLL или динамический поиск экспортируемых функций в памяти.

Восстановление пораженных объектов. Если после обработки упаковщиком/архиватором или иных внешне безобидных действий, файл неожиданно отказал в работе, одним из возможных объяснений является гибель расширенных атрибутов. При условии, что потоки не использовались для хранения оригинального содержимого файла, у нас неплохие шансы на восстановление. Просто загрузите файл в дизассемблер и, проанализировав работу X-кода, примите необходимые меры противодействия. Более точной рекомендации дать увы не получается, поскольку, такая тактика внедрения существует лишь теоретически и своего боевого крещения еще не получила.

Для удаления ненужных потоков можно воспользоваться уже описанной функцией NtSetEaFile.

категория B: раздвижка заголовка

Никакой, уважающий себя X-код не захочет зависеть от наличия свободного места в подопытном файле, поскольку это унизительно и вообще не по-хакерски.

Когда пространства, имеющегося в PE-заголовке (или какой либо другой части файла) оказывается недостаточно для размещения всего X-кода целиком, мы можем попробовать растянуть заголовок на величину, выбранную по своему усмотрению. До тех пор, пока SizeOfHeaders не превышает физического смещения первой секции, такая операция осуществляется элементарно (см. "внедрение в PE-заголовок") но вот дальше начинаются проблемы, для решения которых приходится кардинально перестаивать структуру подопытного файла. Как минимум необходимо увеличить raw offset'ы всех секций на величину кратную принятой степени выравнивания прописанной в поле File Alignment и физически переместить хвост файла, записав X-код на освободившееся место.

Максимальный размер заголовка равен виртуальному адресу первой секции, что и неудивительно, т. к. заголовок не может перекрываться с содержимым страничного имиджа. Учитывая, что минимальный виртуальный адрес составляет 1000h, а типичный размер заголовка – 300h, мы получаем свое распоряжение порядка 3 Кбайт свободного пространства, достаточного для размещения практически любого X-кода. В крайнем случае можно поместить оставшуюся часть в оверлей. Хитрость заключается в том, что системный загрузчик загружает лишь первые SizeOfHeaders байт заголовка, а остальные (при условии, что они есть) оставляет болтаться в оверлее. Мы можем сдвинуть raw offset'ы всех секций хоть на мегабайт, внедрив мегабайт X-кода в заголовок, но в память будет загружено только SizeOfHeaders байт, а о загрузке остальных X-код должен позаботиться самостоятельно.

К сожалению, одной лишь коррекции raw offset'ов для сохранения файлу работоспособности может оказаться недостаточно, поскольку многие служебные структуры (например, таблица отладочной информации) привязывается к своему физическому местоположению, которое после раздвижки заголовка неизбежно отнесет в сторону. Правила этикета требуют либо скорректировать все ссылки на абсолютные физические адреса (а для этого мы должны знать формат всех корректируемых структур, среди которых есть полностью или частично недокументированные – взять хотя бы туже отладочную информацию), либо отказаться от внедрения, если один или несколько элементов таблицы DATA DIRECTIRY содержат нестандартные структуры (ресурсы, таблицы экспорта, импорта и перемещаемых элементов используют только виртуальную адресацию, поэтому ни в какой дополнительной корректировке не нуждаются). Следует так же убедиться и в отсутствии оверлеев, поскольку многие из них адресуются относительно начала файла. Проблема в том, что мы не можем надежно отличить настоящий оверлей от мусора, оставленного линкером в конце файла и потому приходится идти на неоправданно спекулятивные допущения, что все, что занимает

меньше одного сектора – не оверлей. Или же различимыми эвристическими методами пытаться идентифицировать мусор.

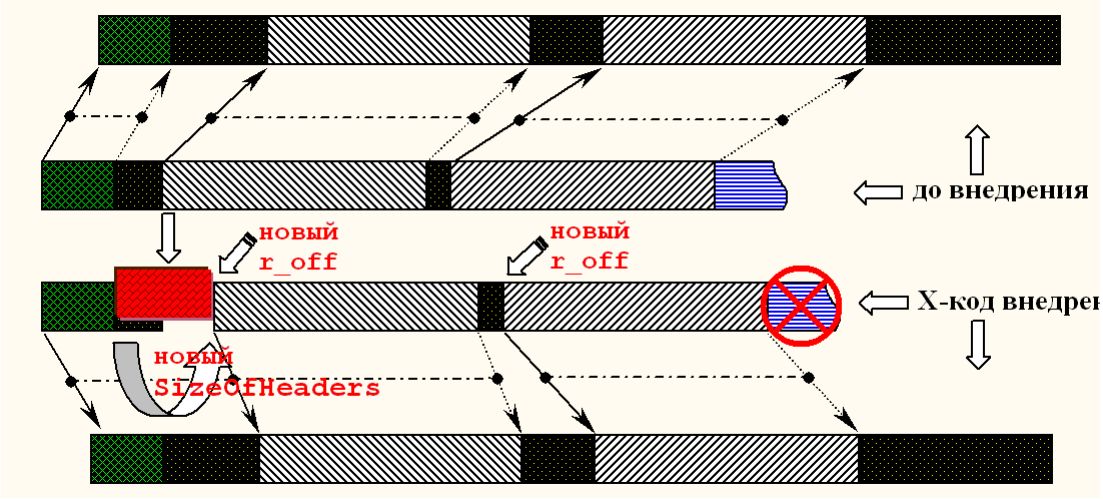

Рисунок 11 подопытный файл и его проекция в память до и после внедрения X-кода путем раздвижки заголовка

Внедрение. Типичный алгоритм внедрения выглядит так:

- считываем PE-заголовок;
- проверяем DATA DIRECTORY на предмет присутствия структур, привязывающихся к своему физическому смещению и, если таковые обнаруживаются, либо отказываемся от внедрения, либо готовимся их скорректировать;
- если SizeOfHeaders = FS.v_a, отказываемся от внедрения, т. к. внедряться уже некуда;
- если SizeOfHeaders != FS.r_off первой секции, подопытный файл содержит оверлей и после внедрения может оказаться неработоспособным, однако, если от SizeOfHeders до raw offset'a содержатся одни нули, внедряться сюда все-таки можно;
- если sizeof(X-code) <= FS.r_off, переходим к главе "внедрение в PE-заголовок";
- если sizeof(X-code) <= FS.v_a, то:
  - вставляем между концом заголовка на началом страничного имиджа ALIGN_UP((sizeof(X-code) + SizeOfHeaders – FS.r_off), FA) байт, физически перемещая хвост файла. При загрузке файла весь X-код будет спроецирован в память;
  - увеличиваем поле SizeOfHeaders на заданную величину;
- иначе:
  - вставляем между концом заголовка на началом страничного имиджа ALIGN_UP((sizeof(X-code) + SizeOfHeaders – FS.r_off), FA) байт, физически перемещая хвост файла; При загрузке файла, системный загрузчик спроецирует только первые FS.v_a – SizeOfHeaders байт X-кода, а все последующие ему придется считывать самостоятельно;
  - SizeOfHeaders := FS.v_a;
- увеличиваем raw offset'ы всех секций на величину физической раздвижки файла;
- корректируем все структуры, привязывающие к физическим смещениям внутри файла, перечисленные в DATA DIRECTORY;

Идентификация пораженных объектов. Данный метод внедрения распознается аналогично обычному методу внедрению в PE-заголовок (см. "внедрение в PE-заголовок") и по соображениям экономии места здесь не дублируется.

Восстановление пораженных объектов. При растяжке заголовка с последующим перемещением физического содержимого всех секций и оверлеев, вероятность нарушения работоспособности файла весьма велика, а причины ее следующие: некорректная коррекция raw offset'ов и привязка к физическим адресам. Ну против некорректной коррекции, вызванной

грубыми алгоритмическими ошибками, не попрешь и с испорченным файлом скорее всего придется расстаться (но все-таки попытайтесь, отталкиваясь от виртуальных размеров/адресов секций определить их физические адреса или идентифицируйте границы секций визуальными методами, благо они достаточно характерны, hex-редактор и холодное пиво вам в помощь!), а вот преодолеть привязку к физическим адресам можно! Проще всего это сделать, вернув содержимое секций/оверлеев на старое место, на их историческую родину там сказать. Последовательно сокращая размер заголовка на величину File Alignment и физически подтягивая секции на освободившиеся место, добейтесь его работоспособности. Ну а если не получится, значит, причина в чем-то еще…

категория B: сброс части секции в оверлей

Вместо того, чтобы полностью или частично сжимать секцию, можно ограничиться переносом ее содержимого в оверлей, расположенный в конце, середине или начале файла. Дописаться в конец файла проще всего. Никакие поля PE-заголовка при этом корректировать не надо – просто копируем sizeof(X-code) байт любой части секции в конец файла, а на освободившееся место внедряем X-код, который перед передачей управления программе-носителю считывает его с диска, возвращая на исходное место.

Сложнее разместить оверлей в средине файла, расположив его между секциями кода и данных, например, что обеспечит ему высокую степень скрытности. Для этого будет необходимо увеличить raw offset'ы всех последующие секций на величину ALIGN_UP(sizeof(X-code), FA), физически сдвинув секции внутри файла. Аналогичным образом осуществляется и создание оверлея в заголовке, о которым мы уже говорили (см. "раздвижка заголовка").

При обработке файла упаковщиков оверлеи (особенно серединные) обычно гибнут, но даже если и выживают, оказываются расположенными совершенно по другими физическими смещениям, поэтому, X-код при поиске оверлея ни в коем случае не должен жестко привязываться к его адресам, вычисляя их на основе физического адреса следующей секции. Пусть длина оверлея составляет OX байт, тогда его смещение будет равно: NS.r_off – OX, а для последнего оверлея файла: SizeOfFile – OX. Оверлеи в заголовках намного более выносливы, но при упаковке UPX'ом гибнут и они.

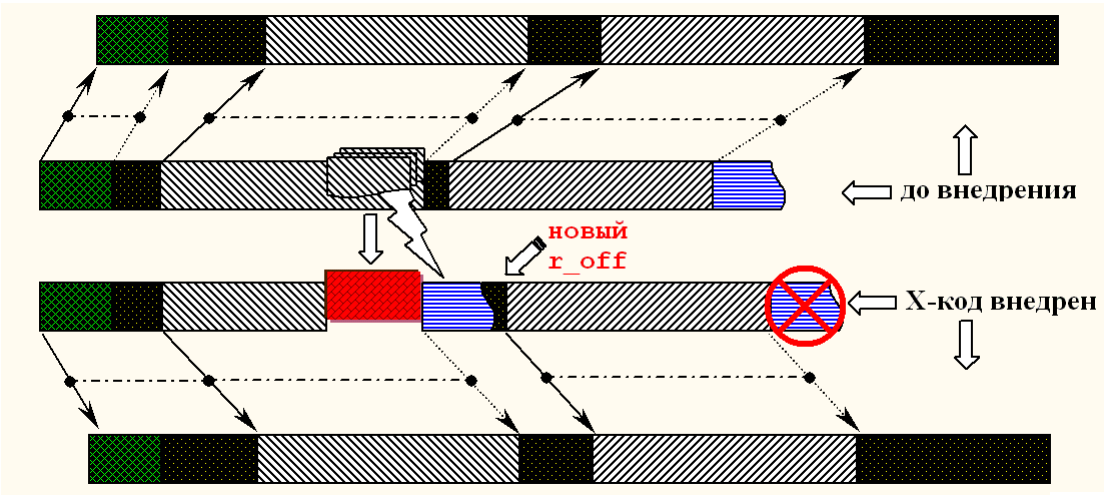

Рисунок 12 внедрение X-кода в файл путем сброса части секции в оверлей

Внедрение. Обобщенный алгоритм внедрения выглядит так:

- считываем PE-заголовок и приступаем к его анализу;
- если DATA DIRECORY содержит ссылку на структуру, привязывающуюся к физическим смещениями, либо готовимся скорректировать ее надлежащим образом, либо отказываемся от внедрения;
- если LS.r_off + LS.r_sz > SizeOfFile, файл скорее всего содержит оверлей и лучше отказаться от внедрения;

- если физический размер какой-либо секции превышает виртуальный на величину большую или равную File Alignment, файл скорее всего содержит серединный оверлей и настоятельно рекомендуется отказаться от внедрения;
- выбираем секцию, подходящую для внедрения (IMAGE_SCN_MEM_SHARED, IMAGE_SCN_MEM_DISCARDABLE сброшены, IMAGE_SCN_MEM_READ или IMAGE_SCN_MEM_EXECUTE установлены, IMAGE_SCN_CNT_CODE или IMAGE_SCN_CNT_INITIALIZED_DATA установлены); которой как правило является первая секция файла;
- физическое смещение начала секции в файле равно ее raw offset'у (это надежное поле и ему можно верить);
- физическое смещение конца секции в файле вычисляется более сложным образом: min(CS.raw offset + ALIGN_DOWN(CS.r_sz, FA), NS.raw_off);
- находим часть секции, не содержащую подструктур служебных таблиц PE-файла, таких, например, как таблицы импорта/экспорта;
- в выбранной части (частях) секции находим один или несколько регионов, свободных от перемещаемых элементов, а если это невозможно, "выкусываем" эти элементы из fixup table для последующей обработке X-кодом вручную;
- при желании находим первый пролог и последний эпилог внутри выбранных частей секции, чтобы линия "отреза" не разорвала функцию напополам (это не нарушит работоспособности файла, но сделает факт внедрения более заметным);
- если мы хотим создать оверлей внутри файла, то:
    - увеличиваем raw offset'ы всех последующих секций на величину ALIGN_UP(sizeof(X-code), FA);
    - физически сдвигаем все последующие секции в файле на эту же величину;
    - перемещаем выбранные части секции в оверлей, записывая их в произвольном формате, но так чтобы сами потом смогли разобраться;
- иначе:
    - дописываем выбранные части секции в конец файла, записывая их в произвольном формате, но так чтобы сами потом смогли разобраться;
- на освободившееся место записываем X-код;

Идентификация пораженных объектов. Дизассемблирование таких файлов не выявляет ничего необычного: X-код расположен в секции кода, там где и положено всякому нормальному коду быть. Никакого подозрительного мусора так же не наблюдается. Правда, обнаруживается некоторое количество перекрестных ссылок, ведущих в середину функций (и эти функции, как нетрудно догадаться принадлежат X-коду, даже если он и обрежет выдираемые фрагменты секций по границам функций, смещения функций X-кода внутри каждого из фрагментов будут отличается от оригинальных, вырезать каждую функцию по отдельности не предлагать – это слишком нудно), однако, такое нередко случается и с заведомо незараженными файлами, поэтому оснований для возбуждения уголовного дела как будто бы нет. Присутствие серединного оверлея легко распознать по несоответствию физических и виртуальный адресов, чего не наблюдается практически ни у одного нормального файла, однако, наличие оверлея в конце файла – это нормально.

Ничего другого не остается, как анализировать весь X-код целиком и если манипуляции с восстановлением секции будут обнаружены – факт внедрения окажется разоблачен. X-код выдает себя вызовом функций VirtualProtect (присвоение атрибута записи) и GetCommandLine, GetModuleBaseName, GetModuleFullName или GetModuleFullNameEx (определение имени файла-носителя). Убедитесь так же, что кодовая секция доступна только лишь на чтение, в противном случае шансы на присутствие X-кода существенно возрастут (и ему уже будет не нужно вызвать VirtualProtect).

Восстановление пораженных объектов. Обычно приходится сталкиваться с двумя алгоритмическими ошибками, допущенными разработчиками внедряемого кода: корректной проверкой на пересечение сбрасываемой части секции со служебными данными и внедрение в секцию с неподходящими атрибутами. Обе полностью обратимы.

Реже встречается ошибки определения длины сбрасываемой секции: если CS.v_sz < CS.r_sz и CS.r_off + CS.raw_sz > NS.raw_off, то системный загрузчик загружает лишь CS.v_sz байт секции, а внедряемый код сбрасывает CS.r_sz байт секции, захватывая кусочек следующей секции, не учитывая, что она может проецироваться совершенно по другим адресам и при восстановлении оригинального содержимого сбрасываемой секции, кусочек следует

секции так и не будет восстановлен. Хуже того, X-код окажется как бы разорван двумя секциями напополам и эти половинки могут находится как угодно далеко друг от друга! Естественно, работать после этого он не сможет.

Если же пораженный файл запускается нормально, для удаления X-кода просто немного потрассируйте его и, дождавшись момента передачи управления основной программе, снимите дамп.

категория B: создание своего собственного оверлея

Оверлей может хранить не только оригинальное содержимое секции, но и X-код! Правда, полностью вынести весь X-код в оверлей не удастся – как ни крути, но хотя бы крохотный загрузчик в основное тело все-таки придется внедрить, расположив его в заголовке, предсхостии, регулярной последовательности или других свободных частях файла.

Достоинства этого механизма в простоте его реализации, надежности, не конфликтности и т. д. Теперь уже не требуется анализировать служебные подструктуры на предмет их пересечения со сбрасываемой частью секции (мы ничего не сбрасываем!) и если случиться так, что оверлей погибнет, загрузчик просто передаст управления основной программе без нарушения ее работоспособности.

Располагать оверлей следует либо в конце файла, либо в его заголовке. Хоть это и будет более заметно, шансы выжить при упаковке у него значительно возрастут.

Внедрение. Алгоритм внедрения полностью идентичен предыдущему за тем лишь исключением, что в оверлей сбрасывается не часть секции файла-хозяина, а непосредственно сам X-код, обрабатываемый специальным загрузчиком. Внедрение загрузчика обычно осуществляется по категории A (см. "внедрение в пустое место файла"), хотя в принципе можно использовать и другие категории.

Идентификация пораженных объектов. Внедрения этого типа легко распознаются визуально по наличию загрузчика, как правило внедренного по категории A и присутствию сверления в начале, конце или середине файла.

Восстановление пораженных объектов. Если X-код спроектирован корректно, для его удаления достаточно убить оверлей (например, упаковав программу ASPack'ом со сброшенной галочкой "сохранять оверлеи"). Методика удаления загрузчика, внедренного по категории A, уже была описана выше, так что не будем повторяться.

категория C: расширение последней секции файла

Идея расширения последней секции файла не нова и своими корнями уходит глубоко в историю, возвращая нас во времена господства операционной системы MS-DOS и файлов типа OLD-EXE (помните, историю с фальшивыми монетами, на которых было отчеканено 2000 г. д. н. э.? древние не знали, что они живут до нашей эры! OLD-EXE тогда еще не были OLD).

Это наиболее очевидный и наиболее популярный алгоритм из всех алгоритмов внедрения вообще (часто даже называемый "стандартным способом внедрения"), однако его тактико-технические характеристики оставляют желать лучшего: он чрезвычайно конфликтен, слишком заметен и реально применим лишь к некоторым PE-файлам, отвечающим всем, предъявляемым к ним требованиям (зато он хорошо безболезненно переносит упаковку и обработку протекторами).

На первый взгляд идея не встречает никаких препятствий: дописываем X-код в хвост последней секции, увеличиваем размер страничного имиджа на соответствующую величину, не забывая о ее выравнивании и передаем на X-код управление. Никаких дополнительных передвижений одних секций относительно других осуществлять не нужно, а, значит, не нужно корректировать и их raw offset'ы. Проблема конфликтов со служебными структурами PE-файла также отпадает и нам нечего опасаться, что X-код перезапишет данные, принадлежащие таблице импорта или, например, ресурсам.

Но стоит только снять розовые очки и заглянуть в глаза реальности, как проблемы попрут изо всех щелей. А что если конец последней секции не совпадает с концом файла? Может же там оказаться оверлей или просто мусор, оставленный линкером? А что если последней секций файла является секция неинициализированных данных или DISCARDABLE-секция, которая в любой момент может быть выгружена из файла?

Внедряться в последнюю секцию файла не только технически неправильно, но и политически некорректно. Тем не менее и этот способ имеет право на существование, поэтому, рассмотрим его поподробнее.

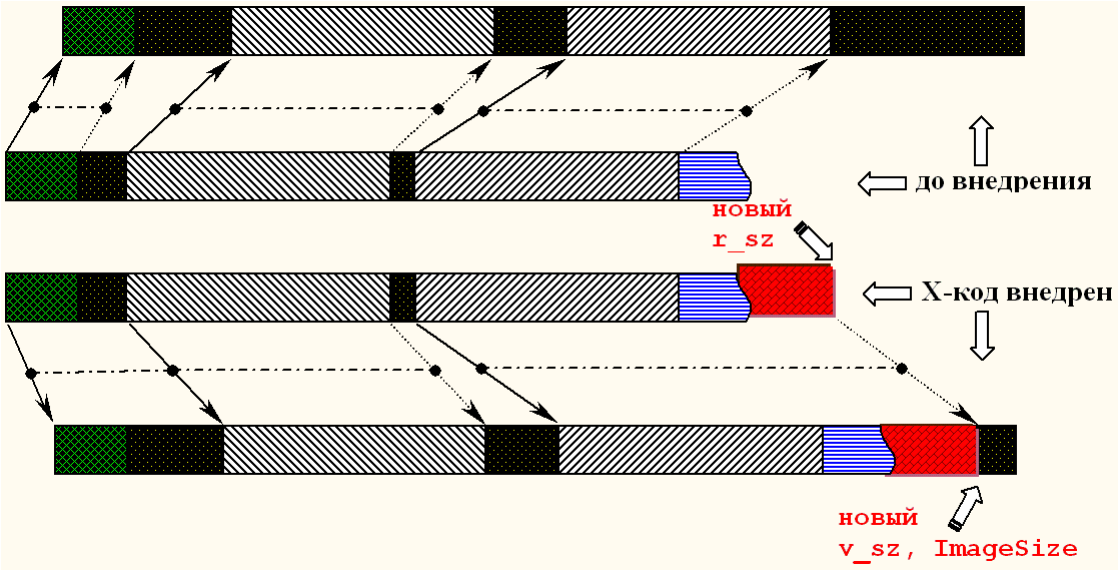

Рисунок 13 внедрение X-кода в файл путем расширения последней секции

Внедрение. Если физический размер последней секции, будучи выровненным на величину File Alignment, не "дотягивается" до физического конца файла, значит, X-код должен либо отказаться от внедрения, либо пристегивать свое тело не к концу секции, а к концу файла. Разница непринципиальна за исключением того, что оверлей теперь придется загружать в память, увеличивая как время загрузки, так и количество потребляемых ресурсов. Внедряться же между концом секции и началом оверлея категорически недопустимо, т. к. оверлеи чаще всего адресуются относительно начала файла (хотя могут адресоваться и относительно конца последней секции). Другая тонкость связана с пересчетом виртуального размера секции – если он больше физического (как чаще всего и случается), то он уже включает в себя какую-то часть оверлея, поэтому алгоритм вычисления нового размера существенно усложняется.

С атрибутами секций дела обстоят еще хуже. Секции неинициализированных данных вообще не обязаны загружаться с диска (хотя 9x/NT их все-таки загружают), а служебные секции (например, секция перемещаемых элементов) реально востребованы системой лишь на этапе загрузки PE-файла, активны только на стадии загрузки, а дальнейшее их пребывание в памяти негарантированно и X-код запросто может схлопотать исключение еще до того, как успеет передать управление основной программе. Конечно, X-код может скорректировать атрибуты последней секции по своему усмотрению, но это ухудшит производительность системы и будет слишком заметно. Если физический размер последней секции равен нулю, что характерно для секций не инициализированных данных, лучше ее пропустить, внедрившись в предпоследнюю секцию.

Типичный алгоритм внедрения выглядит так:

- загружаем PE-заголовок и анализируем атрибуты последней секции;
- если флаг IMAGE_SCN_MEM_SHARED установлен, отказываемся от внедрения;
- если флаг IMAGE_SCN_MEM_DISCARDABLE установлен, либо отказываемся от внедрения, либо самостоятельно сбрасываем его;
- если флаг IMAGE_SCN_CNT_UNINITIALIZED_DATA установлен лучше всего отказаться от внедрения;
- если ALIGN_UP(LS.r_sz, FA) + LS.r_a > SizeOfFile, файл содержит оверлей и лучше отказаться от внедрения;
- если LS.v_sz > LS.r_rz, хвост секции содержит данные инициализированные нулями и следует либо отказаться от внедрения, либо перед передачей управления подчистить все за собой;
- дописываем X-код к концу файла;
- устанавливаем LS.r_sz на SizeOfFile – LS.r_off;

- если LS.v_sz >= (LS.r_a + LS.r_sz + (SizeOfFile – (LS.r_a + ALIGN_UP(LS.r_sz, FA))) оставляем LS.v_sz без изменений; иначе LS.v_sz := 0;
- если LS.v_sz != 0 пересчитываем Image Size;
- при необходимости корректируем атрибуты внедряемой секции: сбрасываем атрибут IMAGE_SCN_MEM_DISCARDABLE, и присваиваем атрибут IMAGE_SCN_MEM_READ;
- пересчитываем Image Size;

Идентификация пораженных объектов. Внедрения этого типа идентифицировать проще всего – они выдают себя присутствием кода в последней секции файла, которой обычно является либо секция неинициализированных данных, либо секция ресурсов, либо служебная секция, например, секция импорта/экспорта или перемещаемых элементов.

Если исходный файл содержал оверлей (или мусор, оставленный линкером), он неизбежно перекрывается последней секций.

Восстановление пораженных объектов. Любовь начинающих программистов к расширению последней секции файла вполне объяснима (более или менее подробное описание этого способа внедрения можно найти практически в любом вирусном журнале), но вот алгоритмические ошибки, совершенные ими непростительны и требуют сурового наказания ну или по крайней мере общественного порицания.

Чаще всего встречаются ошибки трех типов: неверное определение позиции конца файла, отсутствие выравнивания и неподходящие атрибуты секции, причем, большая часть из них необратима и пораженные файлы восстановлению не подлежат.

Начнем с того, что LS.r_off + LS.r_sz не всегда совпадает с концом файла и если файл содержит оверлей, он будет безжалостно уничтожен. Если LS.v_sz < LS.r_sz, то r_sz может беспрепятственно вылетать за пределы файла и разработчик X-кода должен это учитывать, в противном случае в конце последней секции образуется каша.

Очень часто встречается и такая ошибка: вместо того, чтобы подтянуть LS.r_sz к концу X-кода, программист увеличиваем LS.r_sz на размер X-кода, и, если конец последней секции не совпадал с концом оригинального файла, X-код неожиданно для себя окажется в оверлее! К счастью, этой беде легко помочь – просто скорректируйте поле LS.r_sz, установив его на действительный конец файла.

Нередко приходится сталкиваться и с ошибками коррекции виртуальных размеров. Как уже говорилось, увеличивать LS.v_sz на размер X-кода нужно лишь тогда, когда LS.v_sz <= LS.r_sz, в противном случае виртуальный образ уже содержит часть кода или даже весь X-код целиком. Если LS.v_sz != 0, такая ошибка практически никак не проявляет себя, всего лишь увеличивая количество памяти, выделенной процессору, но если LS.v_sz == 0, после внедрения он окажется равным… размеру X-кода, который много меньше размера всей секции, в результате чего ее продолжение не будет загружено и файл откажет в работе. Для возвращения его в строй, просто обнулите поле LS.v_sz или вычислите его истинное значение.

После изменения виртуальных размеров секции, требуется пересчитать Image Size, что многие программисты делают неправильно, либо просто суммируя виртуальный размеры всех секций, либо увеличивая его на размер внедряемого кода, либо забывая округлить полученный результат на границу 64 Кбайт, либо допуская другие ошибки. Правильный алгоритм вычисления Image Size выглядит так: LS.v_a + ALIGN_UP((LS.v_s)? LS.v_s:LS.r_sz, OA).

Самый безобидный баг – неудачные атрибуты расширяемой секции, например, внедрение в DISCARDABLE-секцию, которой, в частности является секция перемещаемых элементов, обычно располагающая в конце файла. Коррекция атрибутов должна решить эту проблему.

Для удаления X-кода из файла, просто отберите у него управления, отрежьте sizeof(X-code) байт от конца последней секции и пересчитайте значения полей: Image Base, LS.r_sz и LS.r_off.

категория C: создание своей собственной секции

Альтернативной расширению последней секции файла, стало создание своей собственной секции, что не только "модно", но и технически более грамотно. Теперь, по крайней мере, ни оверлеи, ни таблицы перемещаемых элементов не будут понапрасну болтаются в памяти.

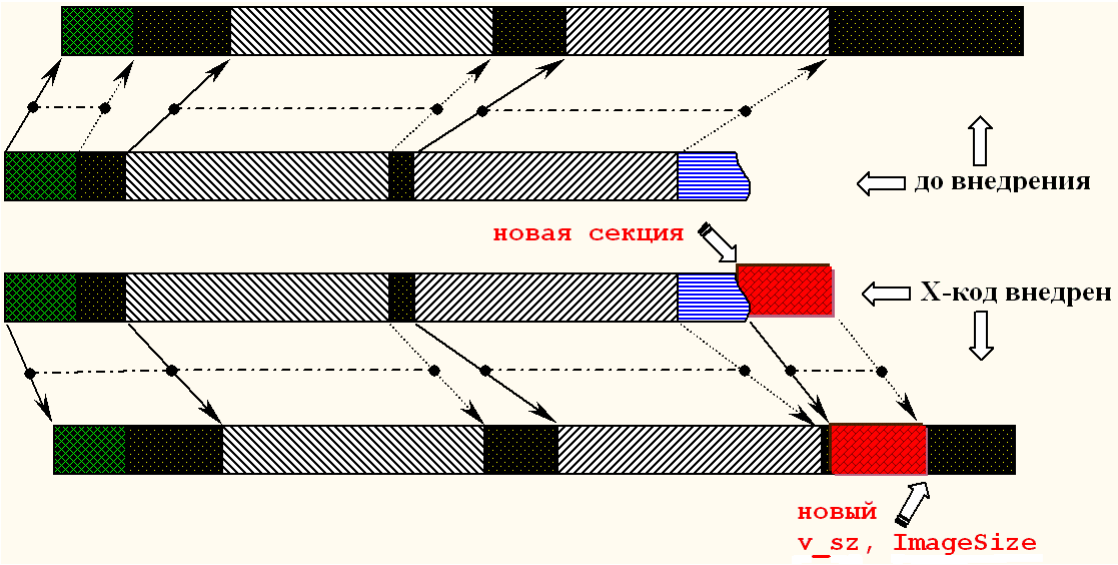

Рисунок 14 внедрение X-кода в файл путем создания собственной секции

Внедрение: обобщенный алгоритм внедрения выглядит так:

- загружаем PE-заголовок и смотрим, что расположено за таблицей секций;
- если здесь не нули, отказываемся от внедрения;
- если (e_lfanew + SizeOfOptionalHeader + 14h + (NumberOfSections + 1)*40) > SizeOfHeaders, раздвигаем заголовок как показано в внедрение в PE-заголовок, а если это невозможно, отказываемся от заражения;
- дописываем X-код к концу файла;
- увеличиваем NumberOfSections на единицу;
- выравниваем LS.r_sz на величину FA;
- дописываем к таблице секций еще один элемент, заполняя поля следующим образом:
  - имя : не имеет значения;
  - v_a : LS.v_a + ALIGN_UP((LS.v_sz)? LS.v_sz: LS.r_sz), Section Alignment);
  - r_offset : SizeOfFile;
  - v_sz : sizeof(X-code) или 0x0;
  - r_sz : sizeof(X-code);
  - Charic. : IMAGE_SCN_CNT_CODE | IMAGE_SCN_MEM_EXECUTE;
  - остальн.: 0x0;
- пересчитываем Image Size;

Идентификация пораженных объектов. Внедрения этого типа легко распознаются по наличию кодой секции в конце файла (стандартно кодовая секция всегда идет первой).

Восстановление пораженных объектов. Ошибочное определение смещения внедряемой секции обычно приводит к полной неработоспособности файла без малейших надежд на его восстановление (подробнее об этом мы уже говорили в предыдущем разделе). Ошибки остальных типов менее коварны.

Живая классика – не выровненный физический размер предпоследней секции файла. Как уже говорилось выше, выравнивать физические размер последней секции необязательно, но при внедрении новой секции в файл, последняя секция становится предпоследней со всеми отсюда вытекающими последствиями.

категория C: расширение серединных секций файла

Внедрение в середину файла относится к высшему пилотажу и обеспечивает X-коду наибольшую скрытность. Предпочтительнее всего внедряться либо в начало, либо в конец кодовой секции, которой в подавляющем большинстве случаев является первая секция файла. Этот алгоритм наследует все лучшие черты создания оверлея в середине, многократно усиливая их: внедренный X-код принадлежит страничному имиджу, оверлея нет, поэтому нет и конфликтов с протекторами/упаковщикам.

Внедрение в начало. Внедрение в начало кодовой секции можно осуществить двояко: либо сдвинуть кодовую секцию вместе со всеми последующими за ней секциями вправо, физически переместив ее в файле, скорректировав все ссылки на абсолютные адреса в страничном имидже, либо уменьшить v_a и r_off кодовой секции на одну и туже величину, заполняя освободившееся место X-кодом, тогда ни физические, ни виртуальные ссылки корректировать не придется, т. к. секция будет спроецирована в память по прежним адресам.

Легко показать, что перемещение кодовой секции при внедрении X-кода в ее начало, осуществляется аналогично перемещению секции данных, при внедрении кода в конец, и поэтому во избежании никому не нужного дублирования описывается в одноименном разделе (см. "внедрение в конец"), здесь же мы сосредоточимся на западной границе кодовой секции и технических аспектах ее отодвижения вглубь заголовка.

Собственно говоря, вся проблема в том, что подавляющее большинство кодовых секций начинается с адреса 1000h, – минимального допустимого адреса, диктуемого выбранной кратностью выравнивания OA, так что отступать уже некуда – заголовок за нами. Здесь можно поступить двояко: либо уменьшить базовый адрес загрузки на величину кратную 64 Кб и скорректировать все ссылки на RVA-адреса (что утомительно, да и базовый адрес загрузки подавляющего большинства файлов – это минимальный адрес, поддерживаемый Windows 9x), либо отключить выравнивание в файле, отодвинув границу на любое количество байт, кратное двум (но тогда файл не будет запускаться под Windows 9x).

Типовой алгоритм внедрения путем уменьшения базового адреса загрузки выглядит так:

- считываем PE-заголовок;
- если Image Base < 1.00.00h и перемещаемых элементов нет отказываемся от внедрения;
- если Image Base <= 40.00.00h и перемещаемых элементов нет, лучше отказаться от внедрения, т. к. файл не сможет запускаться в Windows 9x;
- внедряем 1.00.00h байт в заголовок по методу, описанному в "раздвижка заголовка", оформляя все 1.00.00h байта как оверлей (т. е. оставляя SizeOfHeaders неизменным), а если это невозможно, отказываемся от внедрения;
- уменьшаем FS.v_a и FS.r_off на 1.00.00h;
- увеличиваем FS.r_sz на 1.00.00h;
- если FS.v_sz не равен нулю, увеличиваем его на 1.00.00h;
- увеличиваем виртуальные адреса всех секций, кроме первой на 1.00.00h;
- анализируем все служебные структуры, перечисленные в DATA DIRECTORY (таблицы экспорта, импорта, перемещаемых элементов и т.д.), увеличивая все RVA-ссылки на 1.00.00h;
- внедряем X-код в начало кодовой секции от FS.r_off до FS.r_off + 1.00.00;
- пересчитываем Image Size;

Типовой алгоритм внедрения путем переноса западной границы первой секции, выглядит так:

- считываем PE-заголовок;
- если OA < 2000h лучше отказаться от внедрения, т. к. файл будет неработоспособен на Windows 9x, но если мы все-таки хотим внедриться, то:
  - устанавливаем FA и OA равными 20h;
  - для каждой секции: если NS.v_a – CS.v_a – CS.v_sz > 20h, подтягиваем CS.v_sz к NS.v_a – CS.v_a;
  - для каждой секции: если v_sz > r_sz, увеличиваем длину секции на v_sz – r_sz байт, перемещая все остальные в физическом образе и страничном имидже;
  - для каждой секции: если v_sz < r_sz, подтягиваем v_sz к NS.v_a – CS.v_a, добиваясь равенства физических и виртуальных размеров;
- внедряем в заголовок ALIGN_UP(sizeof(X-code), OA) байт, оформляя их как оверлей;
- уменьшаем FS.v_a и FS.r_off на ALIGN_UP(sizeof(X-code), OA);
- внедряем X-код в начало первой секции файла;
- пересчитываем Image Size;

Внедрение в конец. Чтобы внедриться в конец кодовой секции, необходимо раздвинуть не страничный имидж, заново пересчитав ссылки на все адреса, т. к. старых данных на прежнем месте уже не окажется. Задача кажется невыполнимой (встраивать в X-код полноценный дизассемблер с интеллектом ИДЫ не предлагать), но решение лежит буквально на

поверхности. В подавляющем большинстве случаев для ссылок между секциями кода и данных используются не относительные, а абсолютные адреса, перечисленные в таблице перемещаемых элементов (при условии, что она есть). В крайнем случае, абсолютные ссылки можно распознать эвристическими приемами – если (Image Base +Image Size) >= Z >= Image Size, то Z – эффективный адрес, требующий коррекции (разумеется, предложенный прием не слишком надежен, но все же он работает).

Типовой алгоритм внедрения выглядит так:

- считываем PE-заголовок;
- если нет перемещаемых элементов, лучше отказаться от внедрения, т. к. файл может потерять работоспособность;
- находим кодовую секцию файла;
- если CS.v_sz ==0 или CS.v_sz >= CS.r_sz, увеличиваем r_sz кодовой секции файла;
- если CS.v_sz < CS.r_sz, CS.r_sz := NS.r_off + ALIGN_UP(sizeof(X-code), FA);
- если CS.v_sz < CS.r_sz, CS.v_sz := CS.r_sz;
- физически сдвигаем все последующие секции на ALIGN_UP(sizeof(X-code), FA) байт, увеличивая их r_off на туже самую величину;
- сдвигаем все последующие секции в страничном имидже, увеличивая их v_a на ALIGN_UP(sizeof(X-code), OA) байт;
- если таблица перемещаемых элементов присутствует, увеличиваем все абсолютные ссылки на перемещенных секции на ALIGN_UP(sizeof(X-code), OA) байт, если же таблицы перемещаемых элементов нет, использует различные эвристические алгоритмы;
- пересчитываем ImageSize;

Идентификация пораженных объектов: данный тип внедрения до сих пор не выловлен в живой природе, поэтому, говорить об его идентификации преждевременно.

Восстановление пораженных объектов. ни одного пораженного объекта пока не зафиксировано.

категория Z: внедрение через авто загружаемые dll

Внедриться в файл можно даже не прикасаясь к нему. Не верите? А зря! Windows NT поддерживает специальный ключ реестра, в котором перечислены DLL, автоматически загружающиеся при каждом создании нового процесса. Если Entry Point динамической библиотеки не равна нулю, она получит управление еще до того, как начнется выполнение процесса, что позволяет ей контролировать все, происходящие в системе события (такие, например, как запуск антивирусных программ). Естественно, борьба с вирусами под их руководством ни к чему хорошему не приводит, и система должна быть обеззаражена. Убедитесь, что в HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs перечислены только легальные динамические библиотеки и нет ничего лишнего!

заключение

Хотя собранная автором коллекция методов внедрения претендует на полноту (если вам встречались программы, внедряющиеся в файл другим способом, пожалуйста, дайте об этом знать), технических прогресс не стоит на месте и каждый день приносит новые технологии и идеи. Поэтому не воспринимайте эту статью как догму. Это всего лишь путеводитель по кибернетической стране виртуального мира.