Anti-Debugging With Exceptions

John Leitch

8/29/2011

john@autosectools.com

http://www.autosectools.com/
Introduction
Several techniques for detecting exception swallowing debuggers have been documented. The concept is simple: by design, debuggers handle certain kinds of exceptions. If such an exception is wrapped in a try block, the exception handle is only executed if a debugger is not attached. Hence it can be inferred that a debugger is attached whenever the exception block is not executed.

One Byte Interrupt 3 (0xCC)
Among the most obvious of exceptions that a debugger would handle is the single byte breakpoint, generated by executing interrupt 3.

```c
BOOL IsDebuggerPresent_Int3() {
    __try
    {
        __asm int 3
    }
    __except(1) {
        return FALSE;
    }
    return TRUE;
}
```

Two Byte Interrupt 3 (0xCD 0x03)
An alternative two byte representation of interrupt 3 can be inserted by using the Visual C++ inline assembler __emit pseudoinstruction. Of all the debuggers tested only OllyDbg consumed the generated exception.

```c
BOOL IsDebuggerPresent_Int3_2Byte() {
    __try
    {
        __asm {
            __emit 0xCD
            __emit 0x03
        }
    }
    __except(1) {
        return FALSE;
    }
    return TRUE;
}
```
**Interrupt 0x2C**
Interrupt 0x2C raises a debug assertion exception. This exception is consumed by WinDbg, but none of the other debuggers tested handled it. The instruction can be generated using the __int2c intrinsic. This interrupt only works on Vista or later. Because of this, the windows version is dynamically checked, and if it is less than six false is returned.

```c
BOOL IsDebuggerPresent_Int2c() {
    OSVERSIONINFO osvi;
    ZeroMemory(&osvi, sizeof(OSVERSIONINFO));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
    GetVersionEx(&osvi);
    if (osvi.dwMajorVersion < 6)
        return FALSE;
    __try
    {
        __int2c();
    }
    __except(1)
    {
        return FALSE;
    }
    return TRUE;
}
```

**Interrupt 0x2D**
When interrupt 0x2D is executed Windows raises a breakpoint exception.

```c
BOOL IsDebuggerPresent_Int2d() {
    __try
    {
        __asm int 0x2d
    }
    __except(1)
    {
        return FALSE;
    }
    return TRUE;
}
```

**ICEBP (0xF1)**
ICEBP is an undocumented instruction that serves as a single byte interrupt 1, generating a single step exception.
BOOL IsDebuggerPresent_IceBp()
{
    __try
    {
        __asm __emit 0xF1
    }
    __except(1)
    {
        return FALSE;
    }
    return TRUE;
}

The Trap Flag
Bit number 8 of the EFLAGS register is the trap flag. When set, a single step exception is generated.

BOOL IsDebuggerPresent_TrapFlag()
{
    __try
    {
        __asm
        {
            pushfd
            or word ptr[esp], 0x100
            popfd
            nop
        }
    }
    __except(1)
    {
        return FALSE;
    }
    return TRUE;
}

RaiseException
Several different types of exceptions generated with the RaiseException function are swallowed by debuggers. OutputDebugString, a function commonly used to detect an attached debugger, relies on RaiseException. The effective exception codes are as follows:

<table>
<thead>
<tr>
<th>Exception Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_BREAKPOINT</td>
<td>0x80000003</td>
</tr>
<tr>
<td>STATUS_SINGLE_STEP</td>
<td>0x80000004</td>
</tr>
<tr>
<td>DBG_PRINTEXCEPTION_C</td>
<td>0x40010006</td>
</tr>
<tr>
<td>DBG_RIPEXCEPTION</td>
<td>0x40010007</td>
</tr>
<tr>
<td>DBG_CONTROL_C</td>
<td>0x40010005</td>
</tr>
<tr>
<td>DBG_CONTROL_BREAK</td>
<td>0x40010008</td>
</tr>
<tr>
<td>DBG_COMMAND_EXCEPTION</td>
<td>0x40010009</td>
</tr>
</tbody>
</table>
BOOL TestExceptionCode(DWORD dwCode)
{
    __try
    {
        RaiseException(dwCode, 0, 0, 0);
    }
    __except(1)
    {
        return FALSE;
    }
    return TRUE;
}

BOOL IsDebuggerPresent_RipException()
{
    return TestExceptionCode(DBG_RIPEXCEPTION);
}
## Compatibility Chart

<table>
<thead>
<tr>
<th>Issue</th>
<th>VS2008 SP1</th>
<th>VS2010 SP1</th>
<th>WinDbg 6.12.1.633</th>
<th>OllyDbg 2.01A4</th>
<th>IdA 5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt 3 (1 Byte)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interrupt 3 (2 Byte)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt 0x2C (Vista and later)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Interrupt 0x2D</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ICEBP (0xF1)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Trap Flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException STATUS_BREAKPOINT</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RaiseException STATUS_SINGLE_STEP</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RaiseException DBG_PRINTEXCEPTION_C</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RaiseException DBG RIPEXCEPTION</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RaiseException DBG CONTROL_C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException DBG CONTROL_BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException DBG COMMAND_EXCEPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException ASSERTION_FAILURE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException STATUS GUARD_PAGE_VIOLATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RaiseException SEGMENT_NOTIFICATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RaiseException EXCEPTION WX86_SINGLE_STEP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RaiseException EXCEPTION WX86_BREAKPOINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

## Resources

- Anti-unpacker tricks - part two  
- Anti-unpacker tricks - part three  
- Windows Anti-Debug Reference  
• Undocumented OpCodes: ICEBP
  http://www.rcollins.org/secrets/opcodes/ICEBP.html
• EFLAGS Individual Bit Flags
  http://www.c-jump.com/CIS77/ASM/Instructions/I77_0070_eflags_bits.htm
• __int2c
• Reversing: Secrets of Reverse Engineering
• The _emit Pseudoinstruction
  http://msdn.microsoft.com/en-us/library/1b80826t(v=vs.80).aspx
• NTSTATUS values