Tracer: руководство пользователя

Денис Юричев
<dennis@yurichev.com>

Дата компиляции этой PDF: 28 января 2014 г..
Англоязычная версия текста (а также сам tracer) также доступна по ссылке
http://yurichev.com/tracer-ru.html

Оглавление

Введение iii

Посвящение iv

Благодарности v

1 Общие опции 1

2 Как в tracer задается адрес 3

3 BPF: установка прерывания на исполнение функции 4
- 3.1 Опция TRACE . . . 6
- 3.2 Примеры . . . 8
  - 3.2.1 Простое использование . . . 8
  - 3.2.2 Перехват некоторых Windows-функций для работы с реестром . . . 8
  - 3.2.3 Подавить шумный сигнал . . . 8
  - 3.2.4 Подавить диалоговое окно с сообщением . . . 8
  - 3.2.5 Перехват вызовов rand() . . . 8
  - 3.2.6 FreeCell . . . 9
  - 3.2.7 Проверка ивентов и запись в лог в Oracle RDBMS . . . 9
  - 3.2.8 Слежение за выделением памяти в Oracle 11.1.0.6.0 win32/win64 . . . 10
  - 3.2.9 Слежение за разбором SQL-выражений в Oracle RDBMS . . . 10
  - 3.2.10 Игнорирование неподписанных драйверов . . . 11
  - 3.2.11 Вывод памяти по аргументам функций . . . 11
  - 3.2.12 Вывод памяти по аргументам функций и слежение за её изменением . . . 11
- 3.3 Примеры опции TRACE . . . 12
  - 3.3.1 Трассировка строковых функций . . . 12
  - 3.3.2 Трассируем quicksort() . . . 12

4 BPX: установка прерывания на произвольное место 15
- 4.1 Примеры . . . 15
  - 4.1.1 Task Manager: создать иллюзию что у нас 32 или 64 процессора . . . 15
  - 4.1.2 Перехват развернутой (inline) функции strcmp() . . . 15
  - 4.1.3 Изменение флагов перед тем как условный переход будет совершен . . . 16
  - 4.1.4 Шутка в Microsoft Excel . . . 17

5 BPM: установка прерывания на обращение к ячейке памяти 18
- 5.1 Примеры . . . 18
  - 5.1.1 Слежение за обращением к переменным в Oracle RDBMS . . . 18
  - 5.1.2 Проверяет ли программа целостность своего кода? . . . 19

6 Одноразовое прерывание по INT3 20

7 Взаимодействие во время работы 22

8 Отсоединение от процесса **23**

9 Некоторые технические заметки **24**

10 Известные проблемы **25**
10.1 Windows 2000 25

11 Заключение **26**

Введение

Tracer это win32-отладчик командной строки для выполнения простых отладочных задач.

Главные возможности:

- Установка прерывания выполнении функции, вывод аргументов функции и результата.
- Трассировка каждой инструкции функции и сохранение значений регистров.
- Установка прерывания в любом месте, вывод состояния регистров процессора и возможность изменить их.
- Установка прерывания на обращение к любой ячейки памяти и перехват всех обращений к ней.

Второстепенные возможности:

- Установка прерывания задавая адрес, имя символа или байтмаски.
- Обнаружение юникодных строк в аргументах функций.
- Поддержка как Windows x86 так и Windows x64.
- Поддержка символов Oracle RDBMS .SYM.
- Наличие исходных кодов.

Посвящение

Опции `BPX`, `BPMB/BPMW/BPMD` названы так же как и в SoftICE, великолепном отладчике прошлого.

Благодарности

Alex Ionescu.

Глава 1

Общие опции

`-l:<fname.exe>`: загрузка процесса.

`-c:<cmd_line>`: задание командной строки для загружаемого процесса.

Например:

```
tracer.exe -l:bzip2.exe -c:--help
```

Если командная строка содержит пробелы:

```
tracer.exe -l:rar.exe "-c:a archive.rar *"
```

`-a:<fname.exe or PID>`: присоединение к запущенному процессу по его имени или PID.

Процесс с таким именем должен быть загружен. Если процессов с таким имененем загружено несколько, tracer присоеденяется ко всем сразу одновременно.

`--loading`: вывод имен файлов и базовых адресов для всех загружаемых модулей (обычно, это DLL-файлы).

`--child`: присоеденяться в том числе и к процессам порождаемых главным процессом.

Например, вы можете запустить `tracer.exe --child -l:cmd.exe`, откроется консольное окно cmd.exe и tracer будет присоеденятся к каждому процессу запущенному внутри командного интерпретатора.

`--allsymbols[:<regexp>]`: вывод всех символов в процессе загрузки по регулярному выражению:

`--allsymbols:somedll.dll!.*` опция может быть использована для вывода всех символов в некоторой DLL.

`--allsymbols:.*printf` выведет что-то вроде:

```
New symbol. Module=[ntdll.dll], address=[0x77C004BC], name=[_snprintf]
New symbol. Module=[ntdll.dll], address=[0x77B8E61F], name=[_snwprintf]
...
New symbol. Module=[msvcrt.dll], address=[0x75725F37], name=[vswprintf]
New symbol. Module=[msvcrt.dll], address=[0x75726649], name=[vwprintf]
New symbol. Module=[msvcrt.dll], address=[0x756C3D68], name=[wprintf]
```

`-s`: вывод стека вызовов перед каждым прерыванием.

Например:

`tracer.exe -l:hello.exe -s bpf=kernel32.dll!WriteFile,args:5`

Мы увидим:

```
23B4 (0) KERNEL32.dll!WriteFile (7, "hello to tracer!\r\n", 0x0000000E, 0x0017E3A4, 0) (
    called from 0x7317754E (MSVCR90.dll!_lseeki64+0x56b))
Call stack of thread 0x23B4
return address=731778D8 (MSVCR90.dll!_write+0x9f)
return address=7313FB4A (MSVCR90.dll!_fdopen+0x1c0)
return address=7313F70C (MSVCR90.dll!_flsbuf+0x6e1)
return address=73141E50 (MSVCR90.dll!printf+0x84)
return address=0040100E (hello.exe!BASE+0x100e)
return address=0040116F (hello.exe!BASE+0x116f)
```

```
return address=76FCE4A5 (KERNEL32.dll!BaseThreadInitThunk+0xe)
return address=77C9CFED (ntdll.dll!RtlCreateUserProcess+0x8c)
return address=77C9D1FF (ntdll.dll!RtlCreateProcessParameters+0x4e)
23B4 (0) KERNEL32.dll!WriteFile -> 1
```

Вывод стека вызова очень удобен, например, мы имеем программу показывающую окно с сообщением и перехватывая вызов `USER32.DLL!MessageBoxA` мы можем увидеть путь к этому вызову.

Возможность вывода стека доступна для всех типов прерываний BPF/BPX/BPM.

Замечание: эта возможность пока не очень хорошо работает в x64.

Если указана опция *--dump-fpu*, состояние регистров FPU будут показываться.

Если указана опция *--dump-xmm*, состояние всех регистров XMM также будут выводиться, если только регистр не пуст.

Если указана опция *--dump-seh*, вся доступна информация о SEH будет выведена. Для отображения информации SEH4, нужен доступ к переменной `security_cookie`, tracer будет искать её по имени в файле `.MAP` или `.PDB`.

`-t`: записывать дату и время перед каждой строкой в лог:

`--version`: вывести номер текущей версии и дату/время компиляции, а также проверить наличие новой версии доступной для скачивания.

Например:

```
tracer.exe -l:bzip2.exe bpf=cygwin1.dll!fprintf,args:2 -t
```

```
[2013-07-03 07:15:10:056] TID=13056|(0) cygwin1.dll!fprintf (0x611887b0, "%s: For help,
    type: ‘%s --help’.\n") (called from bzip2.exe!OEP+0x15f1 (0x4025f1))
[2013-07-03 07:15:10:058] TID=13056|(0) cygwin1.dll!fprintf () -> 0x27
```

Эта возможность полезна тогда, когда нужно записывать в журнал время каких-то событий, например, когда именно некая программа обращается к сети.

`--help`: помощь.

`-q`: запрет любого вывода в консоль и лог.

`@`: опция позволяет сохранить все опции в текстовом файле и использовать их многократно:

```
tracer.exe @filename
```

Каждая строка файла представляет опцию. Это очень удобно для длинных и/или часто используемых опций, как байтмаски (смотрите ниже).

Опция `@` также может использоваться с любыми другими опциями:

```
tracer.exe -l:filename.exe @additional_options @even_more_options
```

Глава 2

Как в tracer задается адрес

Имеется три возможности задать адрес прерывания.

- Используя шестнадцатиричный адрес: `0x00400000` — так задается абсолютный адрес внутри win32-процесса. Обратите внимание, что изменение базы загрузки PE-модуля не учитывается, так что, если, например, в IDA, или ином дизассемблере, вы видите один адрес, то этот код все же може быть загружен по другим адресам в памяти процесса (вы можете использовать опцию `--loading`, чтобы увидеть, по каким базовым адресам загружаются модули).

  А для того, чтобы указать некий адрес в определенном PE-модуле, адрес должен быть задан так: `module.dll!0x400000` — и это адрес автоматически подкорректируется, если модуль будет загружен по другому базовому адресу.

- Используя символ.

  Например: `kernel32.dll!writefile`

  Здесь можно использовать регулярные выражения. Например: `.*!printf</i>`: tracer будет искать символ `printf` в каждом загружаемом модуле. Если этот символ имеется в разных модулях, tracer будет использовать только из того модуля, который был загружен раньше всех.

  Для регулярных выражений используется синтаксис POSIX Extended Regular Expression (ERE).

  Из-за того что здесь задается регулярное выражение, некоторые символы, такие как `?`, `.` нужно *escape-ть*. Например, чтобы задать адрес `?method@class@@QAEHXZ`, нужно указывать `\?method@class@@QAE`

  Смещение также можно использовать. Например: `file.exe!BASE+0x1234` (BASE это предопределенный символ, он равен базовому адресу PE-модуля) либо `file.exe!label+0xa`.

Глава 3

BPF: установка прерывания на исполнение функции

Опция BPF, в каком-то смысле, похожа на работу утилиты strace[1].

Главные отличия от strace:

- tracer работает только в win32/win64.
- Прерыванием может быть любоая функция а не только системные вызовы.
- Только 4 прерывания из-за ограничений архитектуры x86.

BPF с адресом но без дополнительных опций будет только показывать момент вызова функции и то что она возвращает.

Например:

```
tracer.exe -l:bzip2.exe bpf=kernel32.dll!WriteFile
```

```
1188 (0) KERNEL32.dll!WriteFile () (called from 0x610AC912 (cygwin1.dll!sigemptyset+0x1022
    ))
1188 (0) KERNEL32.dll!WriteFile -> 1
```

Замечание: tracer не знает о том что функция может иметь тип *void* (т.е., не возвращает ничего). Таким образом, tracer выводит просто то что находится в регистре `EAX`/`RAX` на момент выхода из функции.

Опции:

`ARGS:<number>`: определить количество агрументов для перехватываемой функции.

Например:

```
tracer.exe -l:bzip2.exe -c:--help bpf=kernel32.dll!WriteFile,args:5
```

```
09D0 (0) KERNEL32.dll!WriteFile (0x0000001B, "   If no file names are given, bzip2
    compresses or decompresses", 0x0000003F, "?", 0)
09D0 (0) KERNEL32.dll!WriteFile -> 1
09D0 (0) KERNEL32.dll!WriteFile (0x0000001B, "   from standard input to standard output.
    You can combinesses", 0x0000003B, ";", 0)
09D0 (0) KERNEL32.dll!WriteFile -> 1
09D0 (0) KERNEL32.dll!WriteFile (0x0000001B, "   short flags, so '-v -4' means the same as
     -v4 or -4v, &amp;amp;c.ses", 0x0000003C, "<", 0)
09D0 (0) KERNEL32.dll!WriteFile -> 1
```

[1] http://en.wikipedia.org/wiki/Strace

То что мы видим это попытку вывести 5 агрументов функции при каждом вызове функции WriteFile(). Если аргумент является указателем в пределах памяти процесса и то на что он указывает может быть интерпретировано как ASCII-строка, она будет выведена. Это очень удобно для перехвата строковых функций таких как strcmp(), strlen(), strtok(), atoi(), итд.

Ошибится в количестве аргументов не страшно (кроме случая использования опции `skip_stdcall`, смотрите ниже). Если указанное количество аргументов больше чем на самом деле, возможно, значения из локальных переменных вызывающей функции будут выведены. Или какой-нибудь случайный мусор. Если заданное количество аргументов меньше чем на самом деле, только часть аргументов будет выведена.

`RT:<number>`: подставить другое возвращаемое значение в момент выхода из функции, на лету.

```
tracer.exe -l:filename.exe bpf=function,args:1,rt:0x12345678
```

tracer запишет это значение в регистр `EAX`/`RAX` в момент выхода из функции.

`SKIP`: пропустить выполнение функции. Эта опция может использоваться вместе с опцией `RT`.

```
tracer.exe -l:filename.exe bpf=function,args:1,rt:0x12345678,skip
```

Это означает что в момент начала выполнения функции, управление сразу будет передано на выход и возвращаемое значение будет установлено в 0x12345678.

Замечание: без префикса "0x это значение будет интерпретироваться как десятичное число.

`SKIP_STDCALL`: то же что и `SKIP`, только для stdcall-функций.

Разница между типами функций cdecl и stdcall в том что функция типа cdecl на выходе не выравнивает указатель стека (вызывающая функция должна сделать это). Функция типа stdcall выравнивает указатель стека. cdecl это наиболее используемый тип функций. Хотя, stdcall используется в MS Windows. Так что, если вы хотите пропустить выполнение какой-либо функции в KERNEL32.DLL или USER32.DLL, вы должны использовать `skip_stdcall`. Следовательно, в этом случае, tracer должен знать точное количество аргументов, а без этого процесс может упасть.[2]

Если вы хотите подавить все вызовы функции WriteFile:

```
tracer.exe -l:hello.exe bpf=kernel32.dll!WriteFile,args:5,skip_stdcall,rt:1
```

Не забывайте возвращать 1, для того чтобы вызываемая функция не заподозрила ничего! Количество аргументов функции WriteFile — 5. Поменяйте это значение на что-то другое и процесс упадет.

Замечание: тип функции stdcall отсутствует в Windows x64, так что эта опция отсутствует в 64-битной версии tracer.

`UNICODE`: трактовать строки в аргументах как юникодные (два байта на каждый символ). Это может быть полезно для перехвата win32-функций с суффиксом W, например, MessageBoxW.

К сожалению, tracer умеет автоматически выявлять только строки использующие первую половину таблицы ASCII, так что строки на других языках кроме тех что используют латиницу не будут выявлены автоматически.

`DUMP_ARGS:<size>`: дампить память по аргументам функции (если она чиается) с ограничением size.

Если аргумент функции содержит указатель на читаемый блок памяти, он будет выведен.

На момент выхода из функции, если блок в памяти изменился, то разница будет выведена также.

Например:

```
tracer64.exe -l:test_getlocaltime.exe bpf=.*!getlocaltime,args:1,dump_args:0x30
```

```
TID=6660|(0) KERNEL32.dll!GetLocalTime (0x12ff00) (called from 0x14000100f (getlocaltime.
    exe!BASE+0x100f))
Dump of buffer at argument 1 (starting at 1)
000000000012FF00: 28 FF 12 00 00 00 00 00-00 00 00 00 00 00 00 00 "(..............."
000000000012FF10: 01 00 00 00 00 00 00 00-73 11 00 40 01 00 00 00 "........s..@...."
000000000012FF20: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 "................"
TID=6660|(0) KERNEL32.dll!GetLocalTime -> 0x150
```

[2] Смотрите также: X86 calling conventionshttp://en.wikipedia.org/wiki/X86_calling_conventions

```
Dump difference of buffer at argument 1 (starting at 1)
0000000000000000: D9 07 0C    06    05   -05    10    24    50 01 "... . . . . $ P."
```

Таким образом мы можем увидеть как win32-функция GetLocalTime() заполняет структуру SYSTEMTIME.

`PAUSE:<number>`: Сделать паузу, в миллисекундах. 1000 — одна секунда. Удобно для тестирования, для создания *искусственных* задержек. К примеру, полезно знать, как поведет себя программа при очень медленной сети:

```
tracer.exe -l:test1.exe bpf=WS2_32.dll!WSARecv,pause:1000
```

... или же, если будет считывать информацию с какого-то очень медленного носителя:

```
tracer.exe -l:test1.exe bpf=kernel32.dll!ReadFile,pause:1000
```

`RT_PROBABILITY:<number>`: Используется в паре с опцией `RT:`, задает вероятность срабатывания `RT`. К примеру, если был задан `RT:0` и `RT_PROBABILITY:30%`, то 0 будет подставляться вместо результата функции в 30% случаев. Это также удобно для тестирования — хорошо написанная программа должна корректно обрабатывать ошибки. Например, вот так мы можем симулировать ошибки выделения памяти, 1 вызов `malloc()` на сотню, вернет *NULL*:

```
tracer.exe -l:test1.exe bpf=msvcrt.dll!malloc,rt:0,rt_probability:1%
```

... в 10% случаев, файл не будет открываться:

```
tracer.exe -l:test1.exe bpf=kernel32.dll!CreateFile,rt:0,rt_probability:10%
```

Вероятность также можно задавать и обычным образом, как число в интервале от 0 (никогда) до 1 (всегда). 10% это 0.1, 3% это 0.03, итд.

Об идеях, какие еще ошибки можно симулировать, читайте так же здесь Oracle RDBMS internal self-testing features.

3.1 Опция TRACE

`TRACE`: трассировать функцию по одной инструкции и сохранять значения всех интересующих нас регистров. После исполнения, эта информация сохранится в файлы process.exe.idc, process.exe.txt, process.exe_clear.idc. .idc-файлы являются скриптами для IDA, а к .txt файлу можно применять grep, awk, sed для поиска интересующих нас значений.

Возьмем для примера функцию add_member из статьи *Using Uninitialized Memory for Fun and Profit*[3]:

```
int dense[256];
int dense_next=0;
int sparse[256];

void add_member(int i)
{
        dense[dense_next]=i;
        sparse[i]=dense_next;
        dense_next++;

};

int main ()
{
        add_member(123);
        add_member(5);
```

[3] http://research.swtch.com/2008/03/using-uninitialized-memory-for-fun-and.html

```
        add_member(71);
        add_member(99);
}
```

Скомпилируем и запустим трассировку на функции add_member (вначале узнайте адрес функции при помощи IDA):

```
tracer -l:trace_test4.exe bpf=0x00401000,trace:cc
```

Получим файл trace_test4.exe.txt:

```
0x401000, e=       4
0x401001, e=       4
0x401003, e=       4, [0x403818]=0..3
0x401008, e=       4, [EBP+8]=5, 0x47('G'), 0x63('c'), 0x7b('{')
0x40100b, e=       4, ECX=5, 0x47('G'), 0x63('c'), 0x7b('{')
0x401012, e=       4, [EBP+8]=5, 0x47('G'), 0x63('c'), 0x7b('{')
0x401015, e=       4, [0x403818]=0..3
0x40101a, e=       4, EAX=0..3
0x401021, e=       4, [0x403818]=0..3
0x401027, e=       4, ECX=0..3
0x40102a, e=       4, ECX=1..4
0x401030, e=       4
0x401031, e=       4, EAX=0..3
```

Поле *e* - это сколько раз была исполнена эта инструкция.

Загрузим trace_test4.exe.idc в IDA и увидим:

```
add_member      proc near               ; CODE XREF: _main+5↓p
                                        ; _main+F↓p ...

arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                mov     eax, dense_next ; [0x403818]=0..3
                mov     ecx, [ebp+arg_0] ; [EBP+8]=5, 0x47('G'), 0x63('c'), 0x7b('{')
                mov     dense[eax*4], ecx ; ECX=5, 0x47('G'), 0x63('c'), 0x7b('{')
                mov     edx, [ebp+arg_0] ; [EBP+8]=5, 0x47('G'), 0x63('c'), 0x7b('{')
                mov     eax, dense_next ; [0x403818]=0..3
                mov     sparse[edx*4], eax ; EAX=0..3
                mov     ecx, dense_next ; [0x403818]=0..3
                add     ecx, 1          ; ECX=0..3
                mov     dense_next, ecx ; ECX=1..4
                pop     ebp
                retn                    ; EAX=0..3
add_member      endp
```

Рис. 3.1: trace_test4.png

Понимать работу функции во время исполнения, таким образом, становится намного проще.

Исполненные инструкции подсвечиваются голубым цветом. Неисполненные остаются белыми.

Чтобы стереть все комментарии и подсветку, нужно исполнить скрипт trace_test4.exe_clear.idc

Информация в IDA-скрипте может приводится в сокращенной форме из-за того что IDA имеет ограничение на длину комментария, например: `EAX=[ 64 unique items. min=0xbca6eb7, max=0xfffffed ]`. В текстовом же файле сохраняется всё, поэтому иногда этот файл может оказаться в итоге очень большим.

Недостаток опции TRACE в том что она работает медленно, хотя и функции в системных DLL пропускаются (системной считается та DLL которая находится внутри %SystemRoot%) Вторая про-

блема в том что пока что не очень корректно трассируются вещи вроде исключений, setjmp/longjmp и подобных непредвиденных изменений пути исполнения кода.

3.2 Примеры

3.2.1 Простое использование

```
tracer.exe -l:bzip2.exe bpf=.*!fprintf,args:3
```

```
TID=5128|(0) cygwin1.dll!fprintf (0x61103150, "%s: I won't write compressed data to a
    terminal.\n", "bzip2") (called from 0x401e03 (bzip2.exe!BASE+0x1e03))
TID=5128|(0) cygwin1.dll!fprintf -> 0x34
TID=5128|(0) cygwin1.dll!fprintf (0x61103150, "%s: For help, type: `%s --help'.\n", "bzip2
    ") (called from 0x401c66 (bzip2.exe!BASE+0x1c66))
TID=5128|(0) cygwin1.dll!fprintf -> 0x27
```

3.2.2 Перехват некоторых Windows-функций для работы с реестром

```
tracer.exe -l:someprocess.exe bpf=advapi32.dll!RegOpenKeyExA,args:5 bpf=advapi32.dll!
    RegQueryValueExA,args:6 bpf=advapi32.dll!RegSetValueExA,args:6
```

.. или измените суффиксы функция на W и добавьте опцию UNICODE:

```
tracer64.exe -l:far.exe bpf=advapi32.dll!RegOpenKeyExW,args:5,unicode bpf=advapi32.dll!
    RegQueryValueExW,args:6,unicode bpf=advapi32.dll!RegSetValueExW,args:6,unicode
```

3.2.3 Подавить шумный сигнал

```
tracer.exe -l:beeper.exe bpf=kernel32.dll!Beep,args:2,skip_stdcall,rt:1
```

3.2.4 Подавить диалоговое окно с сообщением

... и сделать так что вызываемая функция будет считать что пользователь каждый раз нажимает OK (константа IDOK равняется 1):

```
tracer.exe -l:filename.exe bpf=user32.dll!MessageBoxA,args:4,skip_stdcall,rt:1
```

... или CANCEL (константа IDCANCEL равняется 2):

```
tracer.exe -l:filename.exe bpf=user32.dll!MessageBoxA,args:4,skip_stdcall,rt:2
```

3.2.5 Перехват вызовов rand()

Бывает весело перехватывать вызовы функции rand() в различных играх. Например, пасьянс Solitaire в Windows использует его для того чтобы сгенерировать случайный расклад. Мы можем установить возвращаемое значение rand() в ноль, и тогда Solitaire будет раздавать один и тот же расклад, всегда:

В Windows XP x86/x64:

```
tracer.exe/tracer64.exe -l:c:\windows\system32\sol.exe bpf=.*!rand,rt:0
```

В Windows 7 x64:

```
tracer64.exe -l:[full path to]\Solitaire.exe bpf=.*!rand,rt:0
```

3.2.6 FreeCell

Когда вы запускаете FreeCell в Windows (XP SP3) и нажимаете F2 (Новая игра), вы видите сообщение "Do you want to resign this game?"Мы можем подавить звуковой сигнал и сделать так что FreeCall будет думать что пользователь всегда нажимает YES:

Константа IDYES - 6. FreeCell использует функцию MessageBoxW - суффикс W означает уникодную версию функции MessageBox.

В Windows XP SP3 x86:

```
tracer.exe -l:c:\windows\system32\freecell.exe bpf=user32.dll!messagebeep,args:1,
    skip_stdcall bpf=user32.dll!messageboxw,args:4,unicode,skip_stdcall,rt:6
```

```
(0) user32.dll!messagebeep (0x20) (called from freecell.exe!BASE+0x1f52 (0x1001f52))
(0) Skipping execution of this function
(0) user32.dll!messagebeep () -> 0x8
(1) user32.dll!messageboxw (0x160152, "Do you want to resign this game?", "FreeCell", 0x24
    ) (called from freecell.exe!BASE+0x1f5f (0x1001f5f))
(1) Skipping execution of this function
(1) user32.dll!messageboxw () -> 0x8
(1) Modifying EAX register to 0x6
```

В Windows XP SP2 x64 Russian:

```
tracer64.exe -l:c:\windows\system32\freecell.exe bpf=user32.dll!messagebeep,args:1,skip
    bpf=user32.dll!messageboxw,args:4,unicode,skip,rt:6
```

```
TID=2836|(0) user32.dll!messagebeep (0x20) (called from freecell.exe!BASE+0x23f9 (0
    x1000023f9))
(0) Skipping execution of this function
TID=2836|(0) user32.dll!messagebeep () -> 0x8
TID=2836|(1) user32.dll!messageboxw (0x5010e, "Do you want to resign this game?", "
    FreeCell", 0x24) (called from freecell.exe!BASE+0x2416 (0x100002416))
(1) Skipping execution of this function
TID=2836|(1) user32.dll!messageboxw () -> 0x8
TID=2836|(1) Modifying RAX register to 0x6
```

3.2.7 Проверка ивентов и запись в лог в Oracle RDBMS

В Oracle 10.2.0.1 win64:

```
tracer64.exe -a:oracle.exe bpf=oracle.exe!ksdpec,args:1 bpf=oracle.exe!ss_wrtf,args:3
```

(Смотрите также: http://blog.yurichev.com/node/14)

```
TID=3032|(0) oracle.exe!ksdpec (0x2743) (called from 0x9580a9 (oracle.exe!opiodr+0x105))
TID=3032|(0) oracle.exe!ksdpec -> 0xff
TID=3032|(1) oracle.exe!ss_wrtf (0x4a0, "*** 2009-12-04 06:19:01.005\n", 0x1b) (called
    from 0x45318d (oracle.exe!sdpri+0x22d))
TID=3032|(1) oracle.exe!ss_wrtf -> 1
TID=3032|(1) oracle.exe!ss_wrtf (0x4a0, "OPI CALL: type=107 argc= 3 cursor=  0 name=SES
    OPS (80)\n", 0x37) (called from 0x45318d (oracle.exe!sdpri+0x22d))
TID=3032|(1) oracle.exe!ss_wrtf -> 1
TID=3032|(0) oracle.exe!ksdpec (0x2743) (called from 0x9580a9 (oracle.exe!opiodr+0x105))
TID=3032|(0) oracle.exe!ksdpec -> 0xff
TID=3032|(1) oracle.exe!ss_wrtf (0x4a0, "OPI CALL: type=59 argc= 4 cursor=  0 name=
    VERSION2\n", 0x32) (called from 0x45318d (oracle.exe!sdpri+0x22d))
```

```
TID=3032|(1) oracle.exe!ss_wrtf -> 1
TID=3032|(0) oracle.exe!ksdpec (0x273e) (called from 0x4a00cc (oracle.exe!kslwte_tm+0x7a8)
    )
TID=3032|(0) oracle.exe!ksdpec -> 0
TID=3032|(0) oracle.exe!ksdpec (0x273e) (called from 0x4a00cc (oracle.exe!kslwte_tm+0x7a8)
    )
TID=3032|(0) oracle.exe!ksdpec -> 0
TID=3032|(0) oracle.exe!ksdpec (0x2743) (called from 0x9580a9 (oracle.exe!opiodr+0x105))
TID=3032|(0) oracle.exe!ksdpec -> 0xff
TID=3032|(1) oracle.exe!ss_wrtf (0x4a0, "OPI CALL: type=104 argc=12 cursor=  0 name=
    Transaction Commit/Rollback\n", 0x46) (called from 0x45318d (oracle.exe!sdpri+0x22d))
TID=3032|(1) oracle.exe!ss_wrtf -> 1
```

3.2.8 Слежение за выделением памяти в Oracle 11.1.0.6.0 win32/win64

```
tracer.exe/tracer64.exe -a:oracle.exe bpf=.*!kghalf,args:6 bpf=.*!kghfrf,args:4
```

```
TID=1600|(0) oracle.exe!kghalf (0x6d35af0, 0xb507ef8, 0x1000, 0, 0, "kzsrcrdi") (called
    from 0x1c7aa83 (oracle.exe!kzctxhugi+0x71))
TID=1600|(0) oracle.exe!kghalf -> 0xfa3ea58

TID=1600|(0) oracle.exe!kghalf (0x6d35af0, 0xb507ef8, 0x58, 1, 0x6d35530, "UPI heap") (
    called from 0x1e7f8b7 (oracle.exe!__PGOSF266_kwqmahal+0x5b))
TID=1600|(0) oracle.exe!kghalf -> 0xfa4d0d8

TID=1188|(0) oracle.exe!kghalf (0xda39540, 0xda39240, 0x88, 0, "ksirmdt array", 0xda39240)
     (called from 0x6afb5b (oracle.exe!ksz_nfy_ipga+0xf1))
TID=1188|(0) oracle.exe!kghalf -> 0x105d0b10

TID=1188|(0) oracle.exe!kghalf (0xda39540, 0xda39240, 0x48, 1, 0x1204e400, "local") (
    called from 0x3684a64 (oracle.exe!kjztcxini+0x58))
TID=1188|(0) oracle.exe!kghalf -> 0x105d0ab0
```

3.2.9 Слежение за разбором SQL-выражений в Oracle RDBMS

В Oracle 11.1.0.6.0 win32/win64:

```
tracer.exe/tracer64.exe -a:oracle.exe bpf=oracle.exe!_?rpisplu,args:8 bpf=oracle.exe!_?
    kprbprs,args:7 bpf=oracle.exe!_?opiprs,args:6 bpf=oraclient11.dll!OCIStmtPrepare,args
    :6</i></p>
```

Замечание: регулярное выражение *_?function* покрывает оба имени: `function` и `_function`.

```
TID=1140|(2) oracle.exe!opiprs (0x13f029d0, "select 1 from obj$ where name='
    DBA_QUEUE_SCHEDULES'", 0x34, 0x10ae7f50, 0x840082, 0xd9f7a10) (called from 0x6ba3bf (
    oracle.exe!__PGOSF423_kksParseChildCursor+0x2dd))
TID=1140|(2) oracle.exe!opiprs -> 0
TID=1140|(2) oracle.exe!opiprs (0x13f029d0, "select 1 from sys.aq$_subscriber_table where
    rownum < 2 and subscriber_id <> 0 and table_objno <> 0", 0x64, 0x10ad5de8, 0, 0
    x13f007e0) (called from 0x6ba3bf (oracle.exe!__PGOSF423_kksParseChildCursor+0x2dd))
TID=1140|(2) oracle.exe!opiprs -> 0
TID=1140|(0) oracle.exe!rpisplu (3, 0, 0, 0, 0, 0x14430ac0, 0, 0) (called from 0x250b33c (
    oracle.exe!kqdGetCursor+0x106))
TID=1140|(0) oracle.exe!rpisplu -> 0
```

```
TID=1288|(2) oracle.exe!opiprs (0x17df8130, "select * from v$version", 0x18, 0x10adee60,
    0, 0) (called from 0x6ba3bf (oracle.exe!__PGOSF423_kksParseChildCursor+0x2dd))
TID=1288|(1) oracle.exe!kprbprs (0xa82bc50, 0, "select timestamp, flags from fixed_obj$
    where obj#=:1", 0x35, 0xffffe3e0, 0x2040800, 1) (called from 0x2ba1b1f (oracle.exe!
    kqldtstr+0x151))
TID=1288|(1) oracle.exe!kprbprs -> 0
TID=1288|(0) oracle.exe!rpisplu (0x1f, 0, 0, 0, 0, 0x2bb5e04, "select  BANNER from
    GV$VERSION where inst_id = USERENV('Instance')", 0xffffc085) (called from 0x2bbcabf (
    oracle.exe!kqldFixedTableLoadCols+0x157))
TID=1288|(1) oracle.exe!kprbprs (0x1090c108, 0, "select timestamp, flags from fixed_obj$
    where obj#=:1", 0x35, 0xffffe3e0, 0x2040800, 1) (called from 0x2ba1b1f (oracle.exe!
    kqldtstr+0x151))
TID=1288|(1) oracle.exe!kprbprs -> 0
TID=1288|(1) oracle.exe!kprbprs (0x10908060, 0, "select timestamp, flags from fixed_obj$
    where obj#=:1", 0x35, 0xffffe3e0, 0x2040800, 1) (called from 0x2ba1b1f (oracle.exe!
    kqldtstr+0x151))
TID=1288|(1) oracle.exe!kprbprs -> 0
TID=1288|(2) oracle.exe!opiprs -> 0
TID=1288|(0) oracle.exe!rpisplu -> 0
TID=1288|(0) oracle.exe!rpisplu (0x16, 0, 0, 0, 0, 0x10b3ce50, 0, 0) (called from 0
    x250b33c (oracle.exe!kqdGetCursor+0x106))
TID=1288|(0) oracle.exe!rpisplu -> 0
```

3.2.10 Игнорирование неподписанных драйверов

```
tracer.exe -l:target.exe bpf=Wintrust.dll!WinVerifyTrust,rt:0
```

3.2.11 Вывод памяти по аргументам функций

```
tracer.exe -l:rar.exe "-c:a archive.rar *.exe" bpf=kernel32.dll!writefile,args:5,dump_args
    :0x10
```

RAR записывает свою сигнатуру в начало файла archive.rar:

```
TID=7000|(0) KERNEL32.dll!WriteFile (0x118, 0x152410, 7, 0x150fc0, 0) (called from 0
    x403721 (rar.exe!__GetExceptDLLinfo+0x26c8))
Dump of buffer at argument 2 (starting at 1)
00152410: 52 61 72 21 1A 07 00 00-50 30 15 00 5D 83 40 00 "Rar!....P0..].@."
Dump of buffer at argument 4 (starting at 1)
00150FC0: 00 00 00 00 21 7B 40 00-10 24 15 00 18 24 15 00 "....!{@..$...$.."
TID=7000|(0) KERNEL32.dll!WriteFile -> 1
```

3.2.12 Вывод памяти по аргументам функций и слежение за её изменением

```
tracer.exe -l:rar.exe "-c:x archive.rar" bpf=kernel32.dll!readfile,args:4,dump_args:0x10
```

Архиватор RAR открывает файл archive.rar и первым делом читает сигнатуру:

```
TID=6148|(0) KERNEL32.dll!ReadFile (0x120, 0x17b3f8, 7, 0x174c50) (called from 0x403966 (
    rar.exe!__GetExceptDLLinfo+0x290d))
Dump of buffer at argument 2 (starting at 1)
0017B3F8: 00 00 00 00 00 00 00 00-00 00 00 00 48 00 00 00 "............H..."
Dump of buffer at argument 4 (starting at 1)
```

```
00174C50: 07 00 00 00 78 4C 17 00-7A 38 40 00 8C 6D 17 00 "....xL..z8@..m.."
TID=6148|(0) KERNEL32.dll!ReadFile -> 1
Dump difference of buffer at argument 2 (starting at 1)
00000000: 52 61 72 21 1A 07      -                        "Rar!..          "
```

3.3 Примеры опции TRACE

3.3.1 Трассировка строковых функций

Возьмем пример применения strtok():

```
// example from http://www.cplusplus.com/reference/clibrary/cstring/strtok/

/* strtok example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] ="- This, a sample string.";
  char * pch;
  printf ("Splitting string \"%s\" into tokens:\n",str);
  pch = strtok (str," ,.-");
  while (pch != NULL)
  {
    printf ("%s\n",pch);
    pch = strtok (NULL, " ,.-");
  }
  return 0;
}
```

И трассируем функцию main():

```
tracer.exe -l:trace_test1.exe bpf=0x00401000,trace:cc
```

После исполнения скрипта в IDA (показана только тело цикла *while*):

```
loc_401050:                             ; CODE XREF: _main+66↓j
                push    eax             ; EAX=0x18ff30, ptr to "This", "sample", "string"
                push    offset aS       ; "%s\n"
                call    esi ; printf    ; ESI=0x6f6b20c1; comments: op1=MSVCR90.dll!printf
                push    offset a__0     ; " ,.-"
                push    0               ; Str
                call    edi ; strtok    ; EDI=0x6f6b6f2e; comments: op1=MSVCR90.dll!strtok
                add     esp, 10h
                test    eax, eax        ; EAX=0, 0x18ff30, ptr to "sample", "string"
                jnz     short loc_401050 ; flags: ZF zf
```

Рис. 3.2: trace_test1.png

Замечание: "a"это слишком короткая строка для автоматического детектора строк в tracer, поэтому её здесь нет, вместо нее адрес этой строки.

3.3.2 Трассируем quicksort()

Возьмем известный пример:

```
//http://cplus.about.com/od/learningc/ss/pointers2_8.htm

/* ex3 Sorting ints with qsort */
//

#include <stdio.h>
#include <stdlib.h>

int comp(const int * a,const int * b)
{
  if (*a==*b)
    return 0;
  else
    if (*a < *b)
        return -1;
     else
      return 1;
}

int main(int argc, char* argv[])
{
   int numbers[10]={1892,45,200,-98,4087,5,-12345,1087,88,-100000};
   int i;

  /* Sort the array */
  qsort(numbers,10,sizeof(int),comp);
  for (i=0;i<9;i++)
    printf("Number = %d\n",numbers[ i ]);
  return 0;
}
```

Трассируем функцию comp():

```
tracer.exe -l:trace_test2.exe bpf=0x00401030,trace:cc
```

Получим после исполнения скрипта в IDA:

В примере все значения уникальны, одинаковых нет. Таким образом, нет ситуации когда comp() возвращает ноль. Поэтому здесь мы видим что часть comp() возвращающая ноль (xor eax,eax / retn) не была исполнена ни разу.

```
; int __cdecl comp(const void *, const void *)
comp            proc near               ; DATA XREF: _main+5↓o

arg_0           = dword ptr  4
arg_4           = dword ptr  8

                mov     eax, [esp+arg_0] ; [ESP+4]=0x18ff20, 0x18ff24, 0x18ff28, 0x18ff2c, 0x18
                mov     ecx, [esp+arg_4] ; [ESP+8]=0x18ff20, 0x18ff24, 0x18ff28, 0x18ff30, 0x18
                mov     eax, [eax]      ; [EAX]=5, 0x2d('-'), 0x58('X'), 0xc8, 0x43f, 0x764, 0x
                mov     ecx, [ecx]      ; [ECX]=5, 0x58('X'), 0xc8, 0x764, 0xff7, 0xfffe7960, 0x
                cmp     eax, ecx        ; EAX=5, 0x2d('-'), 0x58('X'), 0xc8, 0x43f, 0x764, 0xff
                                        ; ECX=5, 0x58('X'), 0xc8, 0x764, 0xff7, 0xfffe7960, 0xf
                jnz     short loc_401013 ; flags: zf
                xor     eax, eax
                retn
; ---------------------------------------------------------------------------

loc_401013:                             ; CODE XREF: comp+E↑j
                xor     edx, edx
                cmp     eax, ecx        ; EAX=5, 0x2d('-'), 0x58('X'), 0xc8, 0x43f, 0x764, 0xff
                                        ; ECX=5, 0x58('X'), 0xc8, 0x764, 0xff7, 0xfffe7960, 0xf
                setnl   dl              ; flags: SF sf of
                lea     edx, [edx+edx-1] ; op2=1, 0xffffffff
                mov     eax, edx        ; EDX=1, 0xffffffff
                retn                    ; EAX=1, 0xffffffff
comp            endp
```

Рис. 3.3: trace_test2.png

Глава 4

BPX: установка прерывания на произвольное место

Содержимое регистров процессора будет выведено.

Если хотя бы один регистр FPU что-то содержит, он также будет выведен.

Если содержимое FPU-регистра NaN (нечисло), содержимое регистра FPU будет трактовано как регистра MMX и также будет выведено.

`DUMP(ADDRESS|REGISTER|SYMBOL[+OFFSET],SIZE)`: вывод содержимого памяти. Определить адрес в памяти можно в виде шестнадцатиричного значения или в виде `REGISTER+OFFSET`. `SIZE` — это размер дампа.

Если перед адресом или регистром поставить символ `*`, то tracer вначале прочитает DWORD (или QWORD в x64-версии), примет его за адрес и выдаст дамп по нему. Например: `dump(*ebx,0x100)` — взять адрес из ячейки памяти на которую указывает регистр ebx и выдать дамп размером 0x100 байт.

`COPY(ADDRESS|REGISTER|SYMBOL[+OFFSET],C-string)`: скопировать Си-строку по указанному адресу. Си-строка может быть как ASCII-строкой, так и содержать последовательности `\xXX`, где `XX` — шестнадцатиричное число. Например: `COPY(EAX,a\x34\x56)` — скопирует три байта 'a', 0x34, и 0x56 по адресу который содержится в EAX.

`SET (REGISTER,VALUE)`: записать значение в регистр. EIP/RIP, регистры FPU ST0..ST7 и флаги (PF, SF, AF, ZF, OF, CF, DF) можно модифицировать. Значение трактуется как десятичное число или как число с плавающей запятой, если только не указан префикс 0x.

Замечание: tracer не модифицирует tag word register в FPU, также он не модифицирует регистр TOP, таким образом, если какой-то регистр FPU маркирован как "пустой"и tracer запишет туда какое-то значение, он останется маркированным как "пустой".

Изменение значения регистра EIP/RIP иными словами это передача исполнения в другое место. Это удобно для того чтобы пропускать некоторые куски кода.

4.1 Примеры

4.1.1 Task Manager: создать иллюзию что у нас 32 или 64 процессора

В Windows XP SP2 x64 Russian:

```
tracer64.exe -l:c:\windows\system32\taskmgr.exe bpx=0x0000000010000A8E4,set(rax,64)
```

В Windows XP SP3 x86 English:

```
tracer.exe -l:c:\windows\system32\taskmgr.exe bpx=0x01006647,set(eax,32)
```

4.1.2 Перехват развернутой (inline) функции strcmp()

Представим что у нас есть такой код который мы компилируем в MS VC 2008:

```
printf ("%d\n", strcmp("one", "two"));
```

После компиляции мы получим:

```
<pre>
.text:00401000 BA 50 A1 40 00                  mov     edx, offset aTwo ; "two"
.text:00401005 B9 54 A1 40 00                  mov     ecx, offset aOne ; "one"
.text:0040100A 8D 9B 00 00 00 00               lea     ebx, [ebx+0]
.text:00401010
.text:00401010                 loc_401010:                             ; CODE XREF:
   _main+2A
.text:00401010 8A 01                           mov     al, [ecx]
.text:00401012 3A 02                           cmp     al, [edx]
.text:00401014 75 29                           jnz     short loc_40103F
.text:00401016 84 C0                           test    al, al
.text:00401018 74 12                           jz      short loc_40102C
.text:0040101A 8A 41 01                        mov     al, [ecx+1]
.text:0040101D 3A 42 01                        cmp     al, [edx+1]
.text:00401020 75 1D                           jnz     short loc_40103F
.text:00401022 83 C1 02                        add     ecx, 2
.text:00401025 83 C2 02                        add     edx, 2
.text:00401028 84 C0                           test    al, al
.text:0040102A 75 E4                           jnz     short loc_401010
.text:0040102C
.text:0040102C                 loc_40102C:                             ; CODE XREF:
   _main+18
.text:0040102C 33 C0                           xor     eax, eax
.text:0040102E 50                              push    eax
.text:0040102F 68 58 A1 40 00                  push    offset byte_40A158 ; char *
.text:00401034 E8 1C 00 00 00                  call    _printf
.text:00401039 83 C4 08                        add     esp, 8
.text:0040103C 33 C0                           xor     eax, eax
.text:0040103E C3                              retn
```

Давайте перехватим эту развернутую функцию strcmp и выведем то на что указывают регистры ECX и EDX:

```
tracer.exe -l:strcmp.exe bpx=8A013A02752984C074128A41013A4201751D83C10283C20284C075E433C0,
   dump(ecx,0x10),dump(edx,0x10)
```

Получим:

```
bytemask_0 is resolved to address 0x401010 (strcmp.exe)
TID=6436|(0) 0x401010 (strcmp.exe!BASE+0x1010)
EAX=0x007722E0 EBX=0x7EFDE000 ECX=0x0040A154 EDX=0x0040A150
ESI=0x00000000 EDI=0x00000000 EBP=0x0018FF88 ESP=0x0018FF44
EIP=0x00401010
FLAGS=PF ZF IF
Dumping memory at ECX
0040A154: 6F 6E 65 00 25 64 0A 00-28 00 6E 00 75 00 6C 00 "one.%d..(.n.u.l."
Dumping memory at EDX
0040A150: 74 77 6F 00 6F 6E 65 00-25 64 0A 00 28 00 6E 00 "two.one.%d..(.n."
```

Замечание: только первое вхождение при поиске байтмаски будет использоваться.

4.1.3 Изменение флагов перед тем как условный переход будет совершен

```
tracer64.exe -l:flags.exe bpx=0x140001014,set(zf,1)
```

Замечание: момент когда tracer меняет состояние регистров это момент *перед* тем как текущая инструкция будет исполнена. Изменение флагов *перед* инструкциями TEST или CMP бессмысленно.

4.1.4 Шутка в Microsoft Excel

Сделать 666 результатом всех операций деления. Введите -(123/456)"для проверки.

Работает для Excel.exe версии 14.0.4756.1000 (Microsoft Office 2010)

```
tracer.exe -l:excel.exe bpx=excel.exe!base+0x11E91B,set(st0,666)
```

```
tracer64.exe -l:excel.exe bpx=excel.exe!base+0x1B7FCC,set(st0,666)
```

(Указанный адрес это место после инструкции FDIV, которая собственно и производит деление)

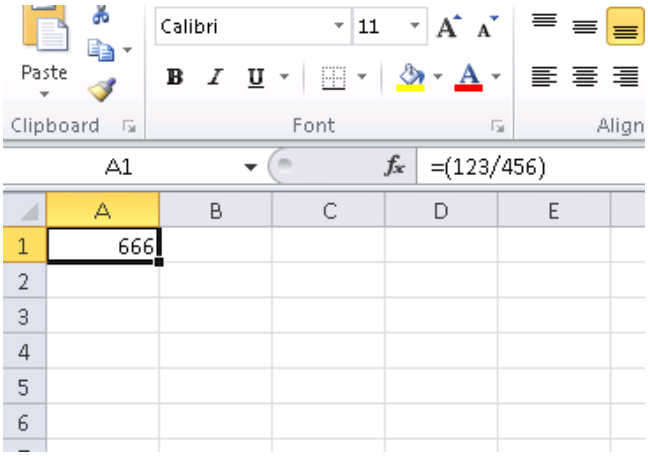

Рис. 4.1: excel_prank.png

Глава 5

BPM: установка прерывания на обращение к ячейке памяти

Архитектура x86 позволяет устанавливать прерывания на обращение к ячейкам памяти.

Таким образом, если кто-то или что-то модифицирует в памяти какое-то значение, tracer тут же будет об этом знать.

Следует также заметить, что это практично только для глобальных переменных а не локальных (размещаемых в стеке).

`BPMB=<address>,<option>`: установить прерывание на обращение к байту. `BPMW=<address>,<option>`: установить прерывание на обращение к 16-битному слову (word).

`BPMD=<address>,<option>`: установить прерывание на обращение к 32-битному слову (dword).

`BPMQ=<address>,<option>`: установить прерывание на обращение к 64-битному слову (qword) (доступно только в tracer64).

`W`: установить прерывание только на запись в ячейку памяти.

`RW`: установить прерывание на запись и чтение из ячейки памяти.

Замечание: по какой-то неизвестной причине, архитектура Intel предоставляет только две эти возможности.

5.1 Примеры

5.1.1 Слежение за обращением к переменным в Oracle RDBMS

Давайте попробуем следить за всеми чтениями и записями в глобальную переменную ktsmgd и видеть стек вызовов:

```
tracer.exe -a:oracle.exe -s bpmd=oracle.exe!_?ktsmgd_,rw
```

Запустите в консоли SQL*Plus (залогиньтесь перед этим как SYS):

```
ALTER SYSTEM SET "_disable_txn_alert"=1;
```

Получим:

```
TID=2852|(0) oracle.exe!_ktsmgdcb+0x18: some code reading or writting DWORD variable at
    oracle.exe!_ktsmgd_ (now it contain 0x1)
Call stack of thread TID=2852
return address=0x4682f0 (oracle.exe!_kspptval+0x704)
return address=0x4674b0 (oracle.exe!_kspset0+0x928)
return address=0x8f23c6 (oracle.exe!_kkyasy+0x3cda)
return address=0x92ba1d (oracle.exe!_kksExecuteCommand+0x475)
return address=0x1f75e02 (oracle.exe!_opiexe+0x4bda)
return address=0x1e98390 (oracle.exe!_kpoal8+0x900)
return address=0x9df597 (oracle.exe!_opiodr+0x4cb)
return address=0x6102eb00 (oracommon11.dll!_ttcpip+0xab0)
```

```
return address=0x9de77e (oracle.exe!_opitsk+0x4fe)
return address=0x1fdf128 (oracle.exe!_opiino+0x430)
return address=0x9df597 (oracle.exe!_opiodr+0x4cb)
return address=0x450b1c (oracle.exe!_opidrv+0x32c)
return address=0x451352 (oracle.exe!_sou2o+0x32)
return address=0x401197 (oracle.exe!_opimai_real+0x87)
return address=0x401061 (oracle.exe!_opimai+0x61)
return address=0x401c55 (oracle.exe!_OracleThreadStart@4+0x301)
return address=0x77e66063 (KERNEL32.dll!GetModuleFileNameA+0xeb)
```

Читайте больше тут: http://blog.yurichev.com/node/3 о параметре `_disable_txn_alert` и значении переменной `ktsmgd`.

5.1.2 Проверяет ли программа целостность своего кода?

Такие точки прерываний удобно устанавливать не только на переменные в памяти, но также и на участки исполняемого кода, чтобы узнать, проверяет ли программа целостность своего кода, был ли он модифицирован?

Часто, в таких случаях, некая функция просто вычисляет контрольную сумму всего исполняемого файла, либо исполняемых PE-секций, либо отдельных функций. Устанавливая `BPMB` с параметром `R` на начало какой-либо функции, можно быстро узнать, происходят такие проверки, или нет.

Глава 6

Одноразовое прерывание по INT3

Этот метод прерывания позволяет поставить сразу очень много прерываний типа INT3 по маске. К примеру, можно поставить точки прерывания на все функции какой-либо DLL:

```
--one-time-INT3-bp:somedll.dll!.*
```

Либо, поставим INT3-прерывание на все функции, имена которых начинаются с префикса `xml`:

```
--one-time-INT3-bp:somedll.dll!xml.*
```

В качестве обратной стороны медали, такие прерывания срабатывают только один раз.

Tracer покажет вызов какой-либо функции, если он случится, но только один раз. Еще один недостаток — увидеть аргументы функции также нельзя.

Тем не менее, эта возможность очень удобна для тех ситуаций, когда вы знаете что некая программа использует некую DLL, но не знаете какие именно функции. И функций много.

Например, попробуем узнать, что использует cygwin-утилита uptime:

```
tracer -l:uptime.exe --one-time-INT3-bp:cygwin1.dll!.*
```

Так мы можем увидеть все ф-ции из библиотеки cygwin1.dll, которые были вызваны хотя бы один раз, и откуда:

```
One-time INT3 breakpoint: cygwin1.dll!__main (called from uptime.exe!OEP+0x6d (0x40106d))
One-time INT3 breakpoint: cygwin1.dll!_geteuid32 (called from uptime.exe!OEP+0xba3 (0
    x401ba3))
One-time INT3 breakpoint: cygwin1.dll!_getuid32 (called from uptime.exe!OEP+0xbaa (0
    x401baa))
One-time INT3 breakpoint: cygwin1.dll!_getegid32 (called from uptime.exe!OEP+0xcb7 (0
    x401cb7))
One-time INT3 breakpoint: cygwin1.dll!_getgid32 (called from uptime.exe!OEP+0xcbe (0
    x401cbe))
One-time INT3 breakpoint: cygwin1.dll!sysconf (called from uptime.exe!OEP+0x735 (0x401735)
    )
One-time INT3 breakpoint: cygwin1.dll!setlocale (called from uptime.exe!OEP+0x7b2 (0
    x4017b2))
One-time INT3 breakpoint: cygwin1.dll!_open64 (called from uptime.exe!OEP+0x994 (0x401994)
    )
One-time INT3 breakpoint: cygwin1.dll!_lseek64 (called from uptime.exe!OEP+0x7ea (0x4017ea
    ))
One-time INT3 breakpoint: cygwin1.dll!read (called from uptime.exe!OEP+0x809 (0x401809))
One-time INT3 breakpoint: cygwin1.dll!sscanf (called from uptime.exe!OEP+0x839 (0x401839))
One-time INT3 breakpoint: cygwin1.dll!uname (called from uptime.exe!OEP+0x139 (0x401139))
One-time INT3 breakpoint: cygwin1.dll!time (called from uptime.exe!OEP+0x22e (0x40122e))
```

```
One-time INT3 breakpoint: cygwin1.dll!localtime (called from uptime.exe!OEP+0x236 (0
    x401236))
One-time INT3 breakpoint: cygwin1.dll!sprintf (called from uptime.exe!OEP+0x25a (0x40125a)
    )
One-time INT3 breakpoint: cygwin1.dll!setutent (called from uptime.exe!OEP+0x3b1 (0x4013b1
    ))
One-time INT3 breakpoint: cygwin1.dll!getutent (called from uptime.exe!OEP+0x3c5 (0x4013c5
    ))
One-time INT3 breakpoint: cygwin1.dll!endutent (called from uptime.exe!OEP+0x3e6 (0x4013e6
    ))
One-time INT3 breakpoint: cygwin1.dll!puts (called from uptime.exe!OEP+0x4c3 (0x4014c3))
```

Глава 7

Взаимодействие во время работы

1) Нажмите ESC или Ctrl-C для отсоединения от запущенного процесса.

2) Нажмите пробел чтобы увидеть стеки вызовов для каждого треда и процесса.

Например: присоеденитесь к какой-нибудь запущенной программе с открытым окном с сообщением (Message Box), нажмите пробел и возможно вы увидите, что привело к появлению этого окна.

Замечание: вывод стека вызовов пока плохо работает в tracer64.

Глава 8

Отсоединение от процесса

tracer использует функцию DebugActiveProcessStop() для отсоединения от запущенного процесса. Эта функция присутствует во всех современных ОС базирующихся на NT, возможно, кроме Windows NT и Windows 2000. Так что всё что tracer может сделать в этих ОС это убить процесс — извините!

Глава 9

Некоторые технические заметки

Архитектура x86 позволяет использовать до четырех прерываний одновременно. Таким образом, опции `BPF/BPX/BPM` могут комбинироваться в любом порядке до четырех раз.

Возможность вывода стека вызовов предпологает что фреймы в стеке "разделены"указателем в регистре EBP:

Смотрите также: Functions and Stack Frames

Это означает что любая функция которая не использует эту схему, будет исключена их стека вызовов — непреднамеренно.

Замечание: в tracer64 эта возможность работает не очень хорошо.

Вся информация выводится в stdout а также записывается в файл tracer.log. Файл создается снова при каждом запуске.

При загрузке или присоеденению к процессу, tracer проверяет все модули: главный исполняемый файл и все файлы DLL загружаемые после. Он извлекает все символы из модуля включая эксплорты DLL. Он также ищет файл FileName.MAP и пытается его парсить. Файл MAP имеет такой же формат как то что делает IDA. tracer также ищет файл FileName.SYM и пытается загрузить символы из него, трактуя его как файл символов из Oracle RDBMS: переменная окружения ORACLE_HOME должна быть установлена для этого. tracer также ищет файл FileName.PDB (компилируйте вашу программу в MSVC с ключом `/Zi` и вы получите отладочный файл PDB для нее).

Если DLL содержит экспорты только по ординалам, т.е., без имен (например, DLL-файлы MFC), имя символа будет получено из ординала в таком формате: `ordinal_<number>`, например, `ordinal_12`.

Глава 10

Известные проблемы

10.1 Windows 2000

Для работы под Windows 2000, библиотека Octothorpe должна быть скомпилирована с флагом TARGET_IS_WINDOWS_2000.

Во-вторых, файл, dbghelp.dll из Windows XP должен находится в той же директории что и tracer.exe.

Глава 11

Заключение

Эта версия еще не была протестирована как следует. Так что будьте готовы к неожиданным падениям. Я очень рекомендую проводить все эксперименты в виртуальной машине.

Если вы нашли ошибку, пожалуйста напишите мне: dennis@yurichev.com. Пришлите также файл tracer.log и/или скриншот того что вывел tracer.

Я буду также благодарен любому комментарию или предложению насчет tracer.

Если вы чувствуете что ваш вклад в код стоит того чтобы быть включенным в мою версию, пожалуйста присылайте ваш патч.

Tracer так же много используется для иллюстраций в моей книге “Краткое введение в reverse engineering для начинающих”, свободно доступной здесь.