энциклопедия антиотладочных приемов — обработка необрабатываемых исключений выпуск #03h

крис касперски, ака мыщъх, a.k.a nezumi, a.k.a souriz, a.k.a elraton, no-email

отладчики, работающие через MS Debugging API (OllyDbg, IDA-Pro, MS VC), вынуждены мириться с тем, что отладочные процессы "страдают" хроническими особенностями поведения, "ломающих" логику программы, что (с огромной выгодной для себя!) используют защитные механизмы, ставящие хакеров в тупик. в частности, API-функция SetUnhandledExceptionFilter() под отладчиком вообще не вызывается и это вовсе не баг отладичка, а документированная фича системы!

fundamentals

Рассматривая обработку структурных исключений в предыдущем выпуске мы мельком упомянули тот факт, 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 в ходе выполнения программы, попадают в пасть первичного обработчика, расположенного в NTDLL.DLL и, в зависимости от настроек оси, либо вызывающего "Доктора Ватсона", либо выводящие 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) для разбора ситуации.

Тоже самое происходит, если программист устанавливает один или несколько обработчиков структурных исключений, но никто из них не в состоянии справится с ситуацией, вот они и передают исключение друг другу, пока оно не докатывается до системного обработчика. При желании, системный обработчик легко подменить своим. Для этого достаточно вместо ссылки на предыдущий EXCEPTION_REGISTRATION, затолкать в поле "prev" значение -1, означающее, 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" (см. рис. 1), поскольку, Windows создавалась в Америке, расположенной на противоположной стороне Земли, где люди ходят вверх ногами и потому первичный системный обработчик с их точки зрения находится на вершине пирамиды структурных исключений, в то время как русские программисты склоны рассматривать его как "основание", но это все лирика, не относящаяся к делу, а дело-то в том, что…

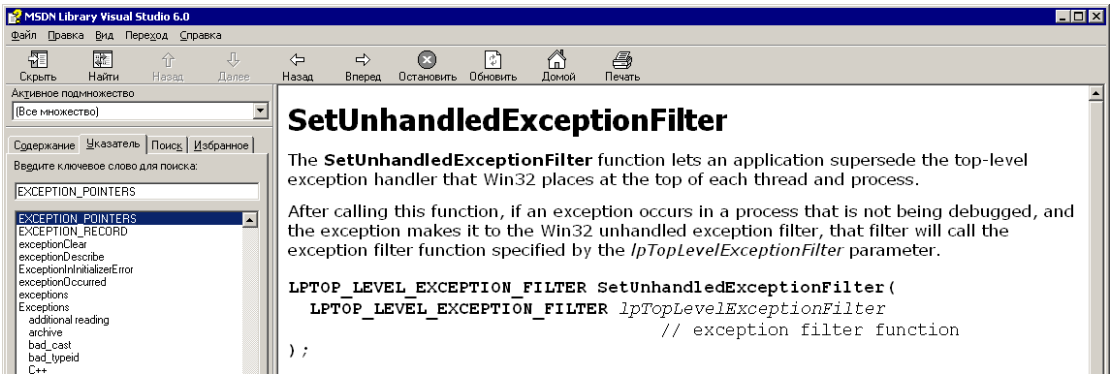

Рисунок 1 SetUnhandledExceptionFilter() на MSDN

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 и это не баг отладчика, не дефект операционной системы, а 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 функционала, например, расшифровщик какой или просто пару строк на Си, меняющих значение флага under_debuiger?***

Правильно — мы получим великолепный способ детекта отладчиков прикладного уровня!

эксперимент #6 – pro-n-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 же главное врубиться в тему и выяснить — насколько надежен этот трюк, можно ли его обойти и если да, то как?

Один из примеров реализации такого тестового стенда приведен на листинге 1.

```
#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));
}
```

Листинг 1 исходный текст программы SetUnhandledExceptionFilter.c, демонстрирующий технику использования обозначенной функции для борьбы с отладчиками

Собираем программу компилятором MS Visual C++ со следующими ключами (см. листинг 2) и получаем файл размером всего в 832 байта (и это еще не предел, — если выкинуть MS-DOS заглушку, программа похудеет еще на полста байт!)

```
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
```

Листинг 2 сборка программы SetUnhandledExceptionFilter.c в командной строке среды Microsoft Visual C++

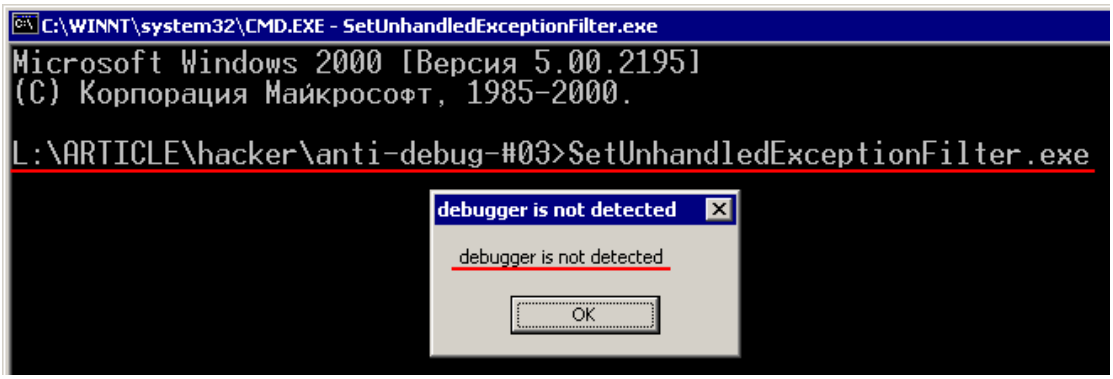

Рисунок 2 запуск тестовой программы под "чистой" осью проходит нормально

Запустив файл на выполнение (см. рис. 2), мы получим сообщение, что отладчик не обнаружен. А как на счет 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!), но не смертельно. Жмем <F9> еще раз для продолжения выполнения и… ловим месседж: "дэбугэр ис дэтэктэт" с наглой ухмыляющейся рожей (см. рис. 3).

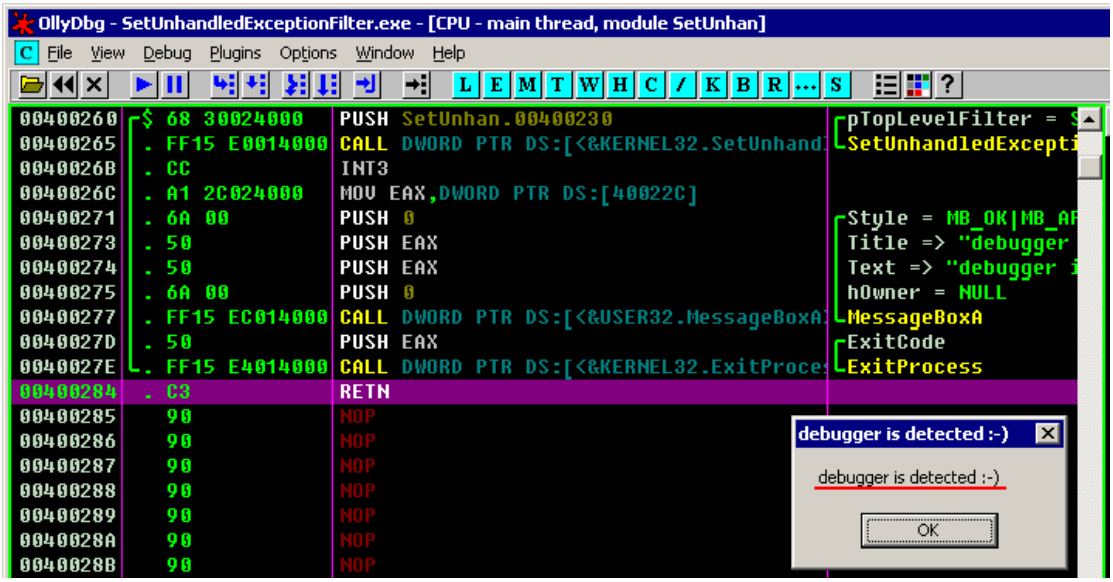

Рисунок 3 запуск тестовой программы под "чистой" Ольгой (без специальных plug-in'ов) приводит к детекции отладчика

Начинаем соображать — как можно это отловить, обломав защите рога. Да нет ничего проще — 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).

Таким образом, для обхода этого трюка необходимо: отловить вызов SetUnhandledExceptionFilter(), запомнив (записав на бумажку) указатель на фильтр, а при возникновении исключения, раскрутить цепочку SEH-фильтров и 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, написав свой собственный скрипт/плагин или воспользовавшись уже готовым, например: "**Hide Debugger**" plug-in by Asterix, который можно бесплатно скачать с OpenRCE (или другого сайта хакерской направленности): http://www.openrce.org/downloads/details/238/Hide_Debugger.

Кстати сказать, Hide_Debugger изначально входит в состав YDbg, представляющий собой популярный мод Ольги и, естественно, бесплатный: http://team-x.ru/guru-exe/Tools/Debuggers/OllyDbg/OllyDbg%20v1.10%20YDbg%20Beta.7z.

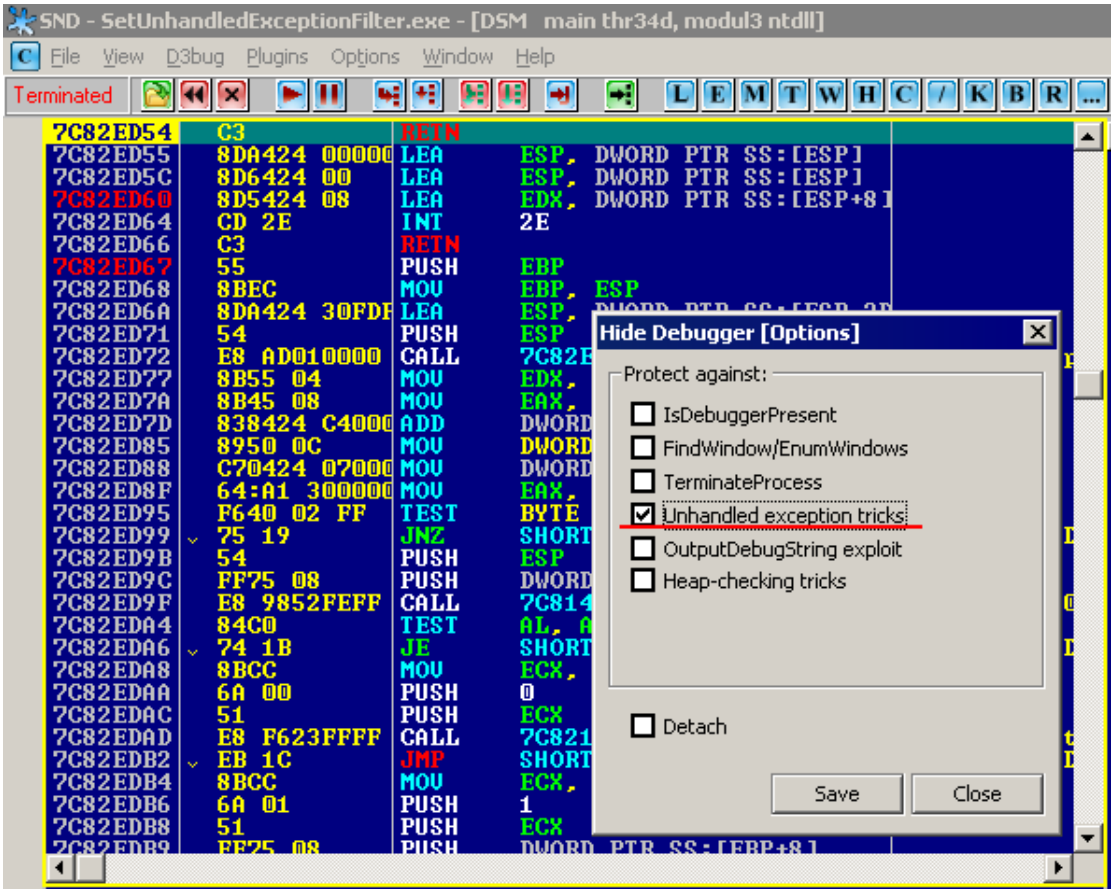

Рисунок 4 "Hide Debugger" plug-in позволяет бороться с защитами, основанными на обработке необрабатываемых исключений

Заходим в меню "Plug-ins", находим там "Hide debugger" и в опциях взводим галочку "Unhanded exception tricks" (см. рис. 4), затем нажимам <CTRL-O>, в открывавшемся диалоговом окне выбираем вкладку "Exceptions" и взводим галочку "INT 03 breaks" для передачи ломаемой программе исключений, генерируемых INT 03h, т. к. 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 никакого отношения к SetUnhandledExceptionFilter и, если бы, мы, например, генерировали исключения путем обращения к нулевому указателю, последнего действия не потребовалось — достаточно было бы просто взвести "INT 03 breaks", но 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. Ухмыляющаяся рожа исчезает и на экране победно отображается "debugger is not detected" (см. рис. 5). Открываем пиво на радостях! Мы нашли способ как обломать этот антиотладочный прием (между прочим, довольно популярный).

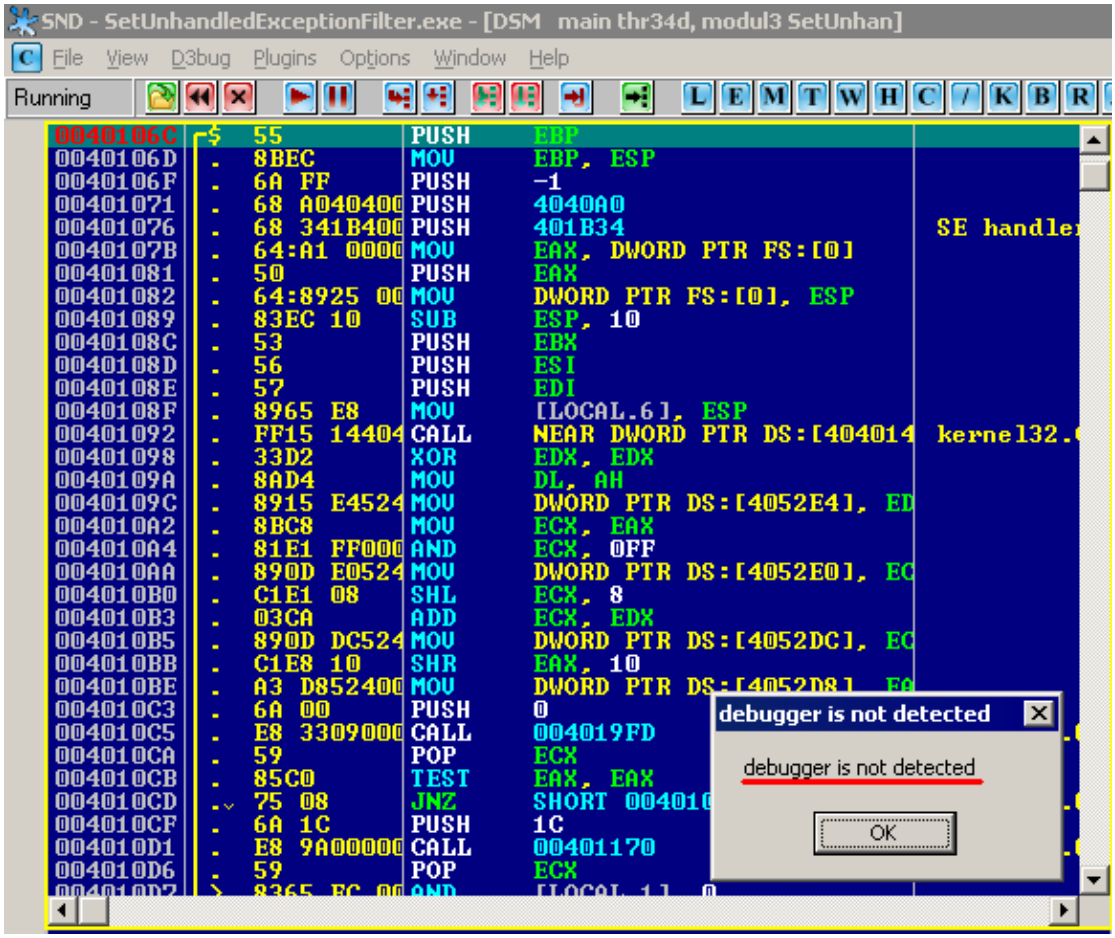

Рисунок 5 plug-in "Hide Debugger" позволил спрятать отладчик от защиты

Что же касается отладчиков типа Soft-Ice и Syser, то они вообще никак не воздействуют на поведение функции SetUnhandledExceptionFilter(), поскольку, отладочного процесса не порождают и ведут себя так, как будто их здесь вообще нет. Однако, применительно к терзаемой нами программе, — если сказать отладчику "I3HERE ON", заставляя его всплывать на программных точках останова (у многих хакеров эта команда пробита в строке инициализации), то отладчик "зажует" исключение, генерируемое инструкцией INT 03h и потому до ломаемой программы оно вообще не дойдет, а, значит, установленный фильтр не будет вызван и на экрана снова появится улыбающаяся харя, подтверждающая детект отладчика (см. рис. 6)

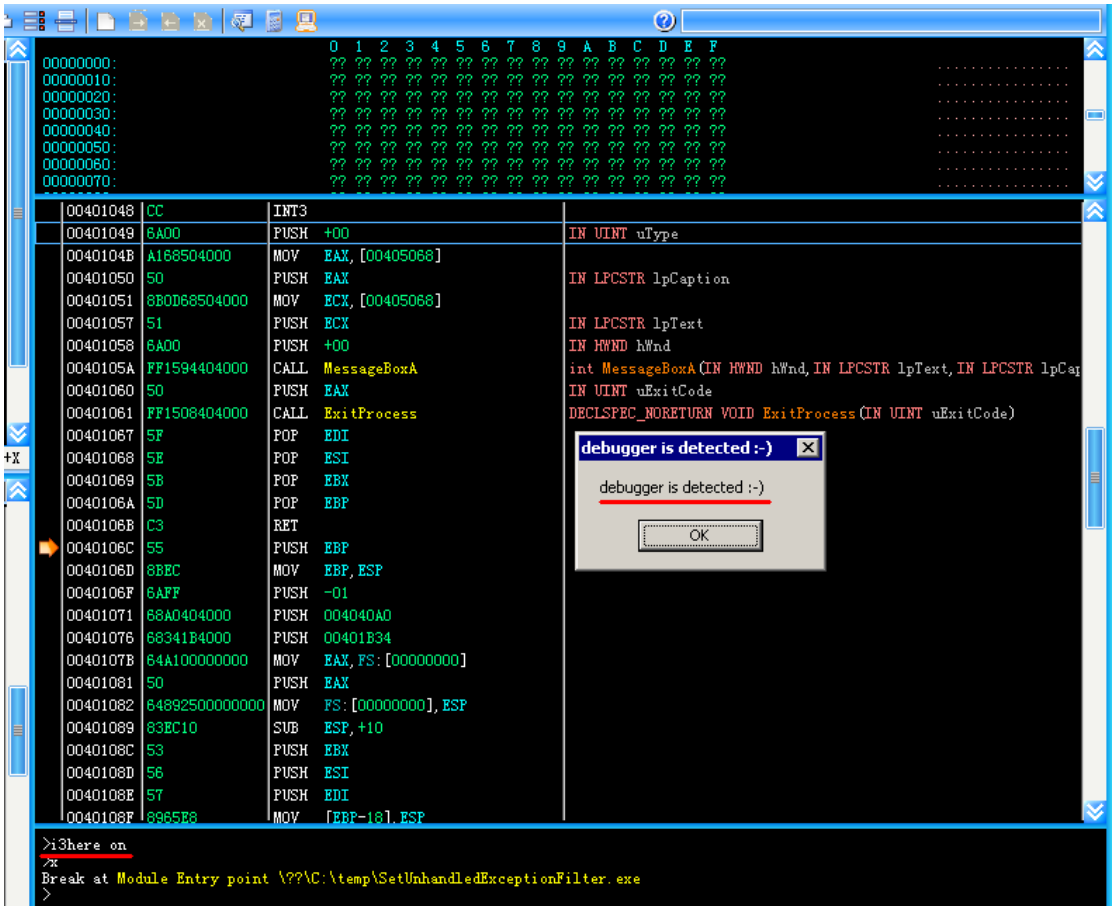

Рисунок 6 отладчики Soft-Ice и Syser так же "паляться", если рычажок "I3HERE" установлен в положение "ON"

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 умолчании, Ольга детектиться только так.

>>> врезка: а вы знаете, что… Hide Debugger plug-in можно задетектить?

Плагин "Hide Debugger" от Asterix'a (как и большинство других plain-in'ов подобного типа) достаточно легко задетектить и тогда защита вновь обломает отладчик. В процессе своей работы "hide debugger" изменяет адрес функции **NtQueryInformationProcess()** в таблице импорта библиотеки KERNEL32.DLL, записывая сюда команду перехода на свой собственный обработчик, расположенный в одном из блоков динамической памяти отлаживаемого процесса, который легко находится сканированием кучи и прямым поиском plug-in'а по сигнатурам. В этом случае функция check_for_asterix_hide_debugger_plugin(), приведенная в листинге 2, возвратит значение ASTERIX_HIDE_DEBUGGER.

Естественно, после того как Asterix перепишет свой plug-in, сигнатуры уйдут лесом и данный метод детекции перестанет работать, однако, нетрудно реализовать универсальный детектор, основанный на самом факте подмены адреса NtQueryInformationProcess(), для чего достаточно распарсить таблицу импорта KERNEL32.DLL и, если адрес NtQueryInformationProcess() выходит за пределы модуля NTDLL.DLL (откуда эта функция, собственного говоря, и импортируется), то, значит, мы имеем дело с "не стерильной" системой, возвращая значение UNKNOWN_HIDE_DEBUGGER (строго говоря, это может быть и не только HIDE_DEBUGGER, но и какой ни будь rootkit, но разработчиков защит подобные мелочи не волнуют).

```
// 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 hidedebugger 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;
}
```

Листинг 3 код детекции "Hide debugger" plug-in'a

>>> врезка сводная таблица детекта отладчиков

отладчик	дополнительные опции	детект отладчика
OllyDbg		YES
	Hide Debugger plug-in by Asterix	YES
IDA-Pro		YES
MS VC		YES
Soft-Ice		NO!
	I3HERE ON	YES
Syser		NO!
	I3HERE ON	YES

>>> врезка: а вы знаете, что… 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, теряя 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 в листинг 1. 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 данная ошибка уже исправлена, однако, учитывая, что 2.x все еще находится в стадии разработки, основным инструментом хакеров остается Ольга 1.10 с этим багом на борту.