новая жизнь старого софта или хачим висту

крис касперски, ака мыщъх, no-email

всем известно как "хорошо" дружит висла с древним софтом, написанным до нее. и хотя сейчас ситуация уже не та, что пару лет назад — над проблемой совместимости активно работают с двух сторон баррикад: ms добавляет специальный "обходной" код для поддержки старых программ, а производители софта взяли за правило тестировать новые приложения под вислой и даже для ранее выпущенных версий имеются специальные заплатки, патчи и фиксы. однако, количество программ (и драйверов!) не совместимых с вислой по-прежнему велико и пока другие пользователи ждут у моря погоды, мы — хакеры — уже вовсю орудуем напильником и дизассемблером, главным образом ковыряя 32х битные системы (под них большое всего несовместимого софта), но и про x86-64 тоже стоит замолвить слово!

введение

Мы не будем говорить о (не)целесообразности перехода на Вислу и агитировать за то, чтобы остаться сидеть на XP (или даже под W2K), поскольку, если следовать этой логике, самой привлекательной операционной системой окажется Linux, а то и FreeBSD. Люди переходят на Вислу по самым разным мотивам (например, сложности поиска драйверов под XP для ноутбука, на котором предустановленна Висла) или просто хотят очутиться в авангарде прогресса.

Лично мыщъх менять W2K на Вислу не собирается, но и полностью игнорировать ее существование тоже не может хотя бы уже потому, что если его программы не пойдут на Висле, объемы продаж упадут в разы. Лого "Vista compatible" дорогого стоит. Microsoft выпустила кучу спецификаций которым мы должны следовать для достижения совместимости и которых столько, что до конца сезона не скурить. Но одних лишь спецификаций мало. Отладчик нужен. И желательно не простой, а ядерный. Но любимый всеми soft-ice под Вислой не идет и поддержка его прекращена много лет тому назад. Хоть грызи зубами лед, хоть убейся о газель, а другого такого отладчика нет и не будет. Вот потому, в настоящее время мыщъх занят переносом soft-ice под Вислу и Server 2008.

Километры распечаток, бессонные ночи, проведенные в ядре Вислы, лицо с характерным отпечатком клавиатуры, на которую мыщъх падал, когда не оставалось сил доползти до топчана… Виста (в худшем смысле этого слова) — это _действительно_ большой шаг вперед и изменений там… Чем более в нее зарываешься дизассемблером, тем сильнее поражаешься — как вообще старые приложения ухитряются на ней работать?! Ведь исходя из самых общих рассуждений — не должны. Ну никак не должны. А работают…

Рисунок 0 Висла и женщины — две вещи, которые не перестают удивлять мыщъх'а

битва за совместимость в лагере Реймонда Чена

На самом деле, ничего удивительно здесь нет. История четверть вековой давности вновь повторяется. Когда вышла Windows 95, то приложения, написанные под MS-DOS/Windows 3.x, хором отказывались под ней работать (особенно игры) и, чтобы завоевать рынок, парням из Microsoft (в число которых входил и Реймонд Чен /* Raymond Chen */ знаменитый своим блогом "The Old New Thing": http://blogs.msdn.com/oldnewthing/) не вылезали из-под дизассемблеров и деббагеров, разбираясь в чем причина отказа.

В большинстве случаев, дело было не в Windows, а в ошибках сторонних разработчиков, которых Microsoft тыкала носом в свое же дерьмо. Типа — нагадили, ну так исправляйте! Если же разработчики шли на принцип и договориться с ними по хорошему не получалось, приходилось править код самой операционной системы, добавляя в загрузчик исполняемых файлов специальный модуль, распознающий конфликтные приложения и выбирающую адекватную модель поведения операционной системы или же правящий код конфликтного приложения непосредственно в оперативной памяти (забавно, но по американским законам для исправления ошибок сторонних разработчиков Microsoft должна была получить от них разрешение на правку багистного кода, вот тебе бабушка и демократия!)

Раздел реестра **AppCompatibility** Менеджера Сессий (Session Manager), содержал (и содержит!) _сотни_ приложений с указанием действий, которые система должна предпринять для их запуска (полный путь выглядит так: HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCompatibility). В частности, некогда популярная игра Sim-City использовала уже освобожденный блок памяти, закладываясь на неизменность его содержимого, что прокатывало в однозадачной MS-DOS, но вот перед запуском Sim-City в многозадачной Windows приходилось применять специальную модель распределения памяти, гарантирующую, что освобожденный блок не будет использован никем другим вплоть до завершения приложения.

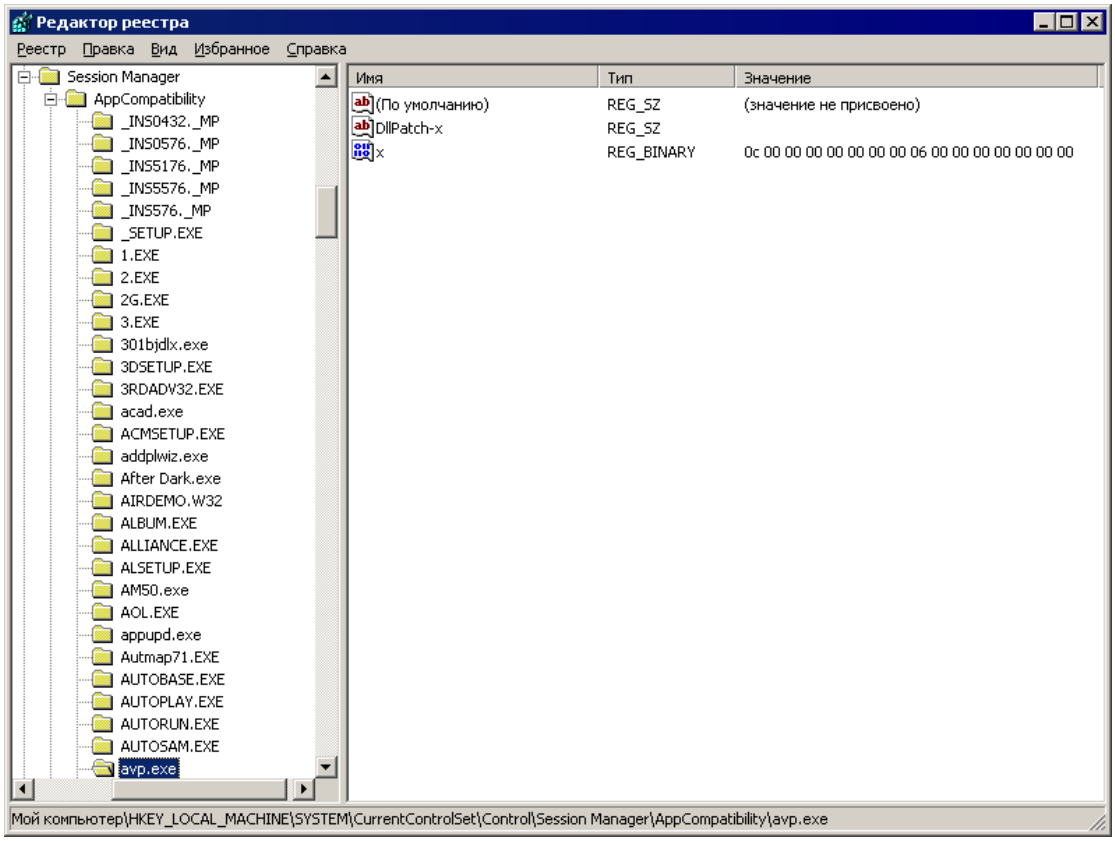

Рисунок 1 содержимое ветки реестра "AppCompatibility" с программами, для которых система предпринимает определенные действия для обеспечения обратной совместимости

Листая содержимое AppCompatibility (см. рис. 1) не устаешь поражаться сколько же здесь знакомых имен! И avp.exe, и commanods.exe и doom95.exe, и nero40.exe, и даже DirectX7a.exe, выпущенный непосредственно самой Microsoft! Выходит, что система имеет встроенные (hard-coded build-in) средства для обеспечения совместимости с приложениями, использующими грязные приемы программирования, например, обращающихся к недокументированным (и потому подверженным постоянным изменениям) API-функциям и структурам данных! Как их примерить с Вислой? Добавляем новую запись в раздел AppCompatibility и… увы! не все так просто!!! Формат ключа AppCompatibility (и флаги, управляющие поведением операционной системы) недокументирован и мыщъх как раз сейчас занимается их всесторонним изучением.

Опознание приложений осуществляется… по имени исполняемого файла (сюрприз, да?!) и потому, мы можем попытать счастья последовательно переименовывая конфликтную программу в одно из имен, прописанных в AppCompatibility — а вдруг их несовместимости совпадут и все магическим образом заработает?! Аналогичного результата можно добиться скопировав существующую ветвь AppCompatibility и присвоив ей имя нашего исполняемого файла (некоторые приложения отказываются работать, если их исполняемый файл переименован). Способ, конечно, грязный, но… он достаточно часто срабатывает, чтобы от него отказываться.

Плохая новость — вирусы с червями уже взяли ключ AppCompatibility на вооружение. Что они могут с ним сделать?! Да много чего! Прописав сюда explorer.exe они не только отключат все защиты, какие только есть, но еще и (при желании с их стороны) пропатчат explorer.exe (или любую другую программу, например, антивирус) так, чтобы она выполняла зловредный shell-код. И количество таких вирусов неуклонно растет!

К сожалению, несмотря на свое высокое положение в Microsoft, Реймонд Чен так и не смог прищемить коллектив разработчиков Вислы, решивших, что самое главное — это написать как можно больше никому ненужного кода, натянуть на него интерфейс с кучей спецэффектов, а на совместимость можно и забить. Но пользователи все-таки не дураки и теперь, чтобы заставить старый код работать в новой операционное среде (или операционном четверге?! был пьян — не помню), приходиться добавлять специальные промежуточные слои, эмулирующие поведение старой системы, ценой разбухания кода и снижения защищенности, но об этом мы

еще поговорим, а пока рассмотрим некоторые технические аспекты AppCompatibility. Мы же ведь хакеры! И пока не пропустим Вислу через дизассемблер, ни за что не успокоимся!

>>> форсированный запуск в режиме совместимости

Прежде чем лезть вглубь и рыть тоннель сквозь толщу скал (см. рис. 6), вспомним, что пользователи — тоже люди, и заставлять их трахаться с дизассемблером — по меньшей мере негуманно, а потому Microsoft предусмотрительно реализовала механизм форсированной совместимости.

Щелкаем правой клавишей по исполняемому файлу, выбираем "Свойства" (Properties), там находим вкладку "Совместимость" (Compatibility) с выпадающим списком операционных систем от XP SP2 до… Windows 95, которые Висла будет эмулировать для устранения конфликтов (см. рис. 2).

(Там же находится чек-бокс "Выполнять эму программу от имени администратора" — ну так он еще со времен W2K торчит, только там это делалось через runas, у которой, кстати говоря, и настроек побольше).

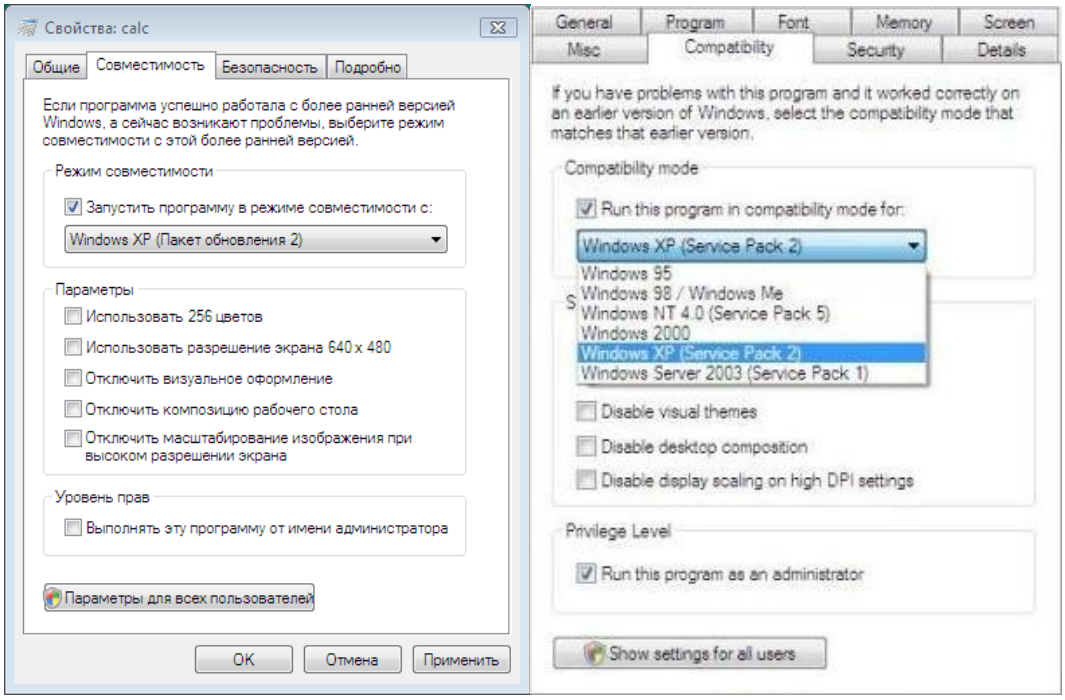

Рисунок 2 ламерский способ запуска программ в режиме совместимости

Это же крышей поехать можно, если представить какие улетные перспективы открываются!!! Однако, попытка запустить под Вислой игрушку, прекрасно идущую под Windows 9x, быстро возвращает крышу на место и эйфории приходит конец. Вот так облом! Для тех, кто не в теме — еще во времена ранней молодости MS-DOS в штатный комплект поставки входила утилита setver, позволяющая задать любую версию системы, какую нам только заблагорассудиться, поскольку некоторые программы отказывались работать с MS-DOS, чьей версии они не знали. Иногда из предосторожности, но чаще по причине использования недокументированных функций и структур данных, меняющихся от версии к версии, а к ним-то setver даже не прикасалась и пользы от нее…

Точно так обстоят дела и с Вислой. Выбор операционной системы влияет только на номер версии, возвращаемый данному приложению и некоторые системные политики, но ядро и сопутствующие ему библиотеки остается прежними. А ядро у Вислы "выдрано" из Server'a 2003 и с XP (базирующейся на W2K) имеет мало общего, не говоря уже о линейке 9x! Все эти ядра писали _разные_ коллективы разработчиков, придерживаясь (или не придерживаясь) определенных внутрифирменных стандартов и спецификаций, а спецификации, как хорошо известно, крайне редко бывают полными и однозначными. Вот и получилось, что одни и те же API-функции каждый коллектив разработчиков реализовал на свой манер и их поведение слегка отличается, но этого "слегка" вполне достаточно для краха приложений. Полного списка

отличий ядра XP от Server 2003 нет ни у кого, даже у Microsoft, а потому прикинуться другой системой Висла не может при всем своем желании!

А если приложение (запущенное в режиме совместимости, конечно) попытается под Вислой вызывать native-API функцию от 9x, то… откуда же ей взяться в NT-подобном ядре?! А ядро в Висле всего одно (ну не совсем одно, конечно, но уж точно не полная коллекция всего, что успели понаписать в Microsoft за это время).

С другой стороны, Висла поддерживает аппаратную виртуализацию, в разы снижающую накладные расходы на эмуляцию. Просто устанавливаем Virtual PC (а обладателем Server 2008 и устанавливать ничего не нужно!) и создаем столько виртуальных машин, сколько заблагорассудиться, устанавливая на них "зоопарк" операционных систем, заставляющих забыть о проблеме совместимости раз и навсегда.

техническая информация по AppCompatibility

Код, обеспечивающий совместимость Вислы со старыми приложениями, не сосредоточен в каком-то конкретном файле, а размазан по всей системе — одни компоненты распознают загрузку проблемных программ и выставляют скрытые флаги, подхватываемые другими компонентами, обитающими на различных уровнях иерархии: от прикладного режима до самого ядра.

Хорошо, заходим FAR'ом в каталог с Вислой и ищем контекстным поиском строку AppCompatibility во всех исполняемых файлах и динамических библиотеках (в уникоде, разумеется). Получаем: USER32.DLL, SHELL32.DLL и SLAYERUI.DLL. Всего три библиотеки?! Но если вспоминать, что это всего лишь вершина айсберга, обрабатывающая обозначенную ветвь реестра и выполняющая простейшие действия по обеспечению обратной совместимости, перекладывая основную работу на ядро, то мало не будет! Тут курить, долбить и ковырять хватит на весь сезон и еще останется!

Анализ показывает, что первичная обработка AppCompatibility осуществляется в недрах библиотеки aclayers.dll с внутреннем именем "Shim Accessory DLL". В переводе с английского "shim" означает "прокладку", а "accessory" — нечто вспомогательное. Короче, промежуточный слой для обеспечения обратной совместимости.

Дизассемблер показывает не только полный путь к ключу AppCompatibility, но и параметры, управляющие режимом совместимости, например, "DllPatch-y" (см. листинг 1), а потому, если кому-то потребуется устранить конфликт с программой, отсутствующей в данном списке, то рыть нужно именно отсюда. Мыщъх уже роет и как наroет — выложит полное описание ключей реестра на всеобщее обозрение.

```
.text:23008310         unicode 0, <y>,0
.text:23008314 aDllpatchY_0:                   ; DATA XREF: sub_23021A93+ED↓o
.text:23008314         unicode 0, <DllPatch-y>,0
.text:23008364 aRegistryMachin:                ; DATA XREF: sub_23021A93+68↓o
.text:23008364 unicode 0,      <\Registry\Machine\System\CurrentControlSet\C>
.text:23008364 unicode 0,      <ontrol\Session Manager\AppCompatibility>,0
```

Листинг 1 фрагмент кода aclayers.dll, , обращающийся к AppCompatibility

Впрочем, не стоит обольщаться. Львиная доля возможностей библиотеки "Shim Accessory DLL" реализована в ней самой. Она часто прибегает к перехвату API-функций из KERNEL32.DLL и навешивает на них свои собственные обработчики, изменяющих поведение перехваченных функций или выполняющих дополнительные действия для устранения конфликтов с Вислой непосредственно в контексте "подопытного" приложения (см. рис. 3).

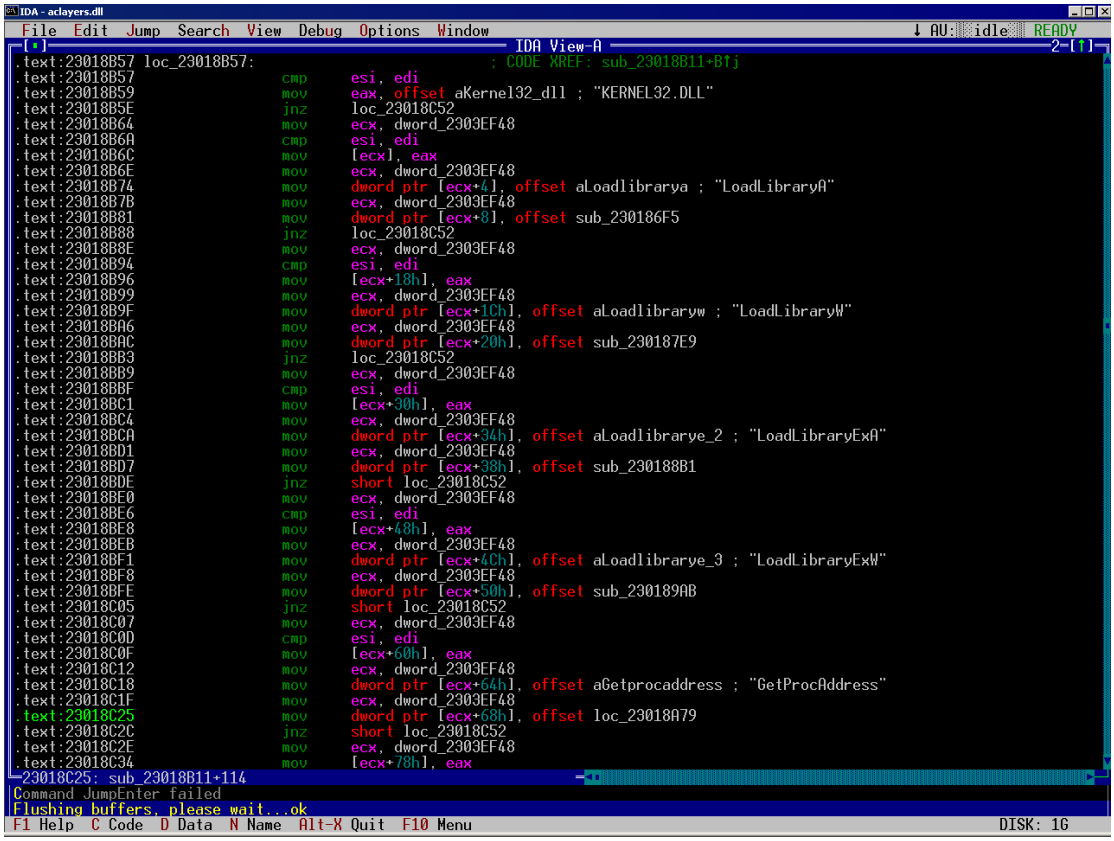

Рисунок 3 фрагмент aclayers.dll

Фактически, ветвь AppCompatibility содержит всего лишь базовые указания: какие действия должна выполнить библиотека aclayers.dll перед запуском конфликтных приложений, при этом 90% действий реализовано внутри aclayers.dll, а не в реестре. То есть, возможности "рукотворного" добавления новых записей в AppCompatibility достаточно жестко ограничены архитектурой системы, однако, если не лезть в исправление тяжелых конфликтов, предоставленного функционала хватает с головой.

Библиотека USER32.DLL (обертка вокруг драйвера WIN32K.SYS, реализующая пользовательских интерфейс вместе с графической подсистемой), так же обращается к AppCompatibility (см. листинг 2) из API-функции ClientThreadSetup, которая в свою очередь вызывается из API-функции UserClientDllInitialize, подготовляющей оконную и графическую подсистему для использования в контексте конкретного приложения с учетом его требований к обратной совместимости:

```
.text:77E43120 aRegistryMach_5:              ; DATA XREF: sub_77E4307E+32↑o
.text:77E43120 unicode 0,     <\Registry\Machine\System\CurrentControlSet\C>
.text:77E43120 unicode 0,     <ontrol\Session Manager\AppCompatibility>,0
```

Листинг 2 фрагмент кода USER32.DLL, обращающийся к AppCompatibility

Вот в эту часть кода лучше не лезть. Оконная подсистема Вислы претерпела значительные изменения, но разобраться какие именно изменения разваливают ранее написанную программу, намного сложнее, чем кажется. Тут одного отладчика и дизассемблера явно недостаточно! Необходимо трассировать программу под новой и старой системой, сравнивая результаты прогонов, представляющих собой огромные log-файлы с кучей непонятных строк.

что нового в Висле и Server 2008

Microsoft предупреждает: использование недокументированных API-функций и структур данных опасно для вашего здоровья! Вот только программисты чихать хотели на свое здоровье (в смысле, на совместимость) и ведут крайне нездоровый образ жизни (стиль кодирования) буквально нашпигованный недокументированными возможностями. И ведь не от хорошей жизни!!!

Взять хотя бы функцию **OpenThread**, отсутствующую в первой редакции стандарта win32-API, разработчики которого умышленно изъяли ее в целях безопасности. Мол, нечего открывать чужие потоки, а если поток хочет чтобы его открыли — пускай передает свой дескриптор через один из многочисленных механизмов, имеющихся в распоряжении программиста. ОК, вообразим себе поток, созданный зловредным вирусом, который (в здравом уме и трезвой памяти) явным образом передает свой дескриптор антивирусам (а для этого еще и протокол взаимодействия вируса с антивирусом придумать нужно и стандартизовать, чтобы все ему подчинялись), мол, братья антивирусы, проверьте меня пожалуйста и, убедившись, что я действительно вирус, а не полезная программа, тут же прибейте меня без суда и следствия. А вот дудки!!! Никакой вирус сотрудничать с антивирусом не будет и без OpenThread разработчикам последних пришлось бы очень туго. Самое интересное, что обозначенная функция в NT-системах таки есть (пускай и не упомянута в документации). В 9x получить дескриптор потока чуть-чуть сложнее, но все-таки возможно.

Мыщъх повторяет еще раз: к недокументированным возможностям программисты обращаются тогда, когда легальными средствами поставленная задача не решается (либо ее решение выходит слишком громоздким и не удовлетворяющим ТЗ). Конечно, встречаются и маньяки, обращающие к недокументированным функциями без всякой нужды (особенно много таких среди разработчиков протекторов и вирусописателей), но это контингент отдельного дурдома, которому программирование вообще противопоказано по жизни. Попав на Linux, они создают программы, запускающиеся только на их собственных машинах, да и то только до первой перекомпиляции ядра. Впрочем, мы отвлеклись. Вернемся к нашим баранам.

Итак, недокументированные API-функции и структуры. Как ни печально, но они (в той или иной мере) встречаются практически в каждой программе, включая коммерческие приложения. Главным образом программисты лезут в TIB (Thread Information Block – Блок Информации о Потоке), на который указывает селектор fs и который содержит указатель на PEB (Process Environment Block – Блок Окружения Процесса), лежащий по смещению 30h байт от начала TIB и содержащей целую кладезь информации. Недокументированной, естественно. Но уже давно исследованной хакерами и описанной в доступной литературе (см. например, http://book.itzero.com/read/microsoft/0507/Microsoft.Press.Microsoft.Windows.Internals.Fourth.Edition.Dec.2004.internal.Fixed.eBook-DDU_html/0735619174/ch06lev1sec3.html и undocumented.ntinternals.net/UserMode/Undocumented Functions/NT Objects/Process/PEB.html).

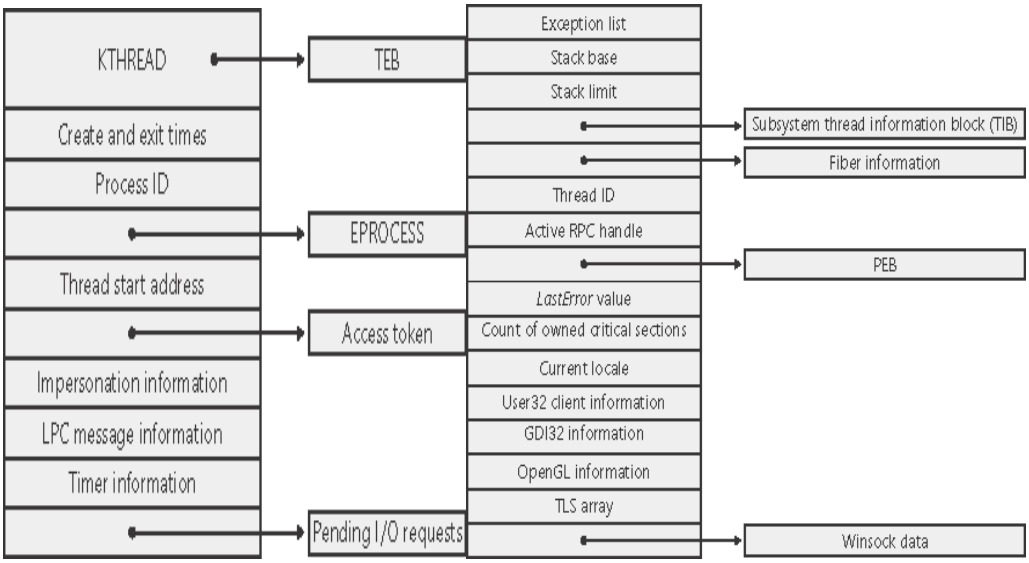

Рисунок 4 недокументированные структуры данных ETHREAD (слева) и TEB (справа)

Проблема в том, что содержимое TIB'а и PEB'а, будучи внутренней кухней операционной системы, непредсказуемым образом меняется от версии к версии и неизменным остается лишь небольшое подмножество базовых полей, да и то без всяких гарантий, что в следующей версии Windows здесь не окажется что-то другое.

Рассмотрим следующий фрагмент кода, выдернутый из отладчика Soft-Ice:

```
mov     eax, fs:[124h]        ; current thread

#if OS == XP
```

```
        mov     eax, [eax+44h] ; KPROCESS
#endif

#if OS == Server2003
        mov     eax, [eax+38]  ; KPROCESS
#endif

mov     eax, [eax+18]          ; DirectoryTableBase
```

Листинг 3 фрагмент псевдокода из soft-ice ("псевдо" потому что в действительности обозначенный код намного сложение)

Вспомнив, что Висла основана на ядре от Server 2003, мы поймем почему большинство программ из тех, что идут на XP, отказываются работать под Вислой (а если учесть и многочисленные изменения, внесенные в ядро Server'a 2003 разработчиками Вислы) становится понятен, где порылась собака на почве "хорошей" обратной совместимости. Самое печальное, что не существует никакой возможности вернуть недокументированные структуры взад, поскольку они в действительности принадлежат не процессу, а ядру. Теоретически вполне возможно заставить ядро работать с несколькими версиями структур данных, но практически для этого львиную долю ядерного кода пришлось бы переписать заново, и выход Вислы состоялся бы не в 2007 году, а… лет эдак через десять. Шутка. На самом деле, Microsoft (прекрасно осведомленная, что программисты напропалую используют недокументированные структуры данных), сохранила наиболее популярные поля на своих местах и заложила механизм эмуляции некоторых полей из тех, что были все-таки перемещены, но… пока он реализован лишь в зачаточной форме.

Другое существенное отличие от XP – механизм рандомизации адресного пространства — Address Space Layout Randomization или, сокращенно, ASLR, впервые появившейся в мире UNIX и скопированный фирмой Microsoft для затруднения хакерских атак. Ну… нападки на Вислу прекращаться не собираются, а вот системные динамические библиотеки теперь отображаются в память по случайным адресам, выбираемым на стадии загрузки операционной системы. Стек и куча так же рандомизированны. Приложения, написанные до Вислы грузятся по адресам, прописанным в их заголовке, а вот для новых приложений имеемся возможность задействовать ASLR и для самого исполняемого модуля, но все-таки вернемся к старым программам. Может ли рандомизация служить причиной отказа их работоспособности. Microsoft полагает, что нет, но практика выявляет обратное.

Вот вполне типичная ситуация — программист сохраняет некоторые структуры данных на диск, в которых оказываются (по чистой случайности, конечно) указатели на локальные переменные, расположенные в стеке. На XP, где стек всегда начинается с одного и того же адреса, эта ошибка (а это именно ошибка) никак не проявляется и все работает. Но стоит только попасть такой программе на Вислу, как она начинает падать стремительным домкратом. Тоже самое относится и к рандомизации кучи. А уж как рандомизация затрудняет отладку можно даже не говорить.

Программистские форумы буквально пестрят вопросами: как отключить это грубанную рандомизацию?! Официально — никак. Но… не будет спешить с выводами, а лучше рассмотрим еще одну интересную особенность Вислы — запрет на исполнение кода в стеке. Впервые этот механизм появился в XP SP2 при обязательной аппаратной поддержке со стороны процессоров, реализовавших пресловутые биты NX/XD, позволяющие управлять атрибутом Executable на уровне отдельных страниц. До этого x86 поддерживали атрибут исполняемый исключительно на уровне селекторов, то есть, учитывая плоскую модель памяти Windows, не поддерживали его вообще. И хотя win32 API предполагает наличие подобного атрибута, программистам было хорошо известно, что x86 процессоры реально поддерживают только два атрибута — атрибут на доступ и атрибут на запись, таким образом PAGE_READ тождественен PAGE_EXECUTE. Большинство программ, исполняющий код в стеке или куче, довольствовались атрибутом на чтение и… все работало. Но вот пришли злые дядьки и постановили: дальше так жить нельзя, типа от этого вирусы разводятся, shell-код выполняется и вообще. Так появился механизм DEP (Data Execution Prevention), предотвращающий исполнение данных там, где по мнению создателей Windows они исполняться не должны. В XP SP2 по умолчанию защищались лишь системные компоненты, но вот в Висле…

К той же категории относится и механизм SafeSEH, впервые анонсированный в XP SP2, но реально доделанный лишь с выходом Вислы. Его безопасность заключается в том, что обработчик исключений не может размещаться в стеке и, что самое неприятное, обработчик не может назначаться динамически. Компилятор (вместе с линкером) должен создавать статические таблицы, размещающиеся в специальной секции PE-файла. Последнее требование

относится, как нетрудно догадаться, только к новым приложениям (у них в заголовке явным образом прописано использование SafeSEH), но вот запрет на помещение обработчика исключения в стек распространяется на _все_ приложения, а ведь многие из них именно так и делает — динамически назначают обработчик и кладут его в стек. На XP SP2 и Висле они, соответственно, работать не могут.

Археологические раскопки недр NTDLL.DLL выявляют весьма любопытный факт. Системный загрузчик проверяет имена секций каждого запускаемого исполняемого файла (динамической библиотеки) и если находит секцию с именем ".aspack", ".pcle", ".sforce" (см. листинг 4) предпринимает ряд действий по обеспечению обратной совместимости и взводит флаги, вырубающие кучу защит, появившихся в Висле и не отключаемых легальным путем. Никаких других дополнительных проверок не выполняется. Достаточно совпадение имени секции — вот и все.

```
.text:77F24FEC aSecserv_dll   db 'secserv.dll',0     ; DATA XREF: sub_77F0B5E5+36o
.text:77F24FF8 a_sforce       db '.sforce',0         ; DATA XREF: sub_77F0B717+85o
.text:77F25000 a_pcle         db '.pcle',0           ; DATA XREF: sub_77F0B717+6Fo
.text:77F25006 a_aspack       db '.aspack',0         ; DATA XREF: sub_77F0B717+59o
```

Листинг 4 фрагмент кода из NTDLL.DLL с именам секций PE-файла для который задействуется специальный режим совместимости

Сюрприз, да?! Оказывается, если упаковать проблемную программу упаковщиком ASPack (кстати говоря, скупленным фирмой Star Force), то шансы, что она заработает под Вислой существенно возрастут. А можно и не упаковывать, а просто взять в руки hiew и переименовать секцию ".text" (или ".CODE") в ".aspack".

Секции с именами ".sforce" принадлежат файлам, защищенным протектором Star Force (странно, что Microsoft вообще знает об этой недоделке), для совместимости с которым приходится не только вырубать кучу защит, но даже эмулировать особенности поведения некоторых недокументированных API-функций и структур данных. Короче, ".sforce" намного круче, чем ".aspack"!

Кому принадлежит имя ".pcle" мыщъх не знает. Поиск по Интернету ничего (вразумительного) не дал, но… если есть желание поэкспериментировать, то можно воспользоваться и им.

Короче, берем HIEW, загружаем в него конфликтное приложение, нажатием на <ENTER> переходим в hex-режим, давим на <F3> для активации редактирования и меняем имя кодовой секции (обычно ".text" или ".CODE") на ".sforce" (см. рис. 5). Сохраняем изменения по <F9> и выходим. Если файл использует контрольную сумму для проверки своей целостности ее можно пересчитать с помощью утилиты editbin.exe, входящей в комплект поставки Microsoft Visual Studio, запущенной с ключом "/RELEASE".

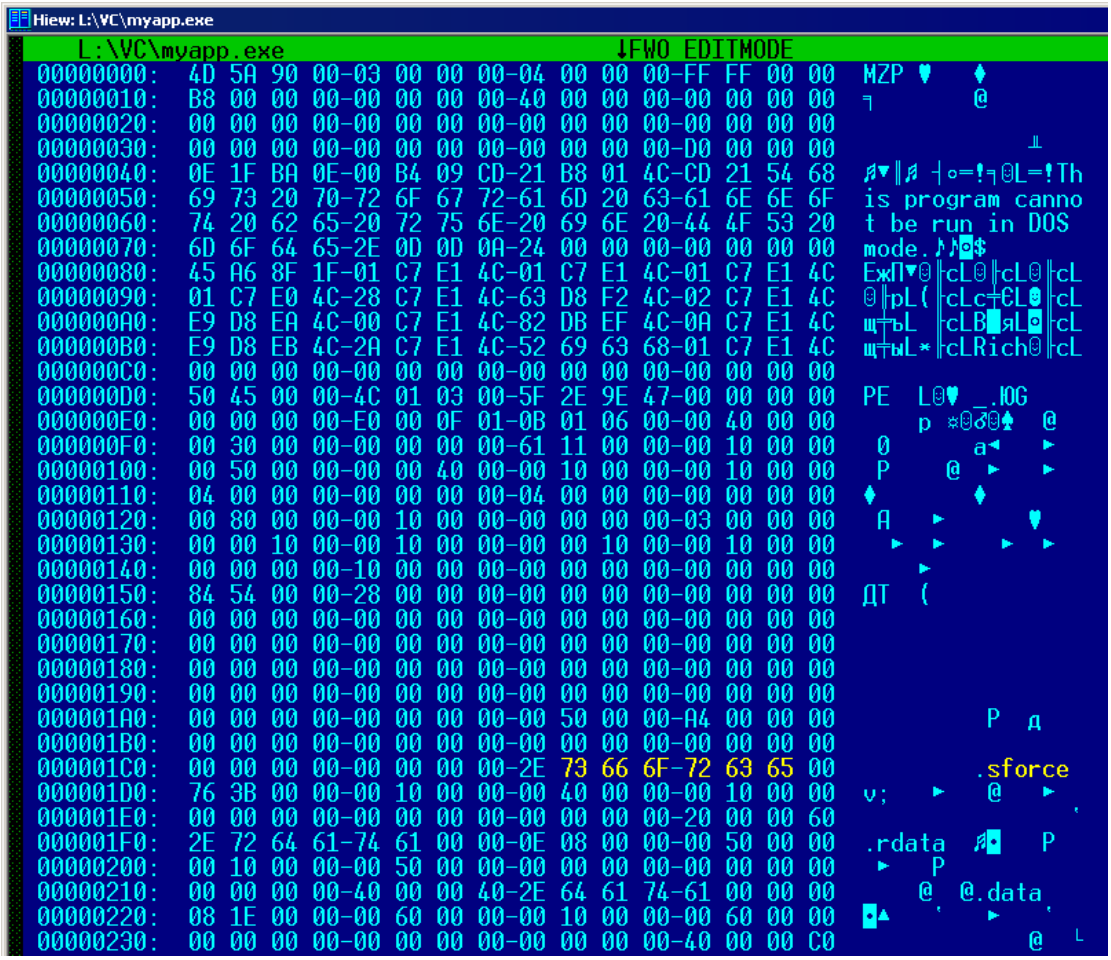

Рисунок 5 хакерский способ запуска конфликтных файлов в режиме совместимости

x86-64: ад на земле или real nightmare

Обратная совместимость налагает серьезные ограничения на дизайн новых версий 32х-битных операционных систем, щемя многие революционные инициативы. Груз ранее написанных программ сдерживает Microsoft, вынуждая ее продвигаться вперед маленькими шагами, каждый из которых приводит к смерти относительно небольшого числа программ. Убивать все приложения сразу — ну и кому нужна такая Windows?! Лучше сразу на Linux перейти…

Архитектура AMD x86-64 оказалась настоящим подарком для Microsoft. На момент выхода первой 64х битной версии Windows никаких программ для нее просто не существовало — вот уж где можно развернуться!!! Тут же на свет появится печально известный механизм PatchGuard, контролирующий целостность ядра от внесения в него каких бы то ни было изменений (например, перехвату системных функций для антивирусного контроля всех открываемых файлов или установки персонального брандмауэра), а цифровая подпись драйверов стала с этого момента обязательной. Неподписанный драйвер не может загрузить даже администратор!!!

Многие из приложений, что нормально работали в 64х-битной XP, отказываются функционировать под Вислой, выбрасывая голубые экраны смерти или не загружая необходимые драйвера из-за отсутствия цифровой подписи.

В электронном журнале "Uninformed" опубликована любопытная статья "PatchGuard Reloaded: A Brief Analysis of PatchGuard Version 3" (http://uninformed.org/?v=8&a=5&t=pdf), объясняющая как отключить этот зловредный PatchGuard, мешающий многим честным программ. Там же выложен демонстраций код "выключателя" (http://uninformed.org/?v=8&a=5&t=code), обновления версия которого входит в состав бесплатного расширения для VM Ware – VMKD (http://www.nynaeve.net/Programs/vmkd.zip), написанного тем же самым хакером, так что ему можно доверять.

ОК, переходим к цифровой подписи. Вообще-то, знаменитый хакерский активатор для Вислы (только не спрашивайте меня где его взять?!) содержит в себе и загрузчик неподписанных драйверов, но мы, как честные люди, пиратский софт не юзаем, легких путей не ищем и, следуя заветам Microsoft, просто задействуем режим отладки с помощью штатной утилиты bcdedit, запущенной с ключом "/debug on", после чего при запуске системы (<F8> во время загрузки) нас будут спрашивать в каком режиме мы хотим работать — обычном или отладочном?

В отладочном драйвера загружаются и без цифровой подписи, но… чтобы мы ненароком не загрузили драйвер, грабящий защищенный цифровой контент, система отрубает DVD. Да… однако, дела! "bcdedit /debug off" все возвращает на свои места.

Рисунок 6 Висла это огромная гора, сквозь которую ведет лишь прорубленная хакерами узкоколейка. и кто знает какие секреты там — в толще скал?