энциклопедия антиотладочных приемов — трассировка — в погоне на TF, игры в прятки выпуск #01h

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

обзор анти-отладочных приемов мы начнем с базовых понятий, фундаментальным из которых является трассировка, то есть пошаговое исполнение кода. сначала мы узнаем зачем нужна трассировка, как и в каких целях она используется отладчиками, по каким признаком защитный код может определить, что его трассируют и какие примочки к отладчикам позволят хакеру избежать расправы

введение

Уже давно никто не трассирует программы от начала и до конца — уж слишком это утомительно и непродуктивно. Основное оружие хакера — точки останова на память, API-функции, Windows-сообщения, API-шпионы и прочие, прочие, прочие… однако, не стоит _полностью_ списывать трассировку со счетов, она и сейчас живее всех живых!

Запутанные участки кода, ответственные за проверку серийного номера, ключевого файла или расшифровку программы, довольно часто прогоняются отладчиком в пошаговом режиме, кроме того, отладчик может "негласно" задействовать трассировку для выполнения некоторых операций. В частности, в OllyDbg установка точки останова на команду и/или диапазон EIP-адресов как раз и реализуются через трассировку. Ее же используют достаточно многие плагины, например, популярный FindString, осуществляющий поиск заданной строки в регистрах, трактуя их как указатели.

Распаковщики упакованных файлов (особенно универсальные) активно используют трассировку для освобождения от упаковщика и восстановления оригинальной точки входа в программу (Original Entry Point или, сокращенно, OEP).

Защита, умело препятствующая трассировке, существенно затрудняет взлом программы, хотя и не делает его невозможным, поскольку, на каждый анти-отладочный болт с хитрой резьбой уже давно придуман свой анти-анти-отладочный ключ.

трассировка в x86-процессорах

Если TF-флаг, хранящийся в регистре EFLSGS (и гнездящийся в 8'ом бите, считая от нуля), взведен, то после исполнения каждой команды процессор генерирует прерывание INT 01h или EXCEPTION_SINGLE_STEP (80000004h) — как его "обозвали" разработчики Windows. Исключение составляют команды, модифицирующие регистр SS (селектор стека), маскирующие прерывание на выполнение следующей команды. На этот шаг разработчики процессоров пошли потому, что в коде достаточно часто встречаются конструкции вида MOV SS, new_ss/MOV ESP, new_ESP. Как нетрудно сообразить, если прерывание произойдет после того, как новый селектор стека уже обозначен, а указатель вершины стека еще не инициализирован, мы получим неопределенное поведение системы, ведущее к краху (а ведь существует команда LSS, одним махом загружающая и SS и ESP, но она не относится к числу самых популярных).

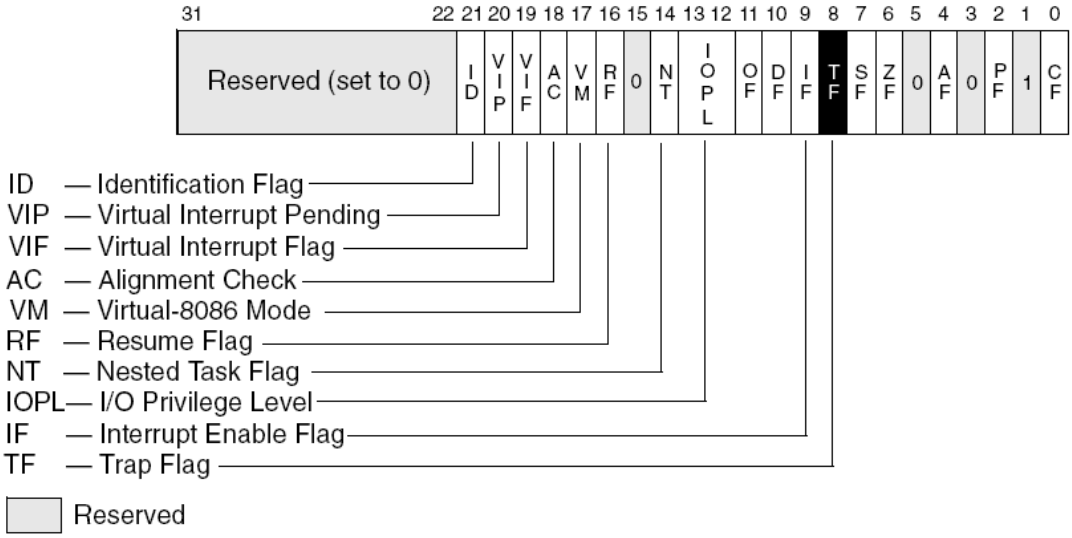

Рисунок 1 TF-флаг в регистре флагов EFLAGS

Простейший способ обнаружения трассировки состоит в чтении регистра флагов (EFLAGS) и проверке состояния бита TF. Если он не равен нулю — нас кто-то злостно трассирует. С прикладного уровня прочитать содержимое регистра флагов можно самыми разными способами: командой PUSHFD, заталкивающей флаги в стек, генерацией исключения (при которой SEH-обработчику передается контекст потока вместе со всеми регистрами, включая регистр флагов), наконец, контекст можно получить API-функцией GetThreadContext.

Сегодня мы будем говорить лишь об первом способе — команде PUSHFD. При кажущейся незамысловатости, она скрывает целый пласт хитростей, известных далеко не всякому хакеру.

эксперимент N #1 – "чистый" PUSHFD

Напишем следующую несложную программку (см. листинг 1), заталкивающую в стек регистр флагов через PUSHFD и тут же выталкивающую ее обратно в EAX для тестирования значение бита TF.

```
char yes[]="debugger is detected :-)";
char noo[]="debugger is not detected";

nezumi()
{
        char *p=noo;                    // презумпция невиновности is on ;-)
        __asm
        {
                ; int 03                        ; для отладки
                pushfd                          ; сохраняем флаги в стеке, включая и TF
                pop eax                         ; выталкиваем сохраненные флаги в eax
                and eax, 100h                   ; проверяем состояние TF-бита
                jz not_under_dbg                ; если TF не взведен, нас не трассирует…
                mov [p], offset yes             ; …ну или мы не смогли это обнаружить ;)
        not_under_dbg:
        }

        MessageBox(0, p, p, MB_OK);
}
```

Листинг 1 простейшая программа TF-0x0-simple.c для обнаружения трассировки через PUSHFD

Откомпилируем ее следующим образом (см. листинг 2). Все это шаманство потребовалось: а) чтобы убить стартовый код и программа сразу же начиналась с интересующей нас функции nezumi(); б) чтобы сократить размер программы, равный в данном случае 768 байтам.

```
cl.exe /c /Ox  /Os /G6 TF-0x0-simple.c
link.exe TF-0x0-simple.obj /ENTRY:nezumi /MERGE:.rdata=.text
               /ALIGN:16 /DRIVER /FIXED /SUBSYSTEM:CONSOLE KERNEL32.LIB USER32.lib
```

Листинг 2 сборка простейшей тестовой программы

Не обращая внимания на ругательство линкера "warning LNK4078: multiple ".text" sections found with different attributes (40000040)" запустим программу, убедившись, что в отсутствии она честно говорит "debugger is not detected", а теперь загрузим ее в MS VC dbg и будем трассировать (клавиша <F11>) пока не достигнем первого call'a (им будет MessageBox). Ага! "debugger is detected :-)"! Цель достигнута!!!

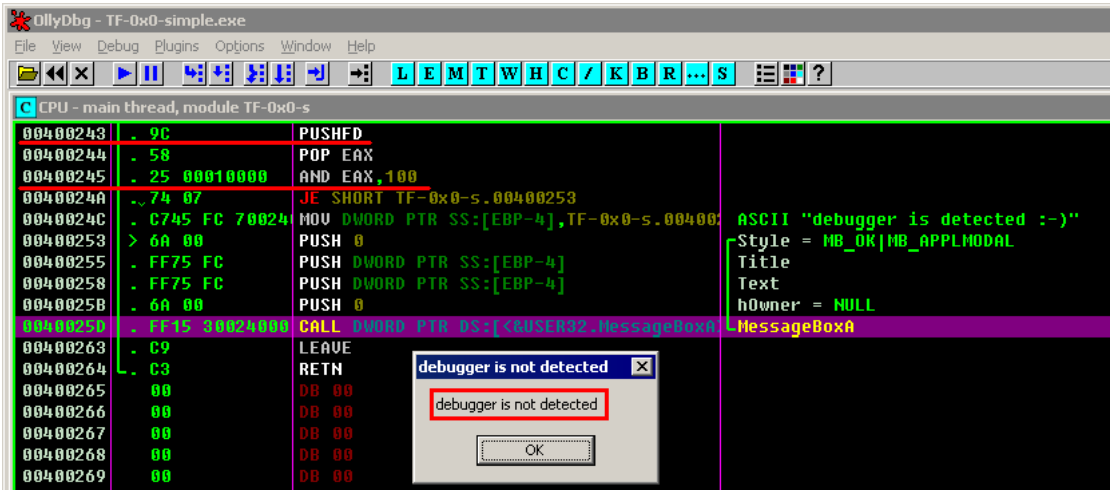

Рисунок 2 Olly с легкостью обходит наш анти-отладочный прием

Теперь испытаем cdb.exe из набора Debugging Tools. Поскольку, он органически не умеет стопиться на OEP, раскомменируем "int 03" и перекомпилируем программу, загрузив ее в отладчик путем указания имени файла в командной строке. Первый раз отладчик всплывает в ntdll!DbgBreakPoint по int 03h. Этот акт всплытия нам совершенно не интересен, так что пишем "g" для продолжения выполнения программы и попадаем на "наш" собственный int 03h, стоящий в начале nezumi(). Последовательно отдавая команду "t", трассируем функцию до достижения CALL'a, а потом говорим "g" и… отладчик не обнаружен!!! Как так?! А очень просто — CDB ыотслеживает команду PUSHFD и _эмулирует_ ее выполнение, "вычищая" TF-бит из стека. Аналогичным образом себя ведет Soft-Ice, Syser, OllyDbg и многие другие "правильные" отладчики. А вот IDA и GDB "честно" показывают TF-бит как он есть, чем и обнаруживают свое присутствие.

эксперимент N #2 – игры с префиксами

В лексиконе x86 помимо самостоятельных команд есть, так называемые, префиксы (prefix), например, префикс повторения (REPE/PEPNE), префикс перекрытия сегмента (CS:, DS:, SS:, ES:, FS:, GS:), префикс изменения разрядности (с опкодом 66h) и т. д. Префиксы работают только со "своим" набором команд, в частности, префикс повторения применяется только совместно со строковыми инструкциями (MOVSD, LODSD, STOSD). На остальные команды он никак не воздействует (лишь увеличивает время их декодирования), а потому PUSHFD и REPE:PUSHFD – синонимы.

Умный отладчик должен учитывать, что перед командой PUSHFD может стоять один или несколько "мусорных" префиксов, автоматически отбрасывая их. Но это в теории. Добавим "REPE" перед "PUSHFD" в нашу программу и перекомпилируем ее, переименовав в TF-0x1-prefix.c.

Такие отладчики как CDB, Soft-Ice и Syser автоматически отбрасывают префиксы, препятствуя их обнаружению. MS VC, IDA и GDB как обнаруживались так и обнаруживаются, а вот OllyDbg (даже в новой версии со всеми плагинами!) палиться даже на банальном REPE, не говоря уже про сочетание нескольких префиксов!

эксперимент N #3 – прерывания в маске

Немного видоизменим нашу тестовую программу, добавив перед инструкцией PUSHFD пару команд MOV AX,SS/MOV SS,AX. И хотя _реальной_ модификации регистра SS при этом не происходит, процессор все равно маскирует трассировочное прерывание на время команды, следующей на MOV SS,AX, которой и является PUSHFD.

```
nezumi()
{
        char *p=noo;                            // презумпция невиновности is on ;-)
        __asm
        {
                int 03                          ; для отладки
                mov     ax,ss                   ; маскируем трассировочное прерывание...
                mov     ss,ax                   ; ...на время выполнения команды PUSHFD
                pushfd                          ; сохраняем флаги в стеке, включая и TF
                pop eax                         ; выталкиваем сохраненные флаги в eax
                and eax,100h                    ; проверяем состояние TF-бита
                jz not_under_dbg                ; если TF не взведен, нас не трассируют
                mov [p],offset yes
        not_under_dbg:
        }
        MessageBox(0, p, p, MB_OK);
}
```

Листинг 3 TF-0x2-SS-change.c — ловля TF-бита через маскирование трассировочного прерывания

Откомпилируем ее и посмотрим, как отладчики справятся с этой ситуацией. Вот мы доходим до MOV SS,AX, нажимаем <F7> (Step into) и… перескакиваем (!) через PUSHFD, позволяя ей сохранить в стеке истинное состояние TF-бита, что немедленно приводит к обнаружению отладчика.

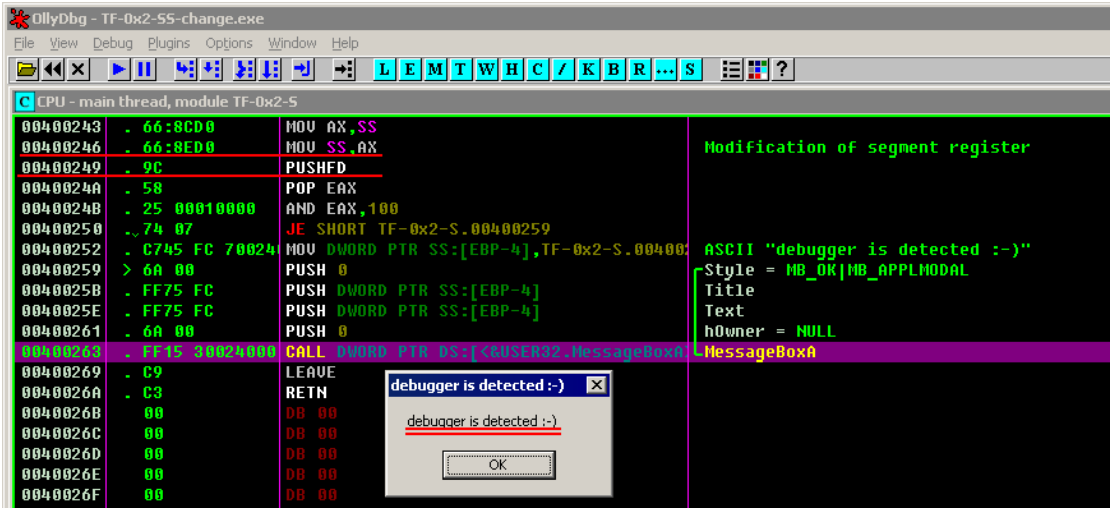

Рисунок 3 Olly "палится" на конструкции MOV SS,AX/PUSHFD

И MS VC, и CDB, и Soft-Ice, и OllyDbg, и IDA, и GDB – все они ловятся на этот крючок. Syser (вплоть до версии 1.95.1900.0894) тоже ловился, пока мыщъх не отписал его разработчикам и они не пофиксли этот баг. В результате чего, Syser стал _единственным_ (на сегодняшний день) отладчиком, распознающим инструкции модифицирующие SS и, если за ними следует PUSHFD, включающий специальный "эмулятор", подсовывающей программе сброшенный TF-бит.

анти-анти-отладка

Пользователям Syser'a хорошо! Им вообще ни о чем заботиться не нужно! А что делать приверженцем остальных отладчиков?! При "ручной" трассировке программы, обнаружив PUSHFD, достаточно прекратить трассировку и, установив точку останова _за_ ее концом, сказать отладчику <Run> или <Go>, прогоняя данный фрагмент кода без трассировки, что (естественно) не позволит обнаружить трассировку, поскольку ее _нет_ вообще.

При автоматизированных прогонах, в OllyDbg можно поставить точки останова на все команды, модифицирующие SS, заставляя его всплывать, передавая бразды правления в наши лапы для разруливания ситуации по вышеописанной методике. Проблема в том, что таких команд очень много, это не только MOV SS,16-bit Reg/Mem и POP SS, но еще MOV,SS/POP SS плюс различные префиксы. В частности, MOV SS, EAX выполняется _точно_ _так_ же как и MOV SS,AX, но имеет другой опкод, что необходимо учитывать при составлении списка команд на которые мы брякаемся.

>>> врезка знаете ли вы… трассировка ветвлений

Pentium-процессоры умеют трассировать… ветвления (условные/безусловные переходы и вызовы функций). Для этого нужно взять MSR-регистр MSR_DEBUGCTLA и взвести в нем бит BTF (single-step on branches), тогда при взведенном TF-бите в регистре флагов EFLAGS трассировочное прерывание будет генерироваться не после каждой машинной команды, а лишь на инструкциях ветвления, что полезно для разбивки программы на функциональные блоки (например, можно написать real-time трассер, сравнивающий прогоны ветвлений программы до си после истечения испытательного срока, что позволит нам легко найти тот "заветный" jxx, который нужно захачить). С другой стороны, если защита взведет BTF-бит, то все известные мыщъх'у отладчики не смогут нормально работать, поскольку не проверяют его состояния при трассировке.

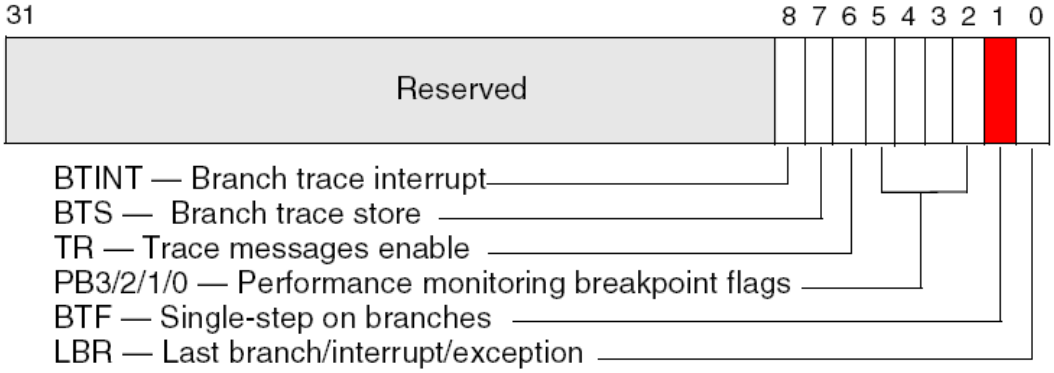

Рисунок 4 бит BTF регистра MSR_DEBUGCTLA

Запись MSR-регистров осуществляется привилегированной командой WRMSR и при попытке ее исполнения на прикладном уровне процессор генерирует исключение, однако, писать свой собственный драйвер для игр с BTF-битом совершенно необязательно и можно воспользоваться недокументированной native-API функцией NtSystemDebugControl(), экспортируемой из NTDLL.DLL пример вызова который можно найти на http://www.openrce.org/blog/view/535/Branch_Tracing_with_Intel_MSR_Registers, однако, для этого необходимо: а) обладать правами администратора; б) в последних пакетах обновления для Server 2003 и XP возможности этой функции были существенно урезаны и, по-видимому, политика урезания продолжится и в дальнейшем, так что все-таки без драйвера не обойтись.

>>> врезка знаете ли вы… что случилось с точками останова?!

Маскирование прерываний после команд, модифицирующих содержимое регистра SS, распространяется так же и на отладочные прерывания, генерируемые в частности аппаратными точками останова по исполнению, установленными на команду, следующую за инструкцией, модифицирующую регистр SS. ***Они, согласно документации от Intel и AMD, не срабатывают и отладчик их мирно пропускает.*** Это не баг в отладчике — это особенность x86-процессоров.

Программные точки останова (представляющие собой опкод CCh) и аппаратные точки останова на чтение/запись данных продолжают работать как ни в чем ни бывало.

>>> врезка знаете ли вы…
как еще можно маскирование прерываний

Существуют два основных способа анализа программ без исходных текстов: статический (дизассемблирование) и динамический (отладка). Дизассемблирование _очень_ плохо справляется с самомодифицирующимся и самогенерируемым кодом. Действительно, защита может затолкать в стек кучу непонятных "циферок", перемещав их самым причудливым образом и передать туда управление. А что у нас там?! Дизассемблер молчит как партизан, хоть пытай его, хоть не пытай! Такой код обычно смотрят под отладчиком.

Представим себе код, расположенный в стеке и помещающий поверх себя несколько машинных команд, первой из которых идет команда модификации регистра SS, затирающая предыдущее содержимое на которое указывает регистр EIP и… благодаря маскированию прерываний "проскакивающая" следующую команду, которая в свою очередь так же может затирать предыдущую. Как следствие — все отладчики, за исключением Syser'a, отобразят лишь _часть_ команд, а остальные будут затерты прежде, чем отладчик получит управление.

Один из примеров реализации такого трюка приведен в программе TF-0x3-crackme.c, которую всем читателям предлагается взломать (благо исходные тексты снабжены подробными комментариями, так что эта задача будет по зубам даже новичкам).

>>> врезка знаете ли вы…
если soft-ice ext отказывается работать, то…

…запускам Редактор Реестра, находим HKLM\SYSTEM\CurrentControlSet\Services\, там мы видим NTice (если только он не был переименован во что-то другое для сокрытия soft-ice от защит) и правим значение параметра KDHeapSize (DWORD) записывая сюда 0x00008000, при необходимости увеличивая и размер стека (KDStackSize, DWORD), увеличивая его на ту же самую величину, после чего перезапускаем soft-ice и, нажав <CTRL-D>, пишем "!PROTECT ON" для сокрытия отладчика от большинства защит.

>>> врезка знаете ли вы что…

…ASPack (и другие упаковщики) использует следующий код, который в действительно равносилен NOP (хотя это и не столь очевидно поначалу):

```
01010002: E803000000   call    .00101000A   -------- (1)
01010007: E9EB045D45   jmp     097C64A25
0101000C: 55           push    ebp
0101000D: C3           retn
0101000E: ...
```

Листинг 4 дизассемблерный фрагмент файла, упакованного ASPack'ом

…а вот лог трассировки:

```
.01010002: E803000000  call    .00101000A
.0101000A: 5D          pop     ebp             ; ebp = 01010007h;
.0101000B: 45          inc     ebp             ; ebp = 01010008h;
.0101000C: 55          push    ebp
.0101000D: C3          retn                    ; goto 01010008h;
.01010008: EB04        jmps    .00101000E
```

Листинг 5 результат трассировки кода упаковщика

Как видно, трассер расставил команды по своим местам и теперь мы без труда можем сказать, что 1010007h:E9EB045D45 скрывает команды: 5Dh 45h POP EBP/INC EBP, EBh 04h — jmps $+6.

Такие вещи легко проходятся в отладчике, но очень тяжело поддаются дизассемблированию.

>>> врезка знаете ли вы что…

…трассировка позволяет ломать программы не меняя в них ни бита кода/данных, ни в оперативной памяти, ни на диске?! В простейшем случае (когда защита состоит из одного лишь jx) мы трассируем до jx, после чего модифицируем значение регистра EIP, всецело принадлежащего _процессору_ а не программе и находящегося все юрисдикции. В более

сложных случаях мы должны воздействовать и на остальные регистры процессора, однако, и эти действия не оговорены во всех лицензионных соглашениях, которые мне только доводилось видеть, а в договорах — что не запрещено, то разрешено!!!

Следовательно, взлом через трассировку с последующим воздействием на регистры, с юридической точки зрения, не является взломом, а потому не может преследоваться по закону, во всяком случае наше законодательство в этом пункте всецело на стороне хакеров. Что же касается штатов, то там _любые_ инструменты, предназначенные для взлома, объявлены вне закона.

>>> врезка сводная таблица с результатами экспериментов

********	ms vc	cdb	soft-ice	soft-ice +IceExt	Syser	OllyDbg	OllyDbg +Phanom	IDA	GDB
PUSHFD	+	-	-	-	-	-	-	+	+
XX: PUSHFD	+	-	-	-	-	+	+	+	+
MOV SS,/PUSHFD	+	+	+	+	-	+	+	+	+

Таблица 1 сводная таблица с результатами экспериментов ("+" — палится, т.е. обнаруживается защитой, "-" — не палится), как мы видим, Syser лидирует среди остальных