отладчик windbg как API- и RPC-шпион

крис касперски ака мыщъх, ака souriz, aka nezumi, aka elraton, no-email

ранние версии windbg'а от Microsoft не пользовались у хакеров большой популярностью и все дружно налегали на soft-ice, но теперь, когда поддержка последнего прекращена и он обречен на медленное, но неотвратимое умирание, возникает вопрос: как дальше жить и чем ломать? тем временем, windbg сильно повзрослел и хотя это кал, по целому ряду характеристик он обгоняет soft-ice. сейчас мыщъх покажет как использовать windbg в качестве API- и RPC-шпиона.

введение

API- и RPC-шпионы входят в активный инструментарий хакера. Эти орудия борьбы должны быть всегда остро заточенными и стоять наготове к десантированию в тыл врага для сбора информации о вызываемых функциях. Зная какие функции (и с какими аргументами) вызывает защита, мы можем ставить точки останова, всплывая отладчиком в непосредственной близости от штаб-квартиры врага. Если же этой информации у нас нет, точки останова приходится вставить вслепую, гадая какой именно API-функцией воспользовалась программа. В частности, одних только функций для чтения текущей даты (необходимой защите для определения периода истечения триала) существует с полдесятка и без шпиона нет никакой возможности угадать какой именно из них воспользовался разработчик.

Тоже самое относится и к RPC-вызовам (Remote Procedure Calls – Удаленные Вызовы Процедур), своеобразном фундаменте множества служб, к числу которых принадлежит и служба печати, да и не только она одна.

В общем, без шпионов ломать становится совсем хреново. Но… soft-ice не предоставляет таких возможностей и приходится использовать сторонние средства, большинство из которых способы шпионить только за честными программами, но столкнулись с защитами начинают мести такую пургу, что прямо метель. К тому же, неудобно каждый раз выходить из отладчика, чтобы запустить очередную утиль. Намного комфортнее держать весь хакерский арсенал в одном месте.

…и этим местом постепенно становиться **Microsoft Windows Debugger**, поддерживающий множество полезных расширений на все случаи жизни и, естественно, позволяющий писать нужные нам расширения самостоятельно.

Преодолевая стойкое убеждение перед продукцией Microsoft, все же скачаем этого зверя и посмотрим на что он способен.

>>> врезка где брать и почем

Microsoft Windows Debugger (далее по тексту просто WinDeb) входит в состав множества продуктов: Platform SDK, DDK, WDF, а так же поставляемся вместе с самостоятельным пакетом "Debugging Tools for Windows", занимающим чуть больше 15 Мбайт, причем версия WinDeb из комплекта "Debugging Tools for Windows" обычно самая свежая и содержащая наибольшее количество всяких полезных расширений.

Скачать ее можно с http://www.microsoft.com/whdc/devtools/debugging (32-битная и 64-битные версии). Microsoft устроила бесплатную раздачу, не требуя даже проверки подлинности копии Windows. Во всяком случае пока… В любом случае, пакет "Debugging Tools for Windows" валяется на многих сайтах, правда далеко не всегда первой свежести.

Рисунок 1 бесплатная раздача Debugging Tools с сайта Microsoft

первое знакомство с WinDbg

Отладчик WinDbg представлен в двух "ипостасях": i386kd.exe (для 64-битной версии — ia64kd.exe) — консольный kernel-debugger (см. рис. 2), отлаживающий _только_ драйвера вместе с дампами памяти ядра и требующий как минимум двух машин, связанных com-шнурком или взаимодействующим через сеть. Если двух машин нет, можно воспользоваться VM Ware, поддерживающий виртуальные сетевые адаптеры и позволяющей загонять com-порт (виртуальный, естественно) в пайп. Несмотря на то, что i386kd.exe – довольно мощная и хорошая штука (подробно описанная у Шрайбера в его "Недокументированных возможностях Windows 2000") для наших хакерских целей она не потребуется.

```
nezumi.bat
f7433678 8a11            mov     dl,[ecx]                ds:0023:6e3c2081=??
Resetting default scope

STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
8054e9e0 65537365 2f3c746e 656d616e 200a0d3e w22n51+0x24678
7479426c 00000000 00000000 00000000 00000000 0x65537365


FOLLOWUP_IP:
w22n51+24678
f7433678 8a11            mov     dl,[ecx]

SYMBOL_STACK_INDEX:  0

FOLLOWUP_NAME:  MachineOwner

SYMBOL_NAME:  w22n51+24678

MODULE_NAME:  w22n51

IMAGE_NAME:  w22n51.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  3ff54d71

STACK_COMMAND:  .trap ffffffff8054e894 ; kb

BUCKET_ID:  0xD1_w22n51+24678

Followup: MachineOwner
---------

kd> d
804e187f  f7 45 70 00 00 02 00 74-0d 83 3d 14 2a 55 80 00  .Ep....t..=.*U..
804e188f  0f 85 69 fe ff ff 83 3d-a0 93 55 80 00 0f 85 5c  ..i....=..U....\
804e189f  fe ff ff 83 3d 20 9e 55-80 00 0f 85 4f fe ff ff  ....= .U....O...
804e18af  b8 ff 00 00 00 eb a8 a1-54 f0 df ff c7 05 54 f0  ........T.....T.
804e18bf  df ff 00 00 00 00 89 45-68 8b e5 e9 8d d7 ff ff  .......Eh.......
804e18cf  90 f7 45 70 00 00 02 00-75 0d f7 45 6c 01 00 00  ..Ep....u..El...
804e18df  00 0f 84 f9 00 00 00 0f-21 c3 0f 21 c9 0f 21 d7  ........!..!..!.
804e18ef  89 5d 18 89 4d 1c 89 7d-20 0f 21 db 0f 21 f1 0f  .]..M..} .!..!..
kd>
```

Рисунок 2 i386kd — консольная версия ядерного отладчика от Microsoft

WinDbg.exe (см. рис.3) – представляет собой типичное GUI-приложение (уже тошнит, да?), довольствующееся одним компьютером и позволяющее отлаживать прикладные приложения, анализировать дампы памяти, шпионить за событиями происходящими системы. А вот для отладки драйверов опять-таки понадобиться второй компьютер, соединенный сетью или шнурком, но мы обойдемся и без шнурка.

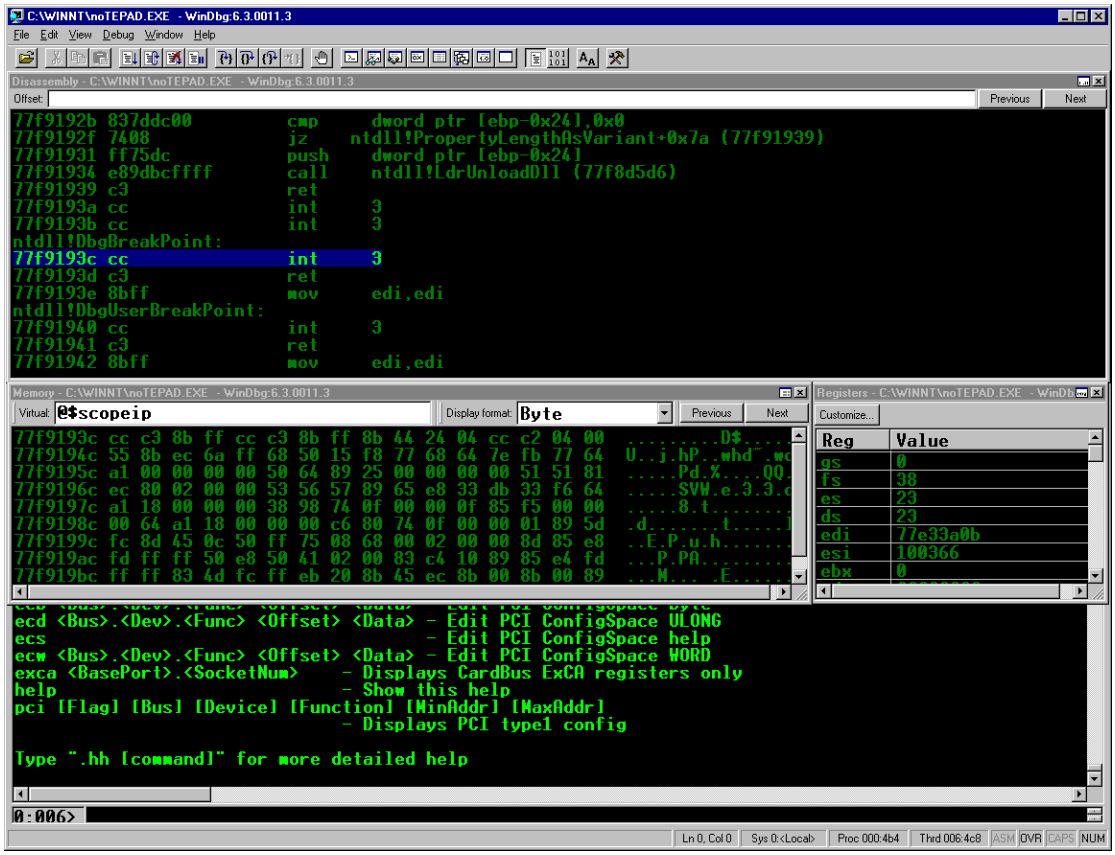

Рисунок 3 графическая версия ядерного/прикладного отладчика от Microsoft

Много полезной информации содержится в справочном файле debugger.chm, который полезно прочитать _до_ запуска отладчика, чтобы не задавать на форумах глупых вопросов и не ломиться в открытые двери. Возможности WinDbg намного больше, чем это кажется на первый взгляд, просто до них через меню и прочие интерфейсные штучки _не_ добраться!

OK, начинаем рыть в глубь. В каталогах w2kchk и w2kfre валяются модули расширения для Windows 2000, конструктивно выполненные в виде динамических библиотек. Содержащие обоих каталогов идентично и разница между ними заключается в том, что "*chk" работает с отладочной версией ядра (checking build), а с "*fre" – с финальной (release).

Посмотрим, что у нас здесь находится:

```
Содержимое папки C:\Program Files\Debugging Tools for Windows\w2kfre

13.08.2005  20:19       <DIR>          .
13.08.2005  20:19       <DIR>          ..
06.02.2004  01:38              105 499 acpikd.dll
06.02.2004  01:38              179 227 gdikdx.dll
06.02.2004  01:38            2 304 541 kdex2x86.dll
06.02.2004  01:38              447 005 kdextx86.dll
06.02.2004  01:38               42 011 ndiskd.dll
06.02.2004  01:38               85 533 ntsdexts.dll
06.02.2004  01:38               78 364 rpcexts.dll
06.02.2004  01:38               24 091 scsikd.dll
06.02.2004  01:38              170 013 userexts.dll
06.02.2004  01:38              302 620 userkdx.dll
06.02.2004  01:38              106 524 vdmexts.dll
               11 файлов      3 845 428 байт
                2 папок     995 536 896 байт свободно
```

Листинг 1 модули расширения для WinDbg, предназначенные для Windows 2000

Назначение большинства модулей можно определить по их названию или на худой конец (не путать конец с мышиным хвостом!) загрузить непонятный модуль в отладчик и вызывать его хелп (чуть позже мы покажем, как это делается). А пока заглянем в каталог winxp, хранящий расширения, специфичные для Windows XP:

```
Содержимое папки C:\Program Files\Debugging Tools for Windows\winxp

13.08.2005  20:19       <DIR>          .
13.08.2005  20:19       <DIR>          ..
21.01.2004  10:43               73 728 acpikd.dll
28.06.2002  15:47                4 841 default.tmf
24.02.2004  13:01              285 696 exts.dll
19.02.2004  18:19            1 096 192 kdexts.dll
21.01.2004  12:30               66 048 minipkd.dll
21.01.2004  12:52               83 968 ndiskd.dll
19.02.2004  18:20               51 200 ntsdexts.dll
21.01.2004  12:25              335 360 rpcexts.dll
21.01.2004  12:30               61 952 scsikd.dll
16.09.2003  19:41               16 467 system.tmf
21.01.2004  13:09              262 144 traceprt.dll
21.01.2004  10:48              154 112 vdmexts.dll
21.01.2004  10:37               12 800 w64cpuex.dll
21.01.2004  10:37                7 168 w64ia32xcpuex.dll
21.01.2004  10:37               14 336 wamd64cpuex.dll
21.01.2004  10:26               50 176 wmitrace.dll
21.01.2004  10:37               77 312 wow64exts.dll
              17 файлов      2 653 500 байт
               2 папок     995 532 800 байт свободно
```

Листинг 2 модули расширения для WinDbg, предназначенные для Windows XP

Как видно, набор расширений для XP намного богаче, но мыщъх все равно не изменит своей любимой Windows 2000, под которой сидел, сидит и будет сидеть, а когда же она окончательно одряхлеет, мигрирует на FreeBSD, тем более, что расширения, ответственные за шпионах, хранятся в каталоге winext общим для всех систем.

```
Содержимое папки C:\Program Files\Debugging Tools for Windows\winext

13.08.2005  20:19       <DIR>          .
13.08.2005  20:19       <DIR>          ..
24.02.2004  13:01              612 864 ext.dll
24.02.2004  13:01              174 592 kext.dll
19.02.2004  18:44              228 864 logexts.dll
13.08.2005  20:19       <DIR>          manifest
24.02.2004  13:01               53 760 uext.dll
21.01.2004  12:28               22 528 wdfkd.dll
               5 файлов      1 092 608 байт
               3 папок     995 524 608 байт свободно
```

Листинг 3 модули расширения для WinDbg, предназначенные для осей всех типов

Файл с ничего не говорящим именем logexts.dll — это и есть шпионский компонент (ну для шпионов маскироваться — вполне нормально и ничего удивительно тут нет).

Теперь что касается самого WinDeb. Раскладку окон, цвет и гарнитуру шрифтов каждый может настроить под свой вкус, поскольку в конфигурации по умолчанию с отладчиком долго работать невозможно, иначе глаза опустятся куда-то в район хвоста, а для хакеров глаза — второй инструмент после серого мозгового вещества!

Так что первое, что нужно сделать с WinDeb это настроить его под свой _собственный_ вкус (а вкусы, как известно, у всех разные).

техника API-шпионажа

В состав Platform SDK изначально входило какое-то подобие API-шпиона (точнее, _пародия_ на API-шпиона), расположенного в \Microsoft Platform SDK\Bin\WinNT\Apimon.Exe, однако он выдавал только общую статистку по API-вызовам без учета их хронологии (см. рис. 4), и постоянно падал при попытке загрузить в него что-то более сложнее чем notepad (см. рис. 5). Короче, для взлома Apimon.exe не годился. Однозначно!

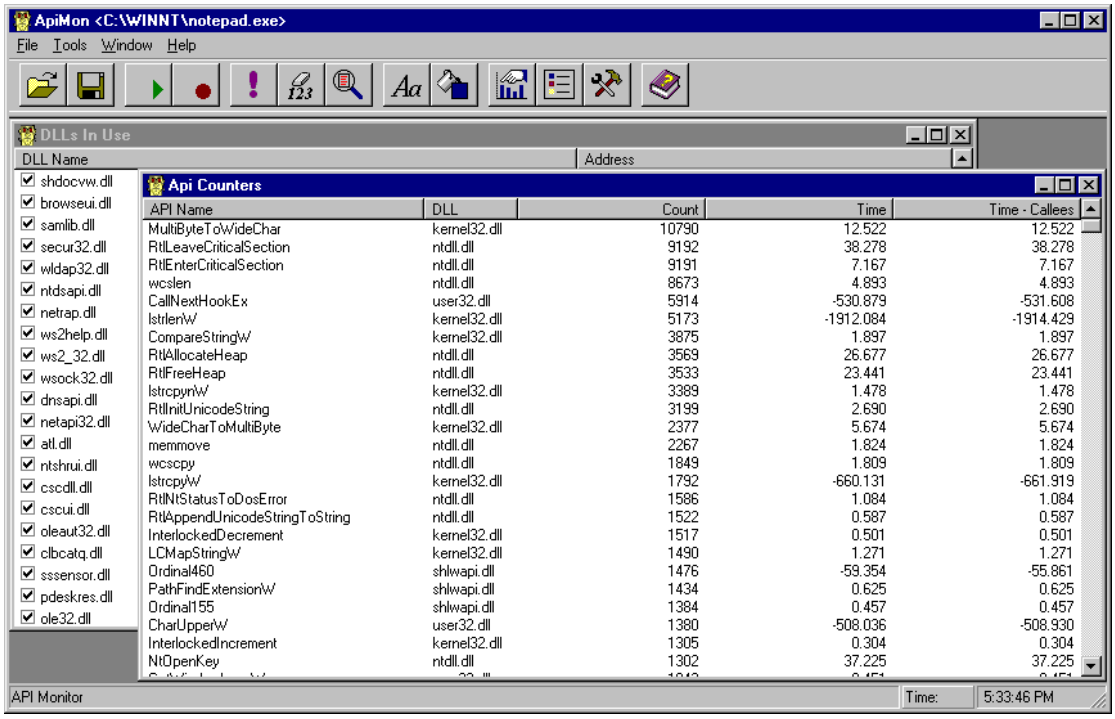

Рисунок 4 отчет ApiMon'a о количестве вызовов API-функций, совершенных подопытной программой

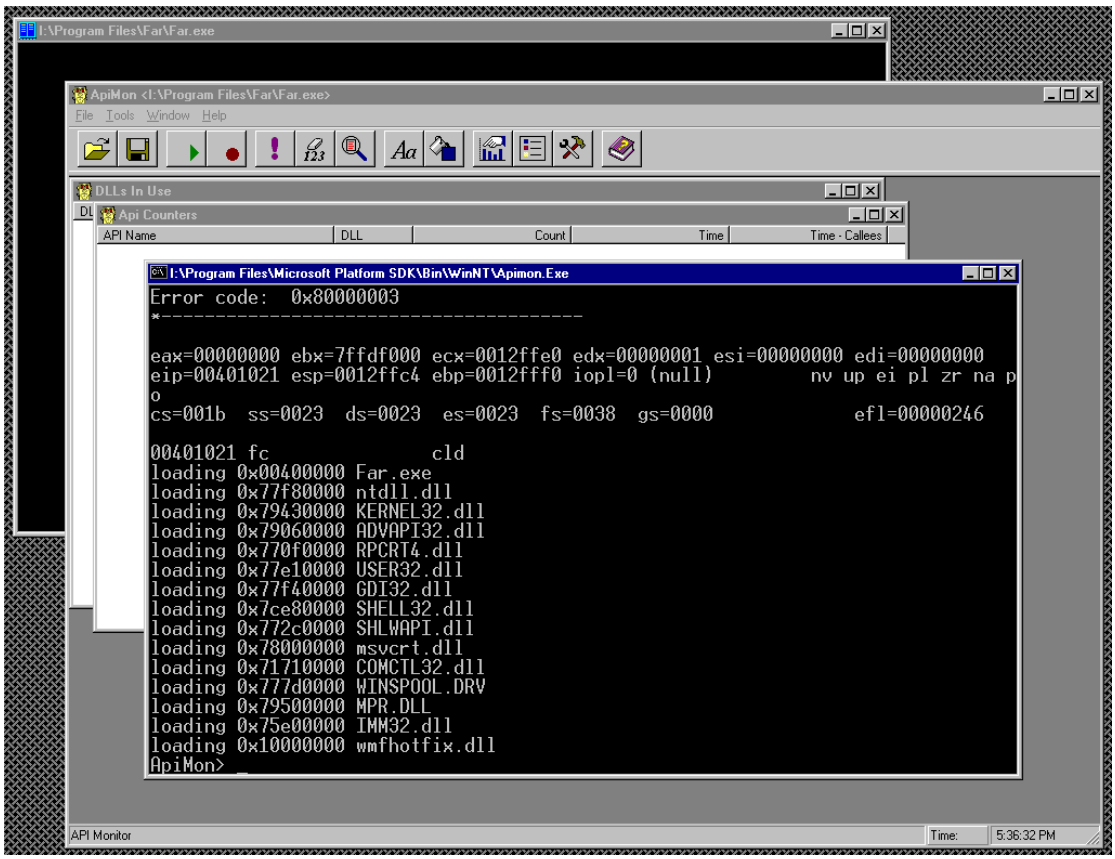

Рисунок 5 крах ApiMon'a при попытке скормить ему FAR, упакованный ASPack'ом

Посмотрим на Windbg – что же в нем изменилось? Забегая вперед, скажем — абсолютно все! Microsoft предоставила нам сложный и могущественный инструмент,

способный решать широкий круг задач и противостоять различным антиотладочным приемам, которыми понапичканы современные защитные механизмы.

Короче, будем считать, что мыщъх вас соблазнил. Приступаем к экспериментом (не путать с экскрементами!) Первым делом загружаем в отладчик файл, за которым мы будем шпионить, например, все тот же Блокнот. Делается это так: file → open executable → notepad.exe или, как вариант, <CTRL-E>, notepad.exe. На крайняк файл можно загрузить из командной строки. Только не нажимайте раскрывающуюся папку на панели инструментов — это все равно не поможет, т. к. она предназначена для работы с исходными текстами и их окружением, которых в нашем распоряжении естественно нет (если бы они были — стали бы мы тут извращаться со шпионажем?!).

```
CommandLine: C:\WINNT\NOTEPAD.EXE
Symbol search path is: *** Invalid ***
****************************************************************************
* Symbol loading may be unreliable without a symbol search path.           *
* Use .symfix to have the debugger choose a symbol path.                   *
* After setting your symbol path, use .reload to refresh symbol locations. *
****************************************************************************
Executable search path is:
ModLoad: 01000000 01010000   notepad.exe
ModLoad: 77f80000 77ffd000   ntdll.dll
ModLoad: 76ae0000 76b1e000   C:\WINNT\system32\comdlg32.dll
ModLoad: 772c0000 77326000   C:\WINNT\system32\SHLWAPI.DLL
ModLoad: 79060000 790c5000   C:\WINNT\system32\ADVAPI32.dll
ModLoad: 79430000 794e4000   C:\WINNT\system32\KERNEL32.dll
ModLoad: 770f0000 77168000   C:\WINNT\system32\RPCRT4.dll
ModLoad: 77f40000 77f7f000   C:\WINNT\system32\GDI32.dll
ModLoad: 77e10000 77e79000   C:\WINNT\system32\USER32.dll
ModLoad: 78000000 78045000   C:\WINNT\system32\msvcrt.dll
ModLoad: 71710000 71794000   C:\WINNT\system32\COMCTL32.DLL
ModLoad: 7ce80000 7d0c6000   C:\WINNT\system32\SHELL32.DLL
ModLoad: 777d0000 777ee000   C:\WINNT\system32\WINSPOOL.DRV
ModLoad: 79500000 79511000   C:\WINNT\system32\MPR.DLL
(510.508): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00071f04 ecx=00000009 edx=00000000 esi=7ffdf000 edi=00071f70
eip=77f9193c esp=0006f984 ebp=0006fc98 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll
-
ntdll!DbgBreakPoint:
77f9193c cc              int     3
```

Листинг 4 неверное определение точки входа при загрузке исполняемого файла в WinDbg

Отладчик послушно загружает файл, отображая все динамические библиотеки, перечисленные в таблице импорта, отображает содержимое регистров и в EntryPoint устанавливаем точку останова. На самом деле, это мы думаем, что отладчик установил точку останова в EntryPoint, а в действительности… все обстоит совсем не так! Судя по адресу 77F9193Ch, лежащему глубоко внутри NTDLL.DLL, это не совсем EntryPoint, точнее _совсем_ _не_ EntryPoint, а native-API функция DbgBreakPoint, которую можно трассировать до конца света, но так ни к чему и не придти. Приходится выкручиваться и применять всякие недетские извращения (впрочем, для хакеров они вполне типичны).

Загружаем notepad.exe в наш любимый hiew.exe (или любой другой hex-редактор по вкусу), жмем на <ENTER> для перехода в hex-режим, давим <F8> для отображения заголовка и переходим в истинную точку входа по <F5>, адрес которой (в моем случае равный 1006420h) высвечивается в левом верхнем углу экрана (см. рис. 6).

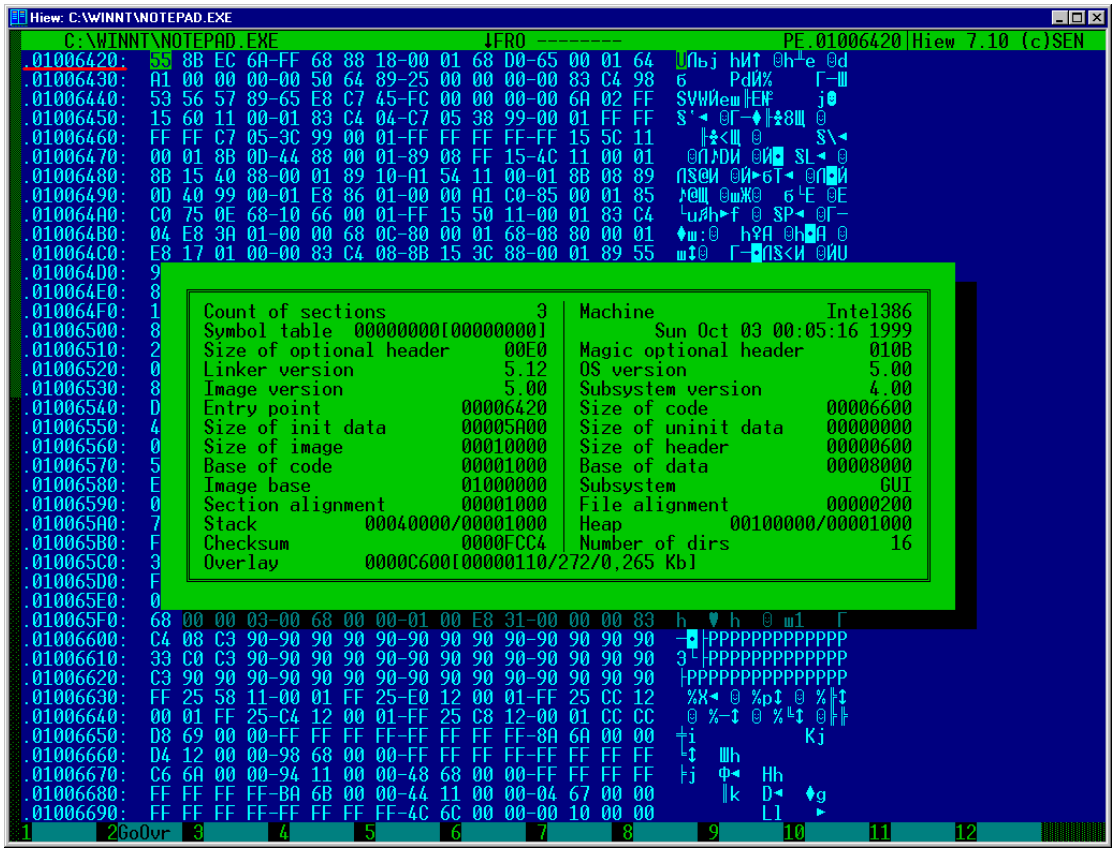

Рисунок 6 определение подлинной точки входа с помощью hiew'a

Возвращаемся в WinDbg и, находясь в окне команд (такая строчка с деловито мерцающим курсором), пишем "BP 1006420", устанавливая точку останова, после чего жмем <F5> (или даем команду "g" – в смысле "goto", продолжение выполнения) и ждем развития событий. А события ждать себя не заставляют (см. листинг 5):

```
0:000> BP 1006420
0:000> g
ModLoad: 75e00000 75e1a000   C:\WINNT\system32\IMM32.DLL
ModLoad: 10000000 10005000   C:\WINNT\system32\wmfhotfix.dll
Breakpoint 0 hit
eax=00000000 ebx=7ffdf000 ecx=00010101 edx=ffffffff esi=0009a0f0 edi=017af640
eip=01006420 esp=0006ffc4 ebp=0006fff0 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000246
notepad+0x6420:
01006420 55              push    ebp
```

Листинг 5 ручная установка бряка на истинную точку входа

Отладчик подгружаем еще две динамические библиотеки (одна из которых — wmfhotfix.dll – заплатка от Ильфака), радостно сообщает, что "Breakpoint 0 hit" (сработала точка останова), выводит значения регистров (ну куда же без них) и первую машинную команду, стоящую в точке входа — push ebp.

Команда "u" позволит дизассемблировать остальные команды, следующие за ней, помогая убедиться, что мы действительно находимся там, где нужно:

```
0:000> u
notepad+0x6420:
01006420 55              push    ebp
01006421 8bec            mov     ebp,esp
01006423 6aff            push    0xff
01006425 6888180001      push    0x1001888
0100642a 68d0650001      push    0x10065d0
0100642f 64a100000000    mov     eax,fs:[00000000]
01006435 50              push    eax
01006436 64892500000000  mov     fs:[00000000],esp
```

Листинг 6 дизассемблирование окрестностей точки входа

Теперь, когда все подготовительные мероприятия завершены, необходимо подключить модуль расширения. Для этого даем команду "**!load <*name*>**", где "name" имя модуля расширения без ".dll" (в данном случае — "logexts") и полностью вся команда выглядит так:

```
0:000> !load logexts
```

Листинг 7 загрузка модуля расширения, ответственного за шпионаж API-функций

Отладчик проглатывает ее как ни в чем не бывало и создается впечатление, что ничего не происходит. Но это не так! Чтобы убедиться, что модуль расширения успешно загружен, вызовем его локальную справку, набрав команду "!logexts.help":

```
0:000> !logexts.help

Windows API Logging Extensions  v2.09

Main control:
  !loge [dir]                 Enable logging. Output directory optional.
  !logi [dir]                 Initialize but don't enable logging.
  !logd                       Disable logging.

Output:
  !logo                       List output settings.
  !logo [e|d] [d|t|v]         Enable/disable output:
                                d - Debugger
                                t - Text file
                                v - Verbose log

Categories:
  !logc                       List all categories.
  !logc p #                   List APIs in category #.
  !logc [e|d] *               Enable/disable all categories.
  !logc [e|d] # [#] [#] ...   Enable/disable category #.

Buffer access:
  !logb p                     Print buffer contents to debugger.
  !logb f                     Flush buffer to log files.

Module inclusion/exclusion:
  !logm                       Display module inclusion/exclusion list.
  !logm [i|x] [DLL] [DLL] ... Specify module inclusion/exclusion list.
```

Листинг 8 выдача справки по "шпионским" командам

Ого! Сколько тут всего, хорошего и разного! С одного захода это даже не разгрызть, но мы все-таки попробуем. Команда "!loge [dir]" активирует шпионаж за API-функциями, при желании позволяя указать каталог, в котором будет автоматически создана поддиректория LogExts для хранения логов. Логи могут писаться как в текстовом, там и в двоичном формате (формат вывода задается командой "!logo"), причем имя лога соответствует имени исполняемого файла за которым он шпионил (например, "noTEPAD.EXE.txt" — да-да, вот именно в таком регистре он и записывается).

Вызванная без параметров, команда "!logo" выводит текущий формат лога. Чтобы включать текстовой формат, необходимо набрать "!logo e t", а чтобы включить все три параметра, необходимо трижды вызывать "!logo" с разными ключами. К сожалению, конструкцию "!logo e dtv" отладчик не переваривает. Редиска!

Для сокращения размеров лога и выкидывания заведомо ненужной информации, WinDbg поддерживает категории API-вызовов, список которых можно вывести на экран командой "!logc":

```
0:000> !logc
Categories:

  1 AdvApi32                      Enabled
  2 AtomFunctions                 Enabled
  3 AVIFileExports                Enabled
  4 Clipboard                     Enabled
  5 ComponentObjectModel          Enabled
  6 DebuggingAndErrorHandling     Enabled
  7 DeviceFunctions               Enabled
```

```
  8 Direct3D                        Enabled
  9 DirectDraw                      Enabled
 10 DirectPlay                      Enabled
 11 DirectSound                     Enabled
 12 GDI                             Enabled
 13 HandleAndObjectFunctions        Enabled
 14 HookingFunctions                Enabled
 15 IOFunctions                     Enabled
 16 MemoryManagementFunctions       Enabled
 17 Multimedia                      Enabled
 18 Printing                        Enabled
 19 ProcessesAndThreads             Enabled
 20 RegistryFunctions               Enabled
 21 Shell                           Enabled
 22 StringManipulation              Enabled
 23 ThreadLocalStorage              Enabled
 24 User32                          Enabled
 25 User32StringExports             Enabled
 26 Version                         Enabled
 27 WinSock2                        Enabled
```

Листинг 9 просмотр категорий API-функций

Как видно, всего имеется 27 категорий и для просмотра функций, входящих в каждую из категорий можно воспользоваться командой "!logic p #", где "#" – номер категории, например, 16 — MemoryManagementFunctions.

```
0:000> !logc p 16
MemoryManagementFunctions:
  AllocateUserPhysicalPages       KERNEL32.DLL
  FreeUserPhysicalPages           KERNEL32.DLL
  GetProcessHeap                  KERNEL32.DLL
  GetProcessHeaps                 KERNEL32.DLL
…
  OpenFileMappingA                KERNEL32.DLL
  OpenFileMappingW                KERNEL32.DLL
  UnmapViewOfFile                 KERNEL32.DLL
```

Листинг 10 просмотр имен API-функций, входящих в категорию MemoryManagementFunctions (для экономии места приведен только фрагмент)

Для шпионажа за определенными категориями функций даем команду: "!logc e # # #", где "e" – включить ("enable") шпионаж, а "#" – перечень категорий. Ключ "d" (от "disable"), соответственно, означает исключить данную категорию (категории) API-функций из круга подозреваемый и не шпионить за ними. Команда "!logc e *" включает все категории (и это основной режим шпиона в котором гоняют его хакеры при первом знакомстве с ломаемой программой).

При желании можно указать перечень динамических библиотек за которыми (не)следует следить. Зачастую это намного проще, чем возиться с категориями. Отображением списка текущий "поднадзорных" библиотек занимается команда "!logm":

```
0:000> !logm
Included modules:
  USER32.DLL
  GDI32.DLL
  ADVAPI32.DLL
```

Листинг 11 просмотр списка динамических библиотек за которыми осуществляется шпионаж

Просматривая этот список, мы с удивлением обнаруживаем в нем отсутствие KERNEL32.DLL – базовой ядерной библиотеки, содержащей максимум интересующих нас функций. Попытка включить ее в список командой "!logm i KERNEL32.DLL" приводит к появлению многоэтажного матерного ругательства: мол, KERNEL32.DLL обязательно должна быть исключена и включению не подлежит:

```
0:000> !logm i KERNEL32.DLL
KERNEL32.DLL is mandatory for exclusion so it can't be included.
Included modules:
```

Листинг 12 категорический отказ отладчика шпионить за KERNEL32.DLL

На самом деле, стоит только нажать <F5>, как в логе (параллельно выводимом на экран и в файл) появится перехваченные имена API-функций, принадлежащих KERNEL32.DLL:

```
Thrd 4c4 010029BD GetProcAddress( NULL "RegisterPenApp") -> NULL [FAIL]
Thrd 4c4 77E202F2 LoadLibraryExW("INDICDLL.dll" NULL ALTERED_SRCH_PATH) -> 0x6E380000
Thrd 4c4 77106CF2 GetProcAddress( 0x77F80000 "NtQuerySystemInformation") -> 0x77F889DC
Thrd 4c4 77106D0D GetProcAddress( 0x77F80000 "NtOpenFile") -> 0x77F886AC
Thrd 4c4 77106D1A GetProcAddress( 0x77F80000 "RtlInitUnicodeString") -> 0x77FABE9C
Thrd 4c4 77E202F2 LoadLibraryExW("PDSHELL.DLL" NULL ALTERED_SRCH_PATH) -> 0x00F30000
Thrd 4c4 77E202F2 LoadLibraryExW("SSSensor.dll" NULL ALTERED_SRCH_PATH) -> 0x013B0000
Thrd 4c4 76AE1DAB GetProcAddress( 0x79430000 "GetUserDefaultUILanguage") -> 0x7947106B
Thrd 4c4 7CEAAF39 GetProcAddress( 0x77E10000 "GetSystemMetrics") -> 0x77E33277
Thrd 4c4 7CEAAF4A GetProcAddress( 0x77E10000 "MonitorFromWindow") -> 0x77E2920B
Thrd 4c4 7CEAAF5B GetProcAddress( 0x77E10000 "MonitorFromRect") -> 0x77E20D54
Thrd 4c4 7CEAAF6C GetProcAddress( 0x77E10000 "MonitorFromPoint") -> 0x77E2A0F2
Thrd 4c4 7CEAAF7D GetProcAddress( 0x77E10000 "EnumDisplayMonitors") -> 0x77E1F61D
Thrd 4c4 7CEAAF8E GetProcAddress( 0x77E10000 "EnumDisplayDevicesW") -> 0x77E18A08
Thrd 4c4 7CEAAFAE GetProcAddress( 0x77E10000 "GetMonitorInfoW") -> 0x77E2A07E
```

Листинг 13 фрагмент "шпионского" протокола, содержащего всю необходимую нам инфу

В наше распоряжение попадают номера потоков (thrd), адреса вызова API-функций вместе с передаваемыми ими аргументами. По приведенному выше фрагменту лога видно, что вызвав функцию GetProcAddress(NULL "RegisterPenApp") по адресу 010029BDh, "Блокнот" погрузился в пучину системных библиотек, лежащих далеко за пределами принадлежащей ему области адресов. Но это не главное. Главное то, что шпион от Microsoft работает и успешно шпионит, практически ни в чем не уступая большинству своих конкурентов, а кое в чем их даже и обгоняя!

техника RPC-шпионажа

RPC-шпионаж осуществляется во всем аналогично API-шпионажу (ну, практически аналогично), только вместо logexts используется расширение rpcexts, загружаемое командой "!load rpcexts" и выдающее справку по своим ключам командой "!rpcexts.help".

Ключи же это настолько обширны, что требуют для своего описания целой статьи... или все-таки не требуют? В большинстве случаев встроенного хелпа вполне достаточно!

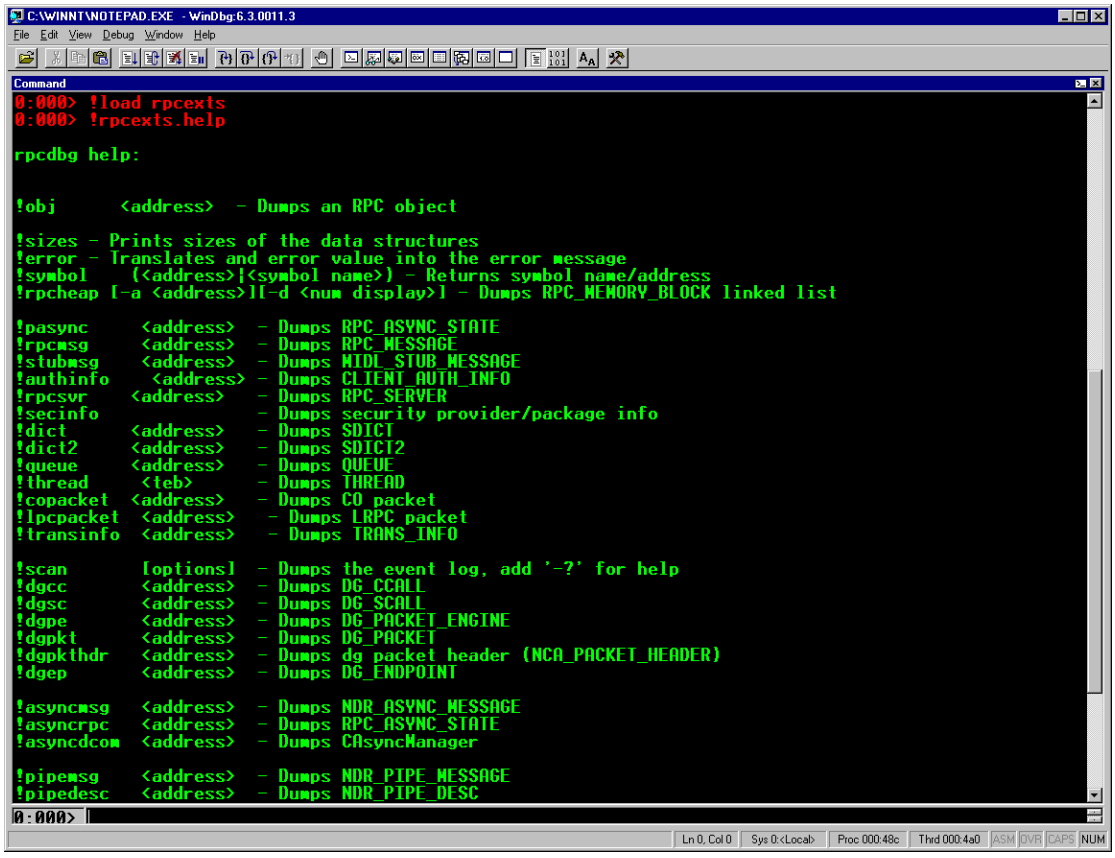

Рисунок 7 загрузка расширения rpcexts, ответственного за RPC-шпионаж (и не только шпионаж!) и выдача справки по нему

заключение

Мы рассмотрели всего лишь два расширения отладчика WinDbg из… очень многих! Ну так чего же мы сидим? Чего ждем?! Загружаем все расширения одно за другим, даем команду "!name.help", смотрим, курим, читаем, втыкаем, после чего экспериментируем постигая все новые границы и мысленно сравнивая возможности WinDbg с soft-ice.

Но, несмотря ни на что, soft-ice все-таки жало. Хороший был отладчик...