Anti-Debug: Timing


anti-debug.checkpoint.com/techniques/timing.html

Contents

Timing

When a process is traced in a debugger, there is a huge delay between instructions and execution. The “native” delay between some parts of code can be measured and compared with the actual delay using several approaches.

1. RDPMC/RDTSC

These instructions require the flag `PCE` to be set in `CR4` register.

`RDPMC` instruction can be used only in Kernel Mode.

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    ULARGE_INTEGER Start, End;
    __asm
    {
        xor  ecx, ecx
        rdpmc
        mov  Start.LowPart, eax
        mov  Start.HighPart, edx
    }
    // ... some work
    __asm
    {
        xor  ecx, ecx
        rdpmc
        mov  End.LowPart, eax
        mov  End.HighPart, edx
    }
    return (End.QuadPart - Start.QuadPart) > qwNativeElapsed;
}
```

`RDTSC` is a User Mode instruction.

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    ULARGE_INTEGER Start, End;
    __asm
    {
        xor  ecx, ecx
        rdtsc
        mov  Start.LowPart, eax
        mov  Start.HighPart, edx
    }
    // ... some work
    __asm
    {
        xor  ecx, ecx
        rdtsc
        mov  End.LowPart, eax
        mov  End.HighPart, edx
    }
    return (End.QuadPart - Start.QuadPart) > qwNativeElapsed;
}
```

2. GetLocalTime()

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    SYSTEMTIME stStart, stEnd;
    FILETIME ftStart, ftEnd;
    ULARGE_INTEGER uiStart, uiEnd;

    GetLocalTime(&stStart);
    // ... some work
    GetLocalTime(&stEnd);

    if (!SystemTimeToFileTime(&stStart, &ftStart))
        return false;
    if (!SystemTimeToFileTime(&stEnd, &ftEnd))
        return false;

    uiStart.LowPart  = ftStart.dwLowDateTime;
    uiStart.HighPart = ftStart.dwHighDateTime;
    uiEnd.LowPart  = ftEnd.dwLowDateTime;
    uiEnd.HighPart = ftEnd.dwHighDateTime;
    return (uiEnd.QuadPart - uiStart.QuadPart) > qwNativeElapsed;
}
```

3. GetSystemTime()

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    SYSTEMTIME stStart, stEnd;
    FILETIME ftStart, ftEnd;
    ULARGE_INTEGER uiStart, uiEnd;

    GetSystemTime(&stStart);
    // ... some work
    GetSystemTime(&stEnd);

    if (!SystemTimeToFileTime(&stStart, &ftStart))
        return false;
    if (!SystemTimeToFileTime(&stEnd, &ftEnd))
        return false;

    uiStart.LowPart  = ftStart.dwLowDateTime;
    uiStart.HighPart = ftStart.dwHighDateTime;
    uiEnd.LowPart  = ftEnd.dwLowDateTime;
    uiEnd.HighPart = ftEnd.dwHighDateTime;
    return (uiEnd.QuadPart - uiStart.QuadPart) > qwNativeElapsed;
}
```

4. GetTickCount()

C/C++ Code

```
bool IsDebugged(DWORD dwNativeElapsed)
{
    DWORD dwStart = GetTickCount();
    // ... some work
    return (GetTickCount() - dwStart) > dwNativeElapsed;
}
```

5. ZwGetTickCount() / KiGetTickCount()

Both functions are used only from Kernel Mode.

Just like User Mode `GetTickCount()` or `GetSystemTime()`, Kernel Mode `ZwGetTickCount()` reads from the `KUSER_SHARED_DATA` page. This page is mapped read-only into the user mode range of the virtual address and read-write in the kernel range. The system clock tick updates the system time, which is stored directly in this page.

`ZwGetTickCount()` is used the same way as `GetTickCount()`. Using `KiGetTickCount()` is faster than calling `ZwGetTickCount()`, but slightly slower than reading from the `KUSER_SHARED_DATA` page directly.

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    ULARGE_INTEGER Start, End;
    __asm
    {
        int  2ah
        mov  Start.LowPart, eax
        mov  Start.HighPart, edx
    }
    // ... some work
    __asm
    {
        int  2ah
        mov  End.LowPart, eax
        mov  End.HighPart, edx
    }
    return (End.QuadPart - Start.QuadPart) > qwNativeElapsed;
}
```

6. QueryPerformanceCounter()

C/C++ Code

```
bool IsDebugged(DWORD64 qwNativeElapsed)
{
    LARGE_INTEGER liStart, liEnd;
    QueryPerformanceCounter(&liStart);
    // ... some work
    QueryPerformanceCounter(&liEnd);
    return (liEnd.QuadPart - liStart.QuadPart) > qwNativeElapsed;
}
```

7. timeGetTime()

C/C++ Code

```
bool IsDebugged(DWORD dwNativeElapsed)
{
    DWORD dwStart = timeGetTime();
    // ... some work
    return (timeGetTime() - dwStart) > dwNativeElapsed;
}
```

Mitigations

- During debugging: Just fill timing checks with `NOP`s and set the result of these checks to the appropriate value.
- For anti-anti-debug solution development: There is no great need to do anything with it, as all timing checks are not very reliable. You can still hook timing functions and accelerate the time between calls.