Counter-measures (anti-debugs, ...) cheat sheet

| name | flags | description |
|---|---|---|
| BeingDebugged | `D.....` | $PEB.BeingDebugged$ db `[fs:[30] + 2]` == 1 |
| IsDbgPresent | `D....A` | $BeingDebugged$ check, via $IsDebuggerPresent$ |
| NtGlobalFlag | `D.....` | $PEB.NtGlobalFlag$ dd `[fs:[30] + 68]` has 70[1]set |
| HeapFlags | `D.....` | $Heap.Flags$ dd `[[fs:[30] + 18] + C] == 2` |
| ForceFlags | `D.....` | $Heap.ForceFlags$ dd `[[fs:[30] + 18] + 10]` is not null |
| msvcrt!trigo | `D....A` | $msvcrt!\_CIasin(invalid)$ → `al = NtGlobalFlag ? a8 : 98` |
| deletefiber | `D....A` | $DeleteFiber(invalid)$ → `LastError = ForceFlags ? 80000003 : 57` |
| gs | `.SE...` | `GS` is reset, on thread switch |
| pop ss | `.S....` | debuggers can't step right after `pop SS` → `TF` set in `EFlags`, `100` via `pushf` |
| smsw | `.SE...` | $operand$ = just after FPU `? 80010031 : 8001003b` |
| int 2c/2e | `.SEI..` | slides over next instruction + sets `EDX` to next `EIP`, but incorrect if stepped |
| int 2d | `D...X.` | triggers $BREAKPOINT$ exception if not under a debugger |
| InvalidHandle | `D...XA` | $CloseHandle(invalid)$ → `INVALID HANDLE` exception if debugger is present |
| ChkRemoteDbg | `D....A` | $CheckRemoteDebuggerPresent(GetCurrentProcess()^2, \&result) = 1 : 0$ |
| NtQueryInfo | `D....A` | $NtQueryInformationProcess(-1,\ ProcessDebugPort = 7, var, .....) \rightarrow [var] = present? -1 : 0$ |
| HideThread | `D....A` | $NtSetInformationThread(-2, ThreadhideFromDebugger = 11, -1, 0)$ → not responding |
| csr | `D....A` | $OpenProcess(..., 0, CsrGetProcessId^3())$ → no error if SeDebugPrivilege acquired |
| Timing | `.S....` | comparison of two $RDTSC$, inlined $GetTickCount$[4], `GS` resets, ... |
| Timing API | `.S...A` | comparison of two APIs like $GetTickCount, GetSystemTime, QueryPerformanceCounter$, ... |
| Exceptions tricks (in the exception handler) | | |
| jmp | `....X.` | change resume address via $Context.regEIP$ (Context+`B8`) |
| step | `....X.` | step next instruction and re-trigger via setting $TF$ in $Context.EFlags$ (Context+`C0`) |
| hwbp | `....X.` | set or detects hardware breakpoint via $Context.dr*$ (Context+`04`/+`18`) |
| higher | `....X.` | overwrite higher handler and trigger exception (`[esp+18]`) |
| return | `....X.` | overwrite return address in stack → context re-loading is skipped (`[esp+24]`) |
| Ollydbg (1.1) specific | | |
| esi | `D.....` | esi = -1 on startup under ollydbg, not in general |
| FPU | `D.....` | Display `FFFFFFFF FFFFFFFF C0/40 3D` as float → crash |
| OdbgStr | `D....A` | $OutputDebugStringA(\text{"}\%s\%s\text{"})$ → crash |
| VmWare specific | | |
| backdoor | `D.....` | `in 'VMXh', 'VX'` → exception if not present, else modified `eax` and `ebx` |
| sidt | `D.....` | `[operand + 5] == e8` or `ff` if present |
| sldt | `D.....` | `result != 0` if present |
| str | `D.....` | `result == 4000h` if present |

## Flags

- **anti *D*ebugger** detects if debugger is attached, or if specific tool is used
- **anti *S*tepping** detects the program is not running (ex, not at full speed)
- **anti *E*mulator** detects the program is running under an emulator - via a non documented or complex behavior
- **get e*I*p** provides a way to retrieve current EIP
- **e*X*ception based** relies on exceptions (Vectored, Structured, Unhandled filter)
- ***A*pi-based** relies on an API call

## Reminders

- TF is used by a debugger for stepping: set TF, an exception will be triggered after next execution is stepped
- TEB is at `fs:[18]`
- the PEB is accessible directly (`fs:[30]`) or via $TEB.EnvironmentPointer$ (`[fs:[18] + 30]`)
- $LastError$ is accessible via $TEB.LastErrorValue$ (`[fs:[18] + 34]`) $\rightarrow$ $GetLastError$ is inlinable

## Notes

1. 70 $=FLG\_HEAP\_ENABLE\_TAIL\_CHECK \mid FLG\_HEAP\_ENABLE\_FREE\_CHECK \mid FLG\_HEAP\_VALIDATE\_PARAMETERS$
2. FFFFFFFF (constant value)
3. $dword$[7C980380] (inlineable)
4. $dword$[7FFE0000] $*$ $dword$[7FFE0004]) $>>$ 24 (inlineable)