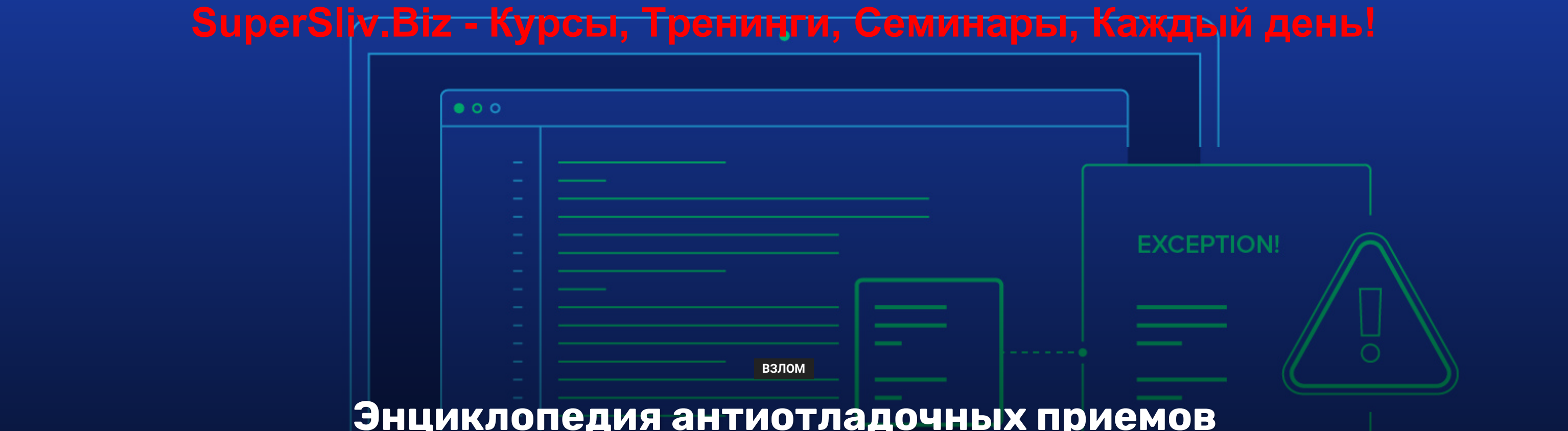


ВЗЛОМ

Энциклопедия антиотладочных приемов 3. Обработка необрабатываемых исключений

Крис Касперски, 01.07.2008

Отладчики, работающие через MS Debugging API (OllyDbg, IDA-Pro, MS VC), вынуждены мириться с тем, что отладочные процессы «страдают» хроническими особенностями поведения. Они «ломают» логику программы, и это с огромной выгодой используют защитные механизмы. В частности, API-функция SetUnhandledExceptionFilter() под отладчиком вообще не вызывается — вовсе не баг отладчика, а документированная фича системы!

FUNDAMENTALS

Рассматривая обработку структурных исключений в предыдущем выпуске, мы мельком упомянули, что всякий процесс от рождения получает первичный обработчик структурных исключений, назначаемый операционной системой по умолчанию. Если программист забыл (или не захотел) назначать свои собственные обработчики, то все исключения, возникающие при выполнении программы, попадают в пасть первичного обработчика. Он расположен в NTDLL.DLL и, в зависимости от настроек оси, либо вызывает «Доктора Ватсона», либо выводит знаменитый диалог о критической ошибке с вариантами: OK — завершить приложение в аварийном режиме и Cancel — вызывать Just-in-Time-отладчик (в роли которого может выступать и «Ольга»).

OllyDbg 1.x имеет коварный баг!

«Ольга» версии 1.10 имеет неприятный баг — если непосредственно за `INT 03h` следует команда `PUSHFD`, заталкивающая флаги в стек, отладчик едет крышей и теряет управление над отлаживаемой программой, даже если мы нажимаем F7/F8 (Step Into / Step Over).
Для демонстрации бага достаточно воткнуть в листинг пару команд `PUSHFD`/`POPFD`. А вот те же самые команды, отделенные от `INT 03h` одной или несколькими инструкциями NOP (или любыми другими), работают вполне нормально.

В «Ольге» 2.x ошибка уже исправлена, однако, учитывая, что 2.x все еще находится в стадии разработки, основным инструментом хакеров остается «Ольга» 1.10 с багом на борту.

То же самое происходит, если программист устанавливает один или несколько обработчиков структурных исключений, но никто из них не в состоянии справиться с ситуацией — вот они и передают исключение друг другу, пока оно не докатится до системного обработчика. Системный обработчик легко подменить своим (было бы желание). Достаточно вместо ссылки на предыдущий `EXCEPTION_REGISTRATION` затолкать в поле prev значение -1. Это будет свидетельствовать, что данный обработчик — последний в цепочке.

Как вариант, можно воспользоваться API-функцией `SetUnhandledExceptionFilter()`, перекрывающей обработчик исключений верхнего уровня (top-level exception handler). Да, именно «верхнего», поскольку Windows создавалась в Америке, расположенной на противоположной стороне Земли, где люди ходят вверх ногами. Первичный системный обработчик, с их точки зрения, находится на вершине пирамиды структурных исключений, в то время как русские программисты склонны рассматривать его как «основание». Но это все лирика, а дело-то в том, что...

SetUnhandledExceptionFilter() на MSDN

Функция `SetUnhandledExceptionFilter()`, перекрывая системный обработчик, в неволе работать отказывается, то есть получает управление, только когда процесс не находится под отладчиком. В противном случае исключение передается непосредственно самому отладчику. Это — задумка проектировщиков, кстати сказать довольно оригинальная и полезная. Если отладчика нет — установленный программистом обработчик берет управление на себя и завершает работу программы максимально корректным образом. Если же процесс находится под отладкой, операционная система передает бразды правления отладчику, позволяя разобраться с ситуацией, поскольку после завершения программы разбираться будет не с чем и некому.

А теперь, внимание, вопрос! Что произойдет, если в обработчик, установленный `SetUnhandledExceptionFilter()`, воткнуть не код аварийного завершения приложения, а кусок функциональности, например расшифровщик какой или просто пару строк на C, меняющих значение флага `under_debuuger`? Правильно — мы получим великолепный способ детекта отладчиков прикладного уровня!

ЭКСПЕРИМЕНТ — PRO ET CONTRA SETUNHANDLEDEXCEPTIONFILTER

Напишем простейшую тестовую программу, позволяющую исследовать реакцию отладчиков на фильтр, установленный функцией `SetUnhandledExceptionFilter()`. На crackme она никак не тянет (слишком прозрачна и элементарна), но crackme мы написать завсегда успеем! Сейчас главное — врубиться в тему и выяснить, насколько надежен этот трюк, можно ли его обойти и если да, то как.

Один из примеров реализации тестового стенда приведен ниже.

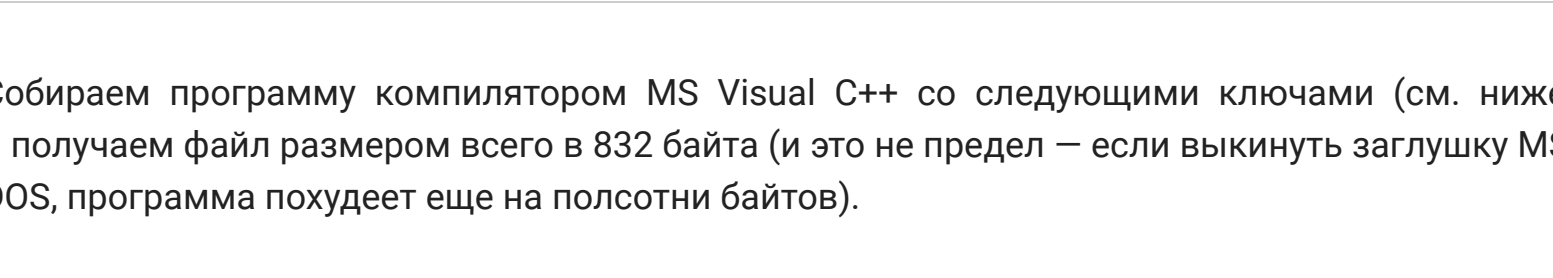
Исходный текст программы SetUnhandledExceptionFilter.c, демонстрирующей технику использования функции для борьбы с отладчиками

```
#include <windows.h>
char dbgnoo[]= "debugger is not detected";
char dbgyes[]= "debugger is detected :-)";

char *p = dbgyes;            // We expect debugger by default

LONG souriz(struct _EXCEPTION_POINTERS *ExceptionInfo)
{
    // if we're here, process is _not_ under debugger
    p = dbgnoo;

    // Skip INT 03 (CCh) command
    ExceptionInfo->ContextRecord->Eip++;

    // We want the program to continue execution
    return EXCEPTION_CONTINUE_EXECUTION;
}

nezumi()
{
    // Supersede the default top-level exception handler by souriz() proc
    SetUnhandledExceptionFilter((LPTOP_LEVEL_EXCEPTION_FILTER)&souriz);
    __asm{
        int 03              ; // generate an exception
    }

    // Terminate the program, showing the result
    ExitProcess(MessageBox(0,p,p,0));
}
```

Собираем программу компилятором MS Visual C++ со следующими ключами (см. ниже) и получаем файл размером всего в 832 байта (и это не предел — если выкинуть заглушку MS-DOS, программа похудеет еще на полсотни байтов).

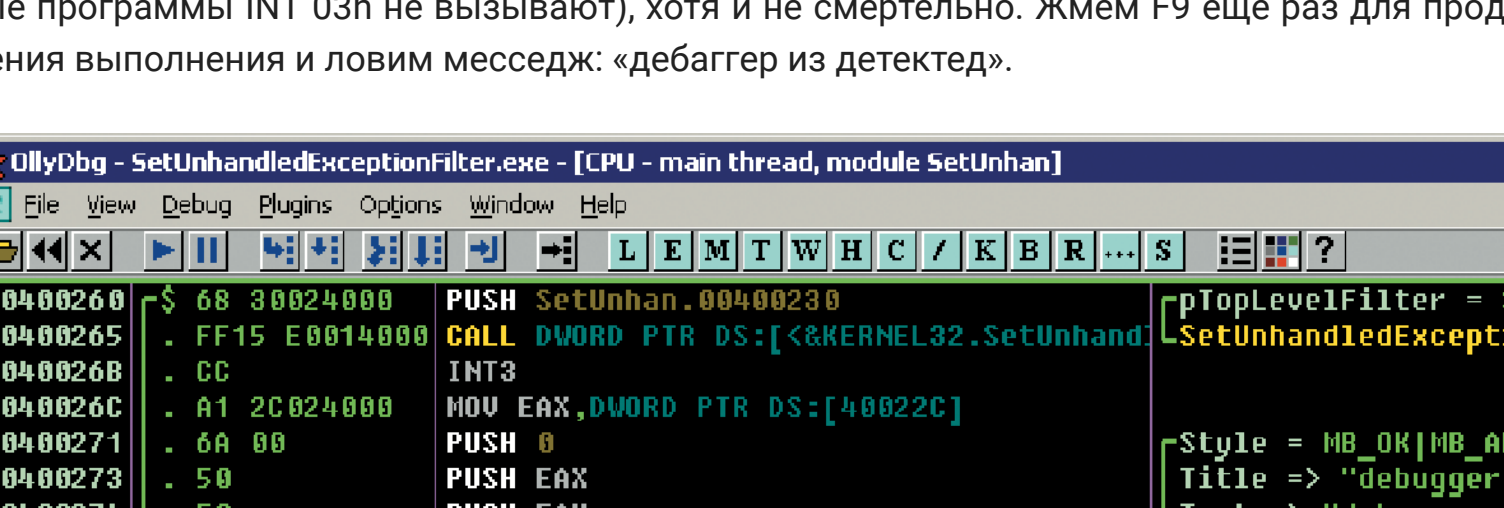
Сборка программы SetUnhandledExceptionFilter.c в командной строке среды Microsoft Visual C++

```
SET NIK=SetUnhandledExceptionFilter
cl.exe /c /Ox %NIK%.c
link.exe %NIK%.obj /FIXED /ENTRY:nezumi \
    /SUBSYSTEM:CONSOLE /ALIGN:16       \
    /MERGE:.data=.text /MERGE:.rdata=.text KERNEL32.LIB USER32.LIB
```



Запуск тестовой программы под «чистой» осью проходит нормально

Запустив файл на выполнение, мы получим сообщение, что отладчик не обнаружен. А как насчет работы под отладчиком? Загружаем `SetUnhandledExceptionFilter.exe` в «Ольгу» и давим F9 (Run). Ага! «Ольга» стопорится на INT 03h, что выглядит подозрительно (нормальные программы INT 03h не вызывают), хотя и не смертельно. Жмем F9 еще раз для продолжения выполнения и ловим месседж: «дебаггер из детектед».



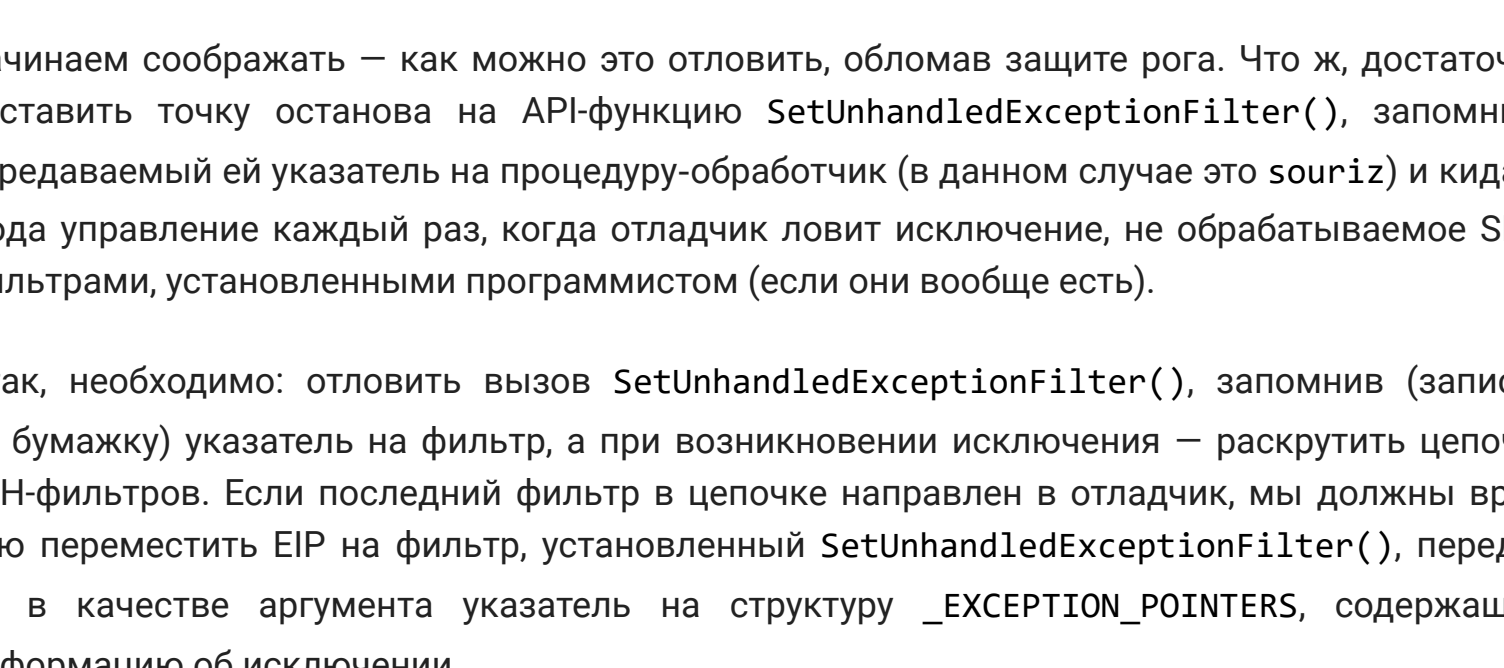
Запуск тестовой программы под «чистой» «Ольгой» (без специальных плагинов) приводит к детекции отладчика

Начинаем соображать — как можно это отловить, обломав защите рога. Что ж, достаточно поставить точку останова на API-функцию `SetUnhandledExceptionFilter()`, запомнить передаваемый ей указатель на процедуру-обработчик (в данном случае это `souriz`) и кидать сюда управление каждый раз, когда отладчик ловит исключение, не обрабатываемое SEH-фильтрами, установленными программистом (если они вообще есть).

Итак, необходимо: отловить вызов `SetUnhandledExceptionFilter()`, запомнив (записав на бумажку) указатель на фильтр, а при возникновении исключения — раскрутить цепочку SEH-фильтров. Если последний фильтр в цепочке направлен в отладчик, мы должны вручную переместить EIP на фильтр, установленный `SetUnhandledExceptionFilter()`, передав ей в качестве аргумента указатель на структуру `_EXCEPTION_POINTERS`, содержащую информацию об исключении.

Чтобы не париться, эту работу можно автоматизировать — написать свой собственный скрипт/плагин или воспользоваться уже готовым. Например, «Hide Debugger» plug-in by Asterix, который можно бесплатно скачать с OpenRCE (или другого сайта хакерской направленности).

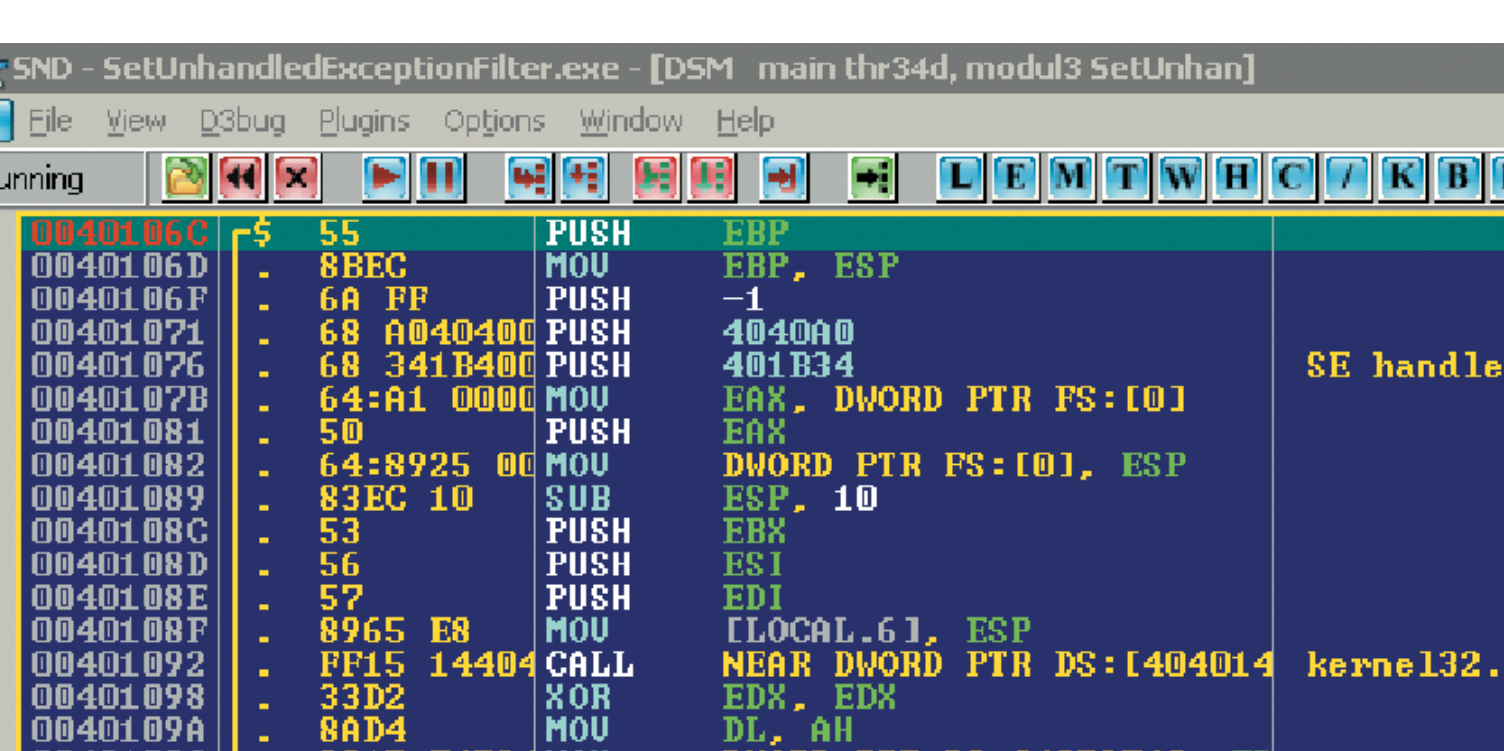
Кстати сказать, Hide_Debugger изначально входит в состав YDbg, представляющий собой популярный мод «Ольги» и, естественно, бесплатный.



«Hide Debugger» plug-in позволяет бороться с защитами, основанными на обработке необрабатываемых исключений

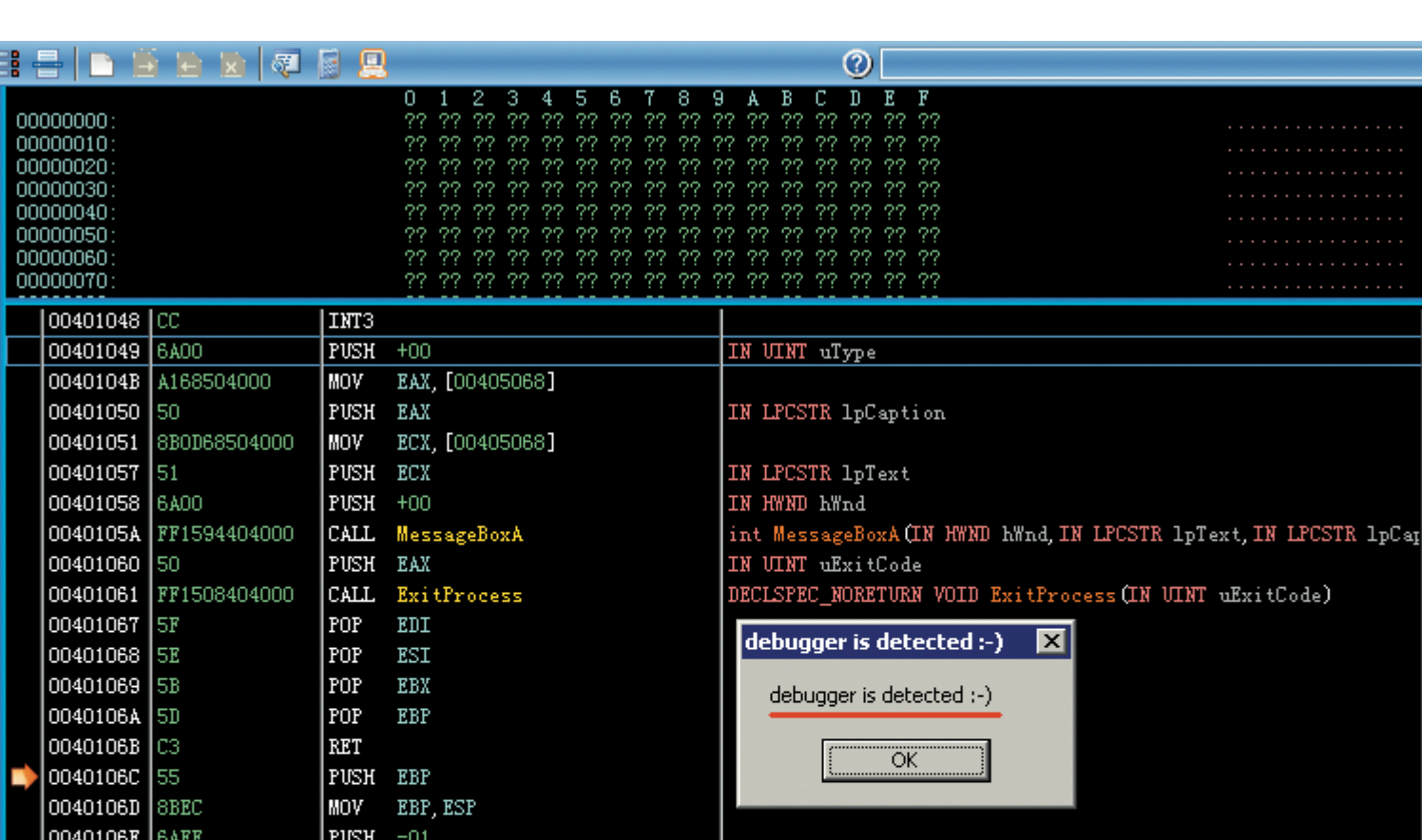
Заходим в меню Plug-ins, находим там Hide debugger. В опциях взводим галочку Unhanded exception tricks, затем нажимаем Ctrl-O. В открывавшемся диалоговом окне выбираем вкладку Exceptions и взводим галочку INT 03 breaks для передачи ломаемой программе исключений, генерируемых `INT 03h`. В конфигурации по умолчанию «Ольга», как и большинство других отладчиков прикладного уровня, молчаливо поглощает `INT 03h` — и потому никакой обработчик исключений вообще не вызывается. На самом деле INT 03h не имеет отношения к `SetUnhandledExceptionFilter`, и, если бы мы, например, генерировали исключения через обращение к нулевому указателю, последнего действия не потребовалось бы. Достаточно было бы просто взвести INT 03 breaks (подробнее о передаче исключений программе мы поговорим в следующем выпуске, а пока вернемся к нашим баранам).

Перезапускаем отладчик, чтобы изменения вступили в силу, и «пытаем» нашу тестовую программу еще раз. На экране победно отображается debugger is not detected. Открываем пиво на радостях! Мы нашли способ, как обломать этот антиотладочный прием (между прочим, весьма популярный).



«Hide Debugger» plug-in позволил спрятать отладчик от защиты

Что же касается отладчиков типа Soft-Ice и Syser, то они никак не воздействуют на поведение функции `SetUnhandledExceptionFilter()`, поскольку отладочного процесса не порождают и ведут себя, будто их здесь вообще нет. Но если, терзая нашу программу, сказать отладчику I3HERE ON и заставить его всплывать на программных точках останова (у многих хакеров эта команда пробита в строке инициализации), то отладчик «зажует» исключение, генерируемое инструкцией INT 03h. Поэтому до ломаемой программы оно вообще не дойдет, а значит, установленный фильтр не будет вызван. На экране снова появится улыбающаяся рожа, подтверждающая детект отладчика.



Отладчики Soft-Ice и Syser также «палятся», если рычажок I3HERE установлен в положение ON

Таким образом, трюк с `SetUnhandledExceptionFilter()` реально работает только с IDA-Pro, MS VC и другими примитивными отладчиками. Для всех остальных он не представляет никакой угрозы, если, конечно, заранее знать, что это такое и с чем его едят, ибо в конфигурации по умолчанию «Ольга» детектится только так.

ОТЛАДЧИК	ДОПОЛНИТЕЛЬНЫЕ ОПЦИИ	ДЕТЕКТ ОТЛАДЧИКА
		YES
OllyDbg	Hide Debugger plug-in by Asterix	YES
		YES
IDA-Pro		YES
MS VC		NO!
Soft-Ice	I3HERE ON	YES
		NO!
Syser	I3HERE ON	YES

Сводная таблица детекта отладчиков

А ты знаешь, что... «Hide Debugger» plug-in можно задетектить?

Плагин Hide Debugger от Asterix (как и большинство других плагинов подобного типа) достаточно легко задетектить, и тогда защита вновь обломает отладчик. При работе Hide Debugger изменяет адрес функции `NtQueryInformationProcess()` в таблице импорта библиотеки KERNEL32.DLL. Он записывает сюда команду перехода на свой собственный обработчик, который расположен в одном из блоков динамической памяти отлаживаемого процесса и легко находится сканированием кучи и прямым поиском плагина по сигнатурам. В этом случае функция `check_for_asterix_hide_debugger_plugin()` возвратит значение `ASTERIX_HIDE_DEBUGGER`.

Естественно, после того как Asterix перепишет свой плагин, сигнатуры уйдут лесом и данный метод детекции перестанет работать. Однако нетрудно реализовать универсальный детектор, основанный на самом факте подмены адреса `NtQueryInformationProcess()`. Достаточно распарсить таблицу импорта KERNEL32.DLL, и если адрес `NtQueryInformationProcess()` выходит за пределы модуля NTDLL.DLL (откуда эта функция, собственного, и импортируется), то, значит, мы имеем дело с «нестерильной» системой (строго говоря, это может быть не только `HIDE_DEBUGGER`, но и какой-нибудь rootkit, однако разработчиков защит подобные мелочи не волнуют).

Код детекции Hide Debugger

```
// Very dirty anti-anti-debug trick
check_for_asterix_hide_debugger_plugin()
{
    int a; int ret;int p=0; BYTE *x;
    MEMORY_BASIC_INFORMATION meminfo;

    // Asterix's Hide Debugger plugin changes addr of NtQueryInformationProcess
    // in the KERNEL32.DLL IAT to his own handler placed in the heap rwe block
    // so, to detect the plug-in, we have to find the _certain_ heap-block and
    // check signature out. It'll work until asterix doesn't rewrite the code.

    while(1)
    {
        if (!VirtualQuery((void*)p, &meminfo, sizeof(meminfo))) break;
        if ((meminfo.RegionSize==0x1000) && (meminfo.Type==MEM_PRIVATE)
        &&(meminfo.State==MEM_COMMIT)&&(meminfo.Protect==PGE_EXECUTE_READWRITE)
        &&(*((unsigned int*)p)==0x04247C83)) return ASTERIX_HIDE_DEBUGGER;
        p += meminfo.RegionSize;
    }

    // I'm too lazy to parse IAT of the KERNEL32.DLL, so I just check out
    // the address of the NtQueryInformationProcess, found in GetLogicalDrives
    // of course, I have no guarantee the code of GetLogicalDrives will be
    // unchanged in the next versions of Windows. I know to parse IAT, but...

    // I don't want to. I told you, I'm too lazzy.

    x = (BYTE*) GetProcAddress(GetModuleHandle("KERNEL32.DLL"),"GetLogicalDrives
    for (a = 0; a < 0x69; a++)
    {
        if( ( *((DWORD*)x)==0x15FFFF6A ) && ( *((WORD*)(x+8))==0x0C085)
        &&(*(DWORD*)(*(DWORD*)(x+4)) < (DWORD)GetModuleHandle("NTDLL.DLL")))
        return MessageBox(0,new, hid,0); return UNKNOWN_HIDE_DEBUGGER;
        x++;
    }

    // If we're here, well... hide-debugger plug-in is not detected :-)
    return NO_HIDE_DEBUGGER;
}
```