Прерывания.
Инструкция **Iret** используется для возврата из прерываний. Не документированной особенностью этой инструкции являются проверки и корректировка сегментных регистров при смене **DPL**. В документации **Intel** на процессор дано краткое описание по обработке **Iret(Intel® 64 And IA-32 Architectures Software Developer's Manual v.2A, p.3-485):**

```
FOR each of segment register (ES, FS, GS, and DS)
  DO;
    IF segment register points to data or non-conforming code segment
    AND CPL > segment descriptor DPL (* stored in hidden part of segment register *)
      THEN (* segment register invalid *)
        SegmentSelector  0; (* null segment selector *)
    FI;
  OD;
END;
```

Частным случаем является обнуление поля **RPL** и теневой части сегментных регистров **Es**, **Fs**, **Gs** и **Ds**, если они содержат нулевой селектор.

ISR прерывания **0x2A** возвращает в **Edx:Eax** текущее число тиков. ISR выглядит следующим образом:

```
KiGetTickCount:
       cmp dword ptr ss:[esp + 4],KGDT_R3_CODE OR RPL_MASK
       jnz short @f
Kgtc00:
       mov eax,dword ptr cs:[KeTickCount]
       mul dword ptr cs:[ExpTickCountMultiplier]
       shrd eax,edx,0x24
       iretd
@@:
```

ISR не изменяет и не использует сегментные регистры кроме **Cs** и **Ss**. По возврату из данного прерывания инструкцией **Iretd** будут сброшены сегментные регистры содержащие нулевой селектор, поэтому в регистре **Ax** будет ноль при исполнении кода:

```
       mov eax,RPL_MASK    ; Requested Privilege Level 3
       mov ds,ax
       int 0x2A
       mov ax,ds
```

Планирование.
Юзермодный поток может быть в любой момент прерван. Обработчики аппаратных прерываний используют инструкцию **Iretd** для передачи управления на прерванный код, также по завершению обработки прерывания может быть вызван планировщик, который переключит процессор на исполнение другого потока если отведённый квант времени текущего потока исчерпан. Более подробно обработка прерываний заканчивается функцией **Kei386EoiHelper**, которая и возвращает управление на прерванный код, восстанавливая состояния процессора на момент прерывания:

```
Kei386EoiHelper:
       [...]
       mov edx,[esp] + TsEdx
       mov ecx,[esp] + TsEcx
       mov eax,[esp] + TsEax
       cmp word ptr [ebp] + TsSegCs, KGDT_R0_CODE
       jz short @f
       lea esp,[ebp] + TsSegGs
       pop gs
       pop es
       pop ds
NonFlatPm_Target:
       lea esp,[ebp] + TsSegFs
       pop fs
@@:
       lea esp,[ebp] + TsEdi
       pop edi
       pop esi
       pop ebx
       pop ebp
       cmp word ptr [esp + 8],0x80
```

```
        ja AbiosExitHelp
        add esp,4
        iretd
```

Нормально поток прерывается около тысячи раз за секунду, тоесть исполняется данный код. Это приводит к сбросу сегментных регистров, если они содержат нулевые селекторы. Время от начало кванта до сброса сегментных регистров не постоянно. Также исключения происходят при обращении к выгруженным в своп страницам. Изза этого код, применяющий описанный механизм должен выполнять дополнительные проверки, дабы исключить такую вероятность.

Регистр **Gs** также сбрасывается програмно функцией **KeContextToKframes()** для юзермодного потока если значение регистра **Cs** равно **0x1B**(**KGDT_R3_CODE | RPL_MASK**). Функции изменяющие контекст потока используют **KeContextToKframes()**. Например не удастся установить значение регистра **Gs** в отличное от нуля(если **Cs = 0x1B)** посредством сервиса **NtSetContextThread** для иного потока, который находится в замороженном состоянии. Следующее чтение контекста вернёт в регистр **Gs** ноль.

Следующий код ожидает прерывание:

```
        mov eax,RPL_MASK
        mov es,ax
@@:
        mov ax,es
        test eax,eax
        jnz @b
```

Проверка что при исполнении кода прерывание не произошло:

```
        mov ebx,RPL_MASK
        xor eax,eax
@@:
        mov gs,bx
        [...]
        mov ax,gs
        test eax,eax
        jz @b
```

Выполнить сброс сегментных регистров можно обратившись к сторожевой странице стека. Это приведет в возникновению исключения, ядро расширит стек и вернёт управление:

```
        mov eax,RPL_MASK
        mov ebp,esp
        mov es,ax
        sub esp,PAGE_SIZE
        and esp,Not(PAGE_SIZE -  1)
        ;Эта инструкция вызывет исключение, стек будет расширен,
        ;после чего инструкция перезапущена и сброшены сегментные
        ;регистры. Поэтому в стеке будет сохранён ноль.
        push es
        pop eax
        mov esp,ebp
```

Хотя время в течении которого поток не будет прерван очень сильно меняется, число прерываний в единицу времени более-менее стабильно.
Факт планирования и измерение числа прерываний является хорошим способом обнаружения трассировки и антиэмуляции.

Апрель 2009, virustech.org