энциклопедия антиотладочных приемов — самотрассировка и прочие головоломки (выпуск #04h)

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

сегодня мы будем ломать мыщъхиный crack-me, напичканный антиотадочными приемами, основанных на особенностях обработки исключений отладчиками и на ошибках в debug engine, о которых мыщъх расскажет по ходу дела, демонстрируя интимные подробности основных хакерских инструментов — ольги, иды, syser'a, x86emu и прочих отладчиков

введение

Разгадывать загадки намного интереснее, чем читать готовые решения. А потому пока еще не поздно оторвемся от статьи и попробуем расковырять JohoR crack-me (который можно взять с диска, прилагаемого к журналу, или скачать из мыщъиного репрозитория на OpenRCE: https://www.openrce.org/repositories/users/nezumi/crackme-jhr.zip). В подсказку не заглядывать! Исходный текст не смотреть! Впрочем... сам по себе исходный текст (даже с учетом всех содержащихся в нем комментариев) ничего не проясняет и совершенно не объясняет как же его отлаживать?!

Здесь отсутствует шифровка, самомодификация и прочие приемы, ослепляющие статический анализ. Дизассемблер выдает аккуратный листинг, каждая машинная команда которого абсолютно понятна, но… результат действия всей программы в целом очень трудно предсказуем и требует довольно глубоких знаний устройства процессора и операционной системы. Поистине танталовы муки! Какой-то несчастный десяток машинных инструкций (ядро crack-me) отделяет нас от победы! Что ж! Тем большее наслаждение испытываешь от взлома! Ну, а если взломать никак не получается, на этот случай мыщъх приводит развернутое объяснение.

что мы будем ломать

Исходный текст JohoR crack-me, написанного на MS VC, приведен в листинге 1 и после компиляции это чудо мыщъхиной инженерии занимает всего 832 байта, большая часть из которых приходится на PE-заголовок, который, конечно, было бы можно ужать, программируя в hex-кодах, но это же сколько труда потратить надо! А так — файл легко компилируются штатными утилитами от Microsoft.

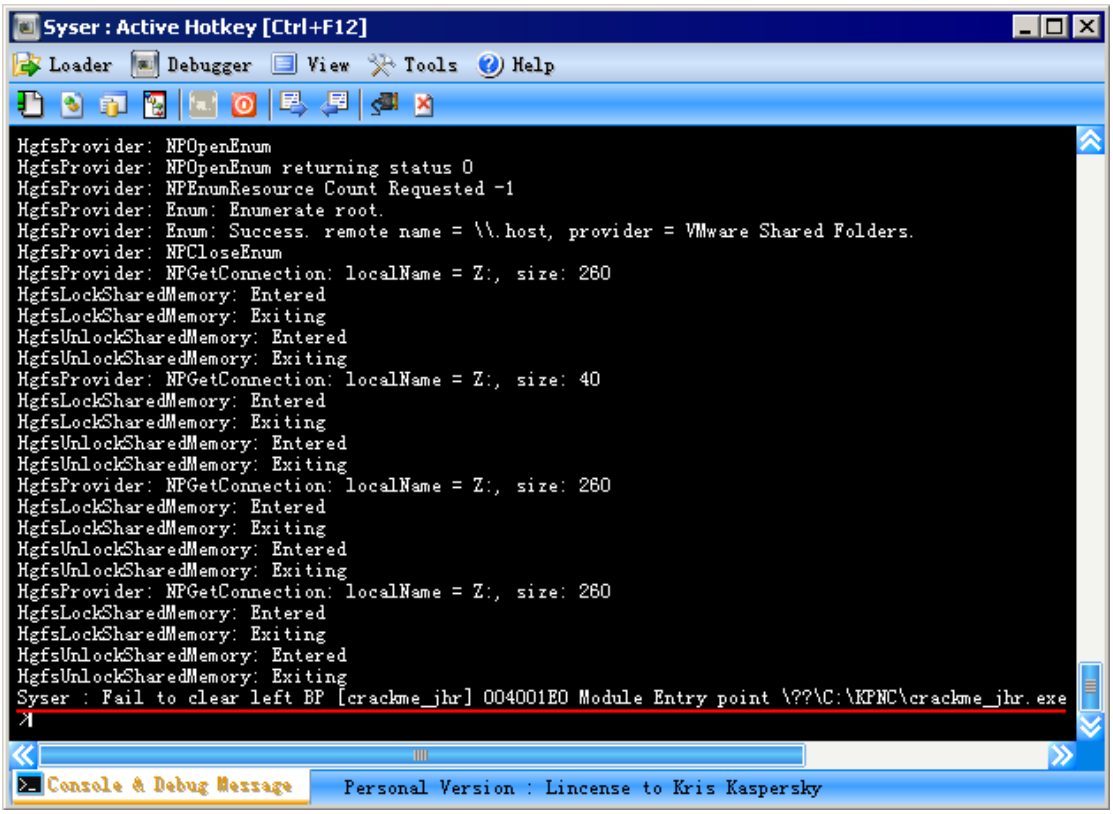

Рисунок 1 Syser вообще отказывается грузить JohoR crack-me в release build'e

Кстати, о компиляции. По многочисленным просьбам трудящихся, мыщъх отказался от командных файлов и перешел на макак (в смысле .mak), обрабатываемых утилитой NMAKE, входящей в комплект поставки MS Visual Studio. "NMAKE /f crackme_jhr.mak" собирает релиз, а NMAKE /f "crackme_jhr.mak" CFG="crackme_jhr - Win32 Debug" — отладочную версию, только все равно отладить ее с помощью MS Visual Studio не удастся — нет смысла даже пытаться.

Так же поддерживается сборка и из IDE – достаточно открыть макаку и сделать build. Тупая студия всегда ищет скомпилированный файл в каталогах \Debug и \Release, тогда как мыщъх создает его в текущей директории, поэтому запуск файла непосредственно из IDE невозможен (хотя, возможно, что в последних версиях MS уже пофиксила этот косяк).

```
#include <windows.h>

int count; char str[]="0123456789ABCDEF!";
__declspec(naked) nezumi()
{
        __asm{
        ; //int 03                              ; // for soft-ice
        xor eax, eax                            ; // eax := 0
        mov ebx, fs:[eax]                       ; // old SEH
        pushfd                                  ; // save EFLAGS
        ;//-[new seh]-----------------------------------------------------------;
        push offset l1                          ; // handler proc
        push -1                                 ; // the last handler in the chain
        mov fs:[eax], esp                       ; // assign the new handler
        ;//-[hacker time]--------------------------------------------------------;
        xor eax,[eax]                           ; // <-- ACCESS VIOLATION
        ;//-[set TF bit]----------------------------------------------------------;
        push -1                                 ; // TF := 1
        xor eax,[eax]                           ; // <-- ACCESS VIOLATION
        popfd                                   ; // EFLAGS := 00244ED7h
        ;//-[TRACE-ZONE]----------------------------------------------------------;
        mov eax, [eax]                          ; // <-- ACCESS VIOLATION
        nop                                     ; // <-- INT 01
        ud2                                     ; // <-- ILLEGAL INSTRUCTION
        nop                                     ; // <-- INT 01
        nop                                     ; // <-- INT 01
        int 03                                  ; // <-- INT 01
        jmp end_of_line                         ; // :-- to exit -->
```

```
        ;//-[seh handler]---------------------;---------------------------------;
        l1: mov eax, [esp + 04h]              ; // *EXCEPTION_RECORD
        l2: mov edx, [esp + 0Ch]              ; // EDX -> ContextRecord
        mov eax, [eax]                        ; // EXCEPTION CODE
        cmp eax, 0C000001Dh                   ; // ILLEGAL INSTRUCTION
        jz x2                                 ; // X-->

        cmp eax, 080000003h                   ; // INT 03
        jz x1                                 ; // -- skip 1 byte -->

        cmp eax, 0C0000005h                   ; // ACCESS VIOLATION
        jnz set_tf_bit                        ; // -- don't skip -->

        x2:inc dword ptr [edx+0B8h]           ; // skip one byte
        x1:inc dword ptr [edx+0B8h]           ; // skip one byte

        set_tf_bit:                           ; // <--X
        cmp dword ptr [edx + 0B8h], offset end_of_line
        jae end_of_handler                    ; // dont set TF-bit _outside_ trace-zone
        or dword ptr [edx+0C0h],100h          ; // <---- set TF-bit _inside_ trace-zone
        end_of_handler:xor eax,eax            ; // EXCEPTION_CONTINUE_SEARCH
        inc [count]                           ; // EXCEPTION COUNT
        ret                                   ; // end of the handler
        ;//-[exit]-----------------------------------------------------------------;
        end_of_line:mov fs:[eax],ebx          ; // restore the old SEH
        sub esp, 8                            ; // restore the stack
        popfd                                 ; // restore the flags
        }

        // print EXCEPTION COUNT
        count = str[(count>0x10)?0x10:count]; MessageBox(0,&count,"JohoR",MB_OK);
        ExitProcess(0);
}
```

Листинг 1 полный исходный текст JohoR crack-me

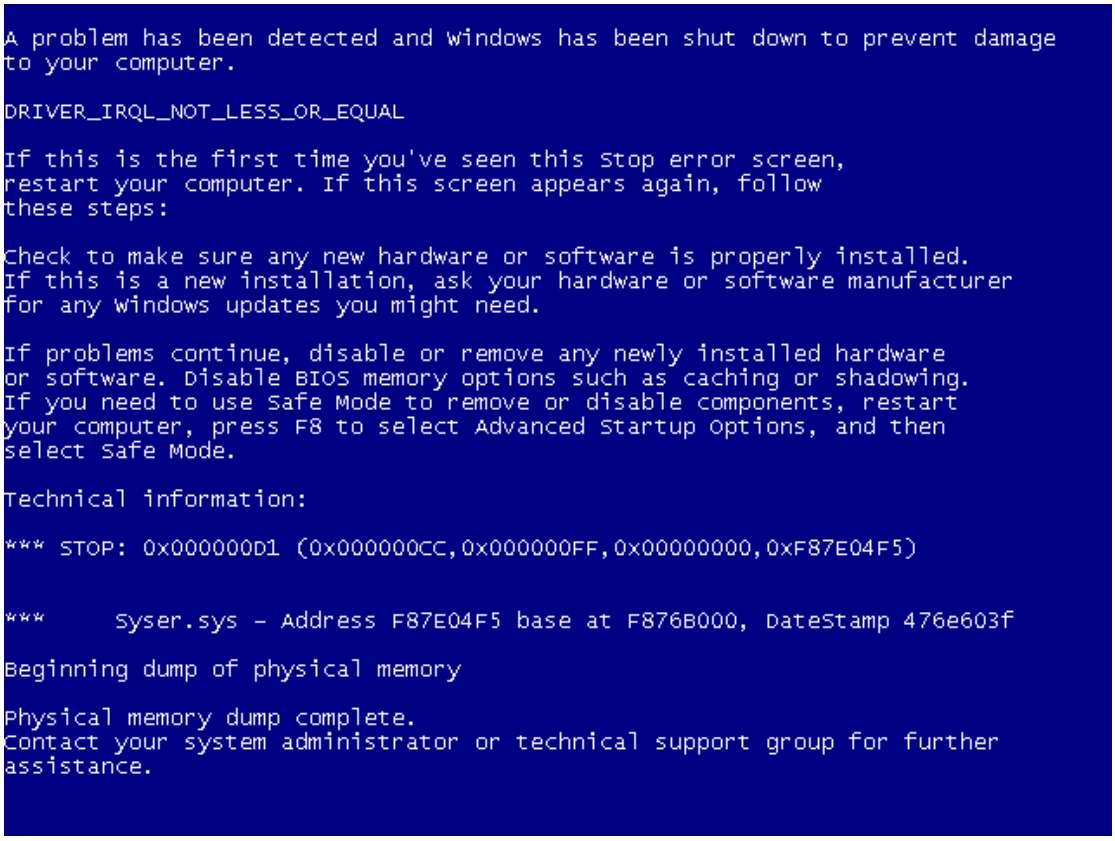

Рисунок 2 попытка отладки дебежного билда JohoR crack-me под Syser'ом — отладчик в панике, система — в ауте, хакер — на полной измене

алгоритм

Первые три команды сохраняют указатель на текущую (системную) SEH-запись в регистре EBX и заталкивают в стек флаги процессора, попутно обнуляя EAX. Следующие три команды устанавливают новый SEH-обработчик, находящийся по смещению l1, замыкая SEH-цепочку термирующим указателем -1 (FFFFFFFFh), сигнализирующим системе о том, что данный обработчик — последний. Вот такой маленький трюк (почему-то большинство остальных хакеров добавляют свой обработчик к цепочке уже существующих, хотя передавать им управление все равно не собираются, так зачем же тогда усложнять код?).

Рисунок 3 результат работы JohoR crack-me при запуске без отладчика

Сразу же после установки SEH-обработчика выполняется команда XOR EAX, [EAX], "выбрасывающая" исключение доступа типа ACCESS VIOLATION. Операционная система ловит его и передает управление на метку l1, с кодом C0000005h, расположенном в двойном слове по адресу [ESP + 04h]. Обработчик видит, что это ACCESS VIOLATION и зная, что его вырабатывает инструкция XOR EAX, [EAX], лезет в регистровый контекст, увеличивая значение EIP на два байта — sizeof(XOR EAX, [EAX]), поскольку, ACCESS VIOLATION представляет собой fault, т.е. в момент генерации исключения, регистр EIP указывает на начало возбудившей его машинной команды и при выходе из обработчика, процессор будет выполять XOR EAX, [EAX] снова и снова, пока мы либо не изменим EAX, так чтобы он указывал на валидную область памяти, либо же не увеличим значение EIP, переходя к выполнению следующей машинной команды, что мы и делаем, попутно увеличивая счетчик вызова исключений (count) на единицу.

Зачем мы это делаем?! А вот зачем — при пошаговой трассировке программы, когда взведен TF-бит, операционная система следом за ACCESS VIOLATION генерирует SINGLE STEP, в результате вместо одного исключения мы получаем целых два и SEH-обработчик под отладчиком вызывается _дважды_ позволяя программе обнаружить, что ее ломает злобный хакер! Отладчики MS VC, MS WinDbg, Soft-Ice (и ряд других) давят SINGLE STEP-исключение, сбрасывая флаг трассировки через контекст, а вот Ольга 1.1x об этом не заботится. Ошибка была исправлена только в версии 2.x (все еще находящейся в разработке). Подобнее об этом мыщъх рассказывает в своем блоге: http://souriz.wordpress.com/2008/05/09/bug-in-olly-windows-behavior-and-peter-ferrie ("bug in Olly, Windows behavior and Peter Ferrie").

Три следующих машинных команды взводят флаг трассировки (на самом деле, флаг трассировки взводится с помощью всего двух команд — PUSH -1/POPFD, а XOR EAX, [EAX], что расположена между ними, на самом деле вставлена для борьбы с одним экспериментальным отладчиком, что "отлавливает" инструкцию POPFD и, если ломаемая программа взводит бит трассировки, отладчик врубает модуль эмуляции, но если управление на POPFD передается посредством правки регистрового контекста, отладчик это не просекает, точнее, это он раньше не просекал, а сейчас ошибка исправлена).

POPFD выталкивает -1 (FFFFFFFFh) из стека, устанавливая все процессорные флаги в единицу, то есть, конечно, далеко не все флаги, а только те, которые позволено модифицировать прикладной программе, о чем, кстати говоря, "догадываются" далеко не каждый эмулирующий отладчик, а потому значение EFLAGS под "живым" процессором и, например, x86emu очень сильно отличается. Но x86emu, в общем-то, и не подряжался эмулировать все флаги процессора. В принципе, здесь можно вставить проверку — если EFLAGS не равняется 00244ED7h, то это одно из двух — либо нас ломают на эмуляторе, либо это какой-то очень левый процессор. Впрочем, поскольку, достойных эмулирующих отладчиков под x86 все равно нет, подобная проверка лишена смысла.

Но не будем отвлекаться. Флаг трассировки успешно взведен и по завершению команды, следующей за POPFD, процессор генерирует пошаговое исключение. А этой командой является… наша старая знакомая XOR EAX, [EAX], "выбрасывающая" ACCESS VIOLATION, предшествующий пошаговому исключению. То есть, в данном случае, система генерирует два вполне законных исключения, вот только большинство отладчиков ошибочно

принимают исключение, сгенерированное программой, за свое собственное и давят его, в результате чего SEH-обработчик вызывается на один раз меньше. Ольга 1.1x данную ситуацию обрабатывает вполне правильно (точнее, никак не обрабатывает, тупо передавая все исключения программе), а вот исправленная Ольга 2.x путается в исключениях и "давит" лишнее (с ее точки зрения) пошаговое исключение. Другими словами, под Ольгой 1.1x SEH-обработчик вызывается на один раз больше, чем под "живым" процессором (с учетом первой команды MOV EAX, [EAX]), а под Ольгой 2.x — на один раз меньше. Красота!!! И какую же версию нам выбирать?! Что касается Soft-Ice (и некоторых других отладчиков), он "кушает" все пошаговые исключения, генерируемые отлаживаемой программой, обламывая самотрассировку и потому SEH-обработчик вызывается только на MOV EAX, [EAX] — как следствие: счетчик вызовов count оказывается намного меньше, чем ожидает защита, сразу же понимающая с кем она имеет дело.

Команда NOP "честно" генерирует пошаговое исключение (ведь бит трассировки взведен!), но… Soft-Ice его поглощает. Остальные отладчики (типа Ольги и IDA-Pro) хотя бы можно настроить на отдачу пошаговых исключений ломаемой программе, причем, IDA-Pro 5.2 предложит сделать это автоматически, в то время как Ольга требует ручной настройки (вкладка "Exceptions" в опциях отладчика).

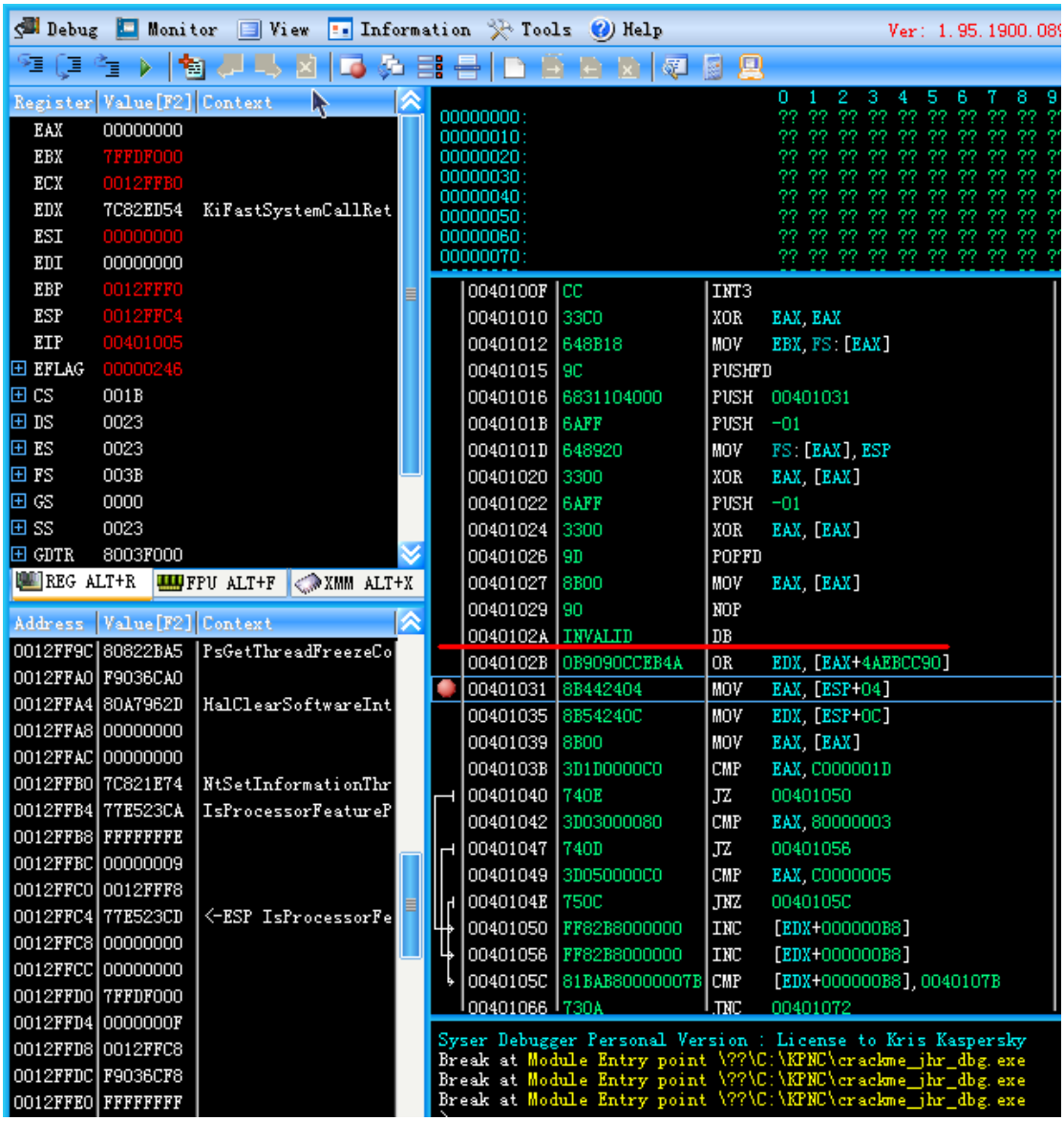

Рисунок 4 реакция Syser'a на машинную команду UD2

Инструкция UD2 генерирует исключение типа ILLEGAL INSTRUCTION, перехватываемое Soft-Ice и не отдаваемое отлаживаемой программой вплоть до отдачи команды

"faults off". Что же касается Syser'a, то он такой команды вообще не знает, "обзывая" ее как "DB" (объявить байт) и неверно дизассемблируя весь последующий код (что не покажется удивительным, если вспомнить, что UD2 – двухбайтовая команда).

Пара последующих NOP'ов не делает ничего, кроме генерации пошагового исключения, особенность обработки которого мы обсуждали двумя абзацами выше. Так зачем же тогда они нужны?! Все очень просто — разные отладчики имеют разные баги, "съедая" различное количество исключений…Ага!!! Правильно! Данный crack-me определяет тип отладчика по значению count, уникальность которого обеспечивается соотношением команд, генерирующих свои собственные исключения, к общему количеству трассируемых инструкций. Если убрать NOP'ы, crackme продолжит детектить активную отладку, но уже не сможет определить какой именно отладчик используется хакером.

Рисунок 5 результат работы JohoR crack-me при запуске под IDA-Pro 4.7

Команда INT 03h так же вставлена неспроста, а с умыслом. Если даже настроить отладчик на отдачу INT 03h ломаемой программе, наличие INT 03h существенно затрудняет отладку. Если бы INT 03h не было, то чтобы быстро выбраться из глубин системного обработчика исключений назад к ломаемой программе, достаточно покрыть трассируемый блок программными точками останова (в Ольге для этого достаточно на каждой команде нажать F2). Программные точки останова представляют собой однобайтовую инструкцию CCh, легко обнаруживаемую подсчетом контрольной суммы и "разваливающую" самомодифицирующий код. Впрочем, ни того, ни другого в crack-me нет, а есть только INT 03h. Чисто теоретически, отладчик может и должен отличать свои собственные INT 03h от чужих, отдавая программе то те исключения, которые она сама же и сгенерировала. Но на практике отладчики путаются. Ольга, настроенная на отдачу INT 03h ломаемой программе, при установке программной точки останова поверх INT 03h вообще циклится, вынуждая хакера применять аппаратные точки останова (которых всего четыре) или точки останова на регион памяти, реализованные через подмену атрибутов страниц, что так же легко обнаруживается.

Кстати, с точки зрения процессора, INT 03h генерирует trap, а не fault, то есть регистр EIP в момент генерации исключения смотрит на команду, следующую за INT 03h, которой в данном случае является двухбайтовая инструкция JMP END_OF_LINE. В чем же подвох?! А в том, что SEH-обработчик отлавливает BREAKPOINT-исключение (соответствующее коду 080000003h) и увеличивает значение EIP на единицу. Стоп!!! Выходит, что управление передается в середину инструкции JMP END_OF_LINE?! Какой хитрый прием против дизассемблера!!! Гм, вот только непонятно... первый байт опкода JMP SHORT равен EBh, второй — представляет относительное смещение целевого перехода. И чтобы оно соответствовало осмысленной машинной инструкции, необходимо, чтобы метка END_OF_LINE располагалась на определенном смещении от команды JMP. А в cracm-me между ними расположен SEH-обработчик. Выходит, если его изменить, то crack-me сразу перестанет работать?! Такая хитрая защита исходных текстов от изменения!!!

И чего только со страху не покажется… Да, в руководствах от Intel черным по белому написано, что BREAKPOINT это trap, а не fault, вот только SEH-обработчик вызывается не процессором, а операционной системой. Той, что написана компаний Microsoft. А Microsoft-way умом не понять. Ну чем можно объяснить, что она подменяет процессорный контекст, умышленно уменьшая EIP на единицу?! Причем, парни из Microsoft впопыхах забыли, что BREAKPOINT может генерироваться как опкодом CCh, так и CDh 03h, а потому если внедрить CDh 03h в программу, и никак ее не обрабатывать, то после выхода из исключения, регистр EIP будет смотреть на опкод 03h, соответствующий команде ADD чего-то там. Допустим, за CDh 03h следует CCh (еще один INT 03h, только слегка другой), тогда процессор выполнит опкод 03h CCh — ADD ECX, ESP. Вот и попробуй догадаться об этом при дизассемблировании!!!

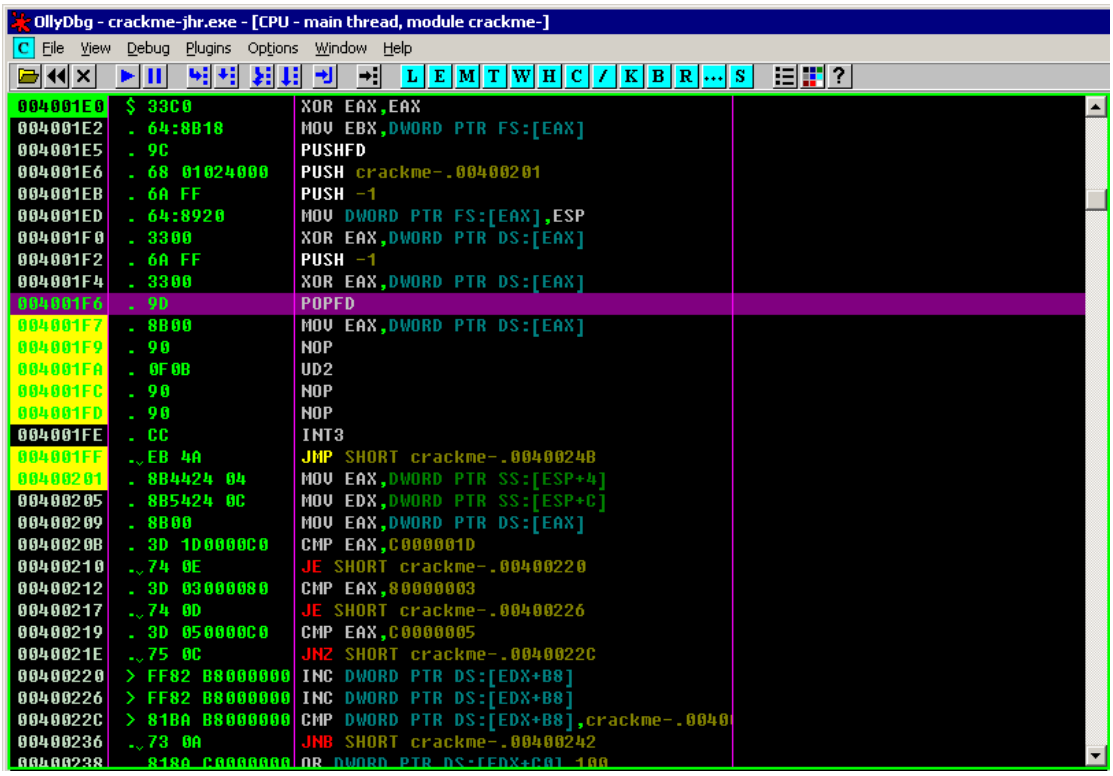

Рисунок 6 установка программных точек останова внутри "горячей" зоны

Наконец, команда JMP END_OF_LINE выводит код из зоны трассировки. Программа восстанавливает прежний SEH, выталкивает из стека флаги и распечатывает значение счетчика исключений, после чего завершает свое выполнение вызовом функции ExitProcess(0).

заключение

Так как же все-таки ломают такие программы? И каким отладчиками?! В случае статического кода (к которому относится данный crack-me) проблема решается установкой точки останова _за_ пределами "горячей" зоны, где происходит выброс исключений, с прогоном их на живом процессоре, то есть _без_ пошаговой трассировки. Если же нам жизненно необходимо подсмотреть значение некоторых регистров или ячеек памяти внутри "горячей" зоны — на них устанавливается аппаратная точка останова.

Динамический код (упакованный, зашифрованный, самомодифицирующийся) заломать намного сложнее, поскольку, нам реально необходимо прогнать его через пошаговый трассировщик, с которым и борется защита, причем, весьма эффективно борется. BOCHS (бесплатная виртуальная машина со встроенным отладчиком) единственный разумный выбор, но сколько грузится на нем Windows лучше не говорить.