ورقات بيضاء في المختبر

WHITE PAPERS in ViTRO

(05/2020) #02

سلسلة

نظرة و تجربة

موضوع اليوم

TLS Callback Functions

بداية، سوف اتطرق الى ما اثار اهتمامي و قمت بتجربته في آلية TLS Callback و هو باراميتر Reason الموجود في الدالة، حيث انه تبين لي ان هذا الباراميتر يعطيني مرونة كبيرة في تحديد متى يتم تنفيذ الأوامر الموجودة في الدالة.

حسب تعريف الدالة في الأسفل، الباراميتر Reason يتأثر بأربعة قيم ثابتة.

TLS Callback Functions

The program can provide one or more TLS callback functions to support additional initialization and termination for TLS data objects. A typical use for such a callback function would be to call constructors and destructors for objects.

Although there is typically no more than one callback function, a callback is implemented as an array to make it possible to add additional callback functions if desired. If there is more than one callback function, each function is called in the order in which its address appears in the array. A null pointer terminates the array. It is perfectly valid to have an empty list (no callback supported), in which case the callback array has exactly one member-a null pointer.

The prototype for a callback function (pointed to by a pointer of type PIMAGE_TLS_CALLBACK) has the same parameters as a DLL entry-point function:

C++ = Copy

```
typedef VOID
(NTAPI *PIMAGE_TLS_CALLBACK) (
    PVOID DllHandle,
    DWORD Reason,
    PVOID Reserved
    );
```

The Reserved parameter should be set to zero. The Reason parameter can take the following values:

Setting	Value	Description
DLL_PROCESS_ATTACH	1	A new process has started, including the first thread.
DLL_THREAD_ATTACH	2	A new thread has been created. This notification sent for all but the first thread.
DLL_THREAD_DETACH	3	A thread is about to be terminated. This notification sent for all but the first thread.
DLL_PROCESS_DETACH	0	A process is about to terminate, including the original thread.

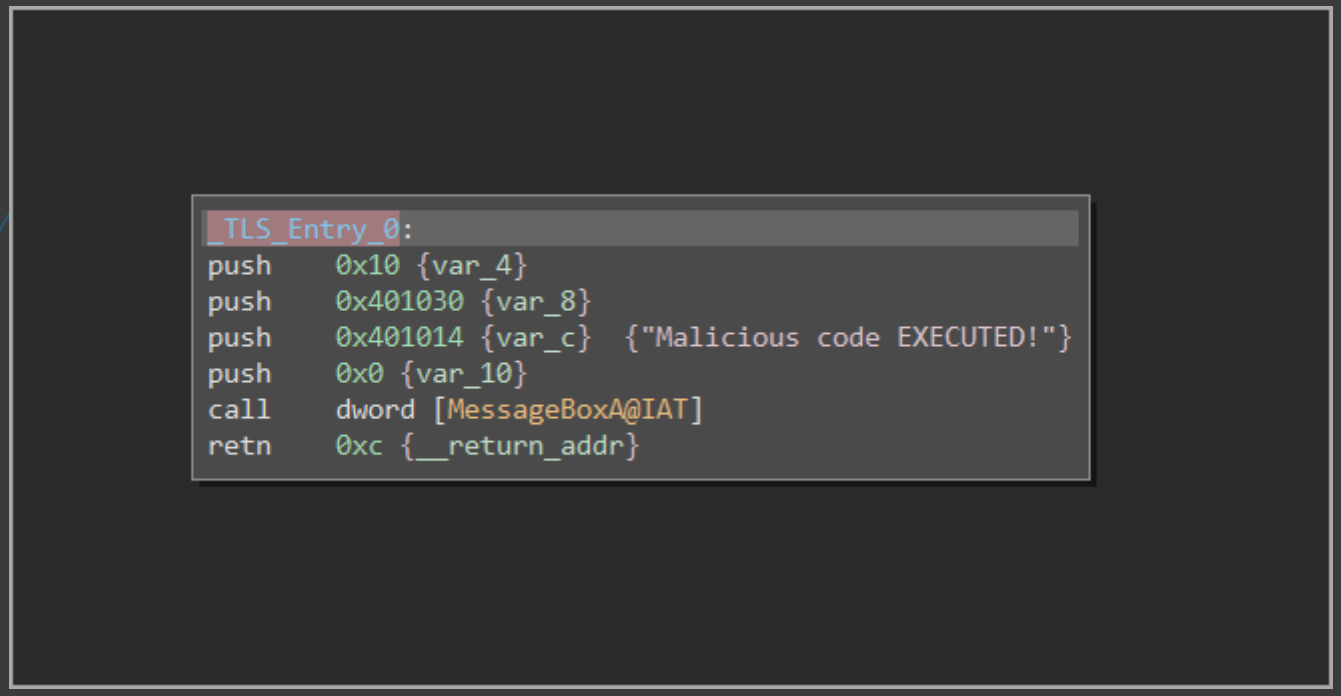
طبعا، من خلال تعريف الدالة فوق، نلاحظ ان طريقة وضع بارامترات الدالة و القيم الثابتة التي تتأثر بها تذكرنا بالدالة الرئيسية لملفات الـ DLL.

في الحقيقة دالة TLS CALLBACK تملك نفس سلوك الدالة الرئيسية لملفات الـ DLL.

بداية التجربة:

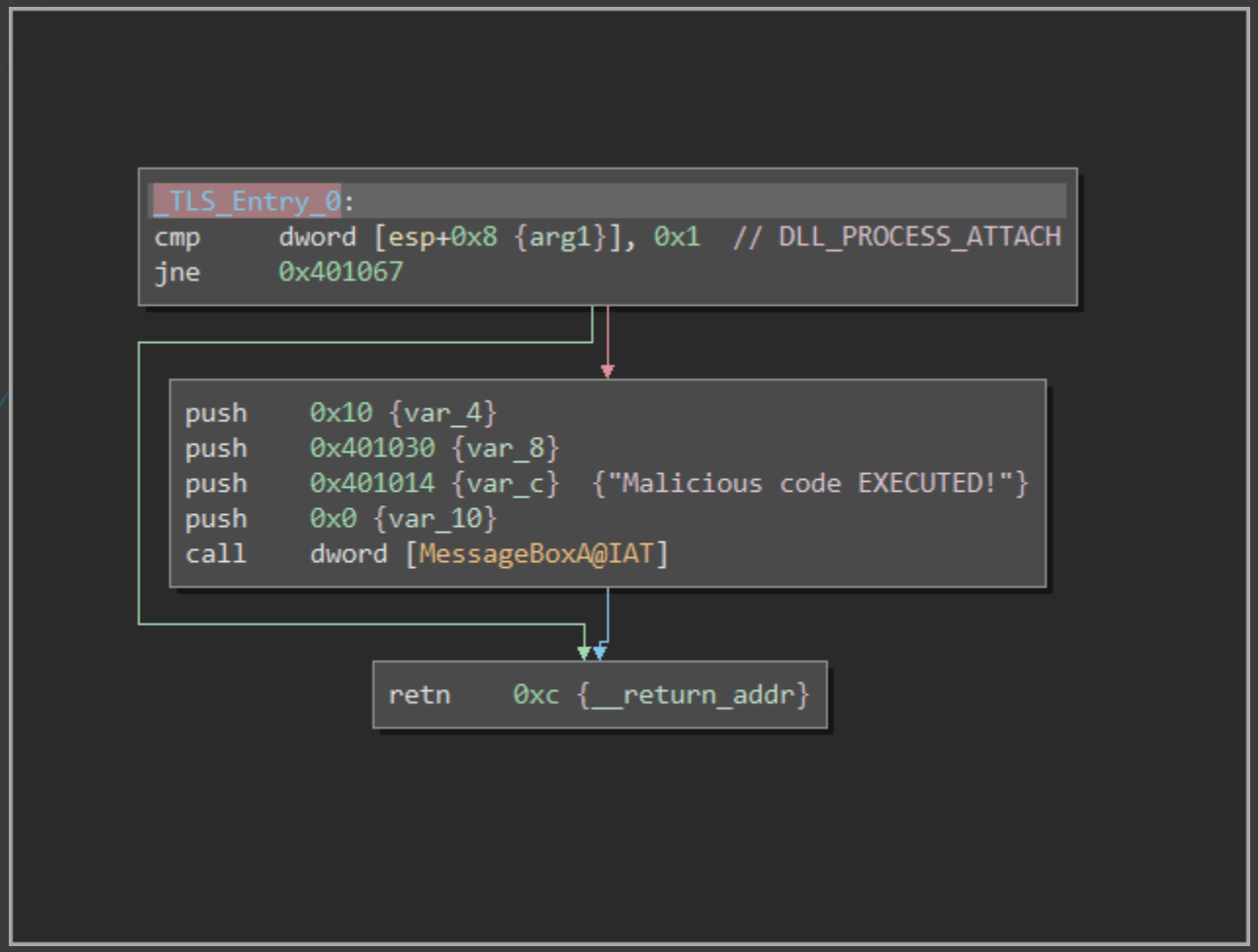
الحالة رقم 01:

نقوم بتحميل الملف case01.exe في المنقح، ما نلاحظه هو عدم وجود أي شروط لتنفيذ الأوامر مما سوف يؤدي الى تنفيذها بتجاهل تحقق شرط (بارامترات Reason) بعدد مرات غير متحكم في حدث (Event) تنفيذها.



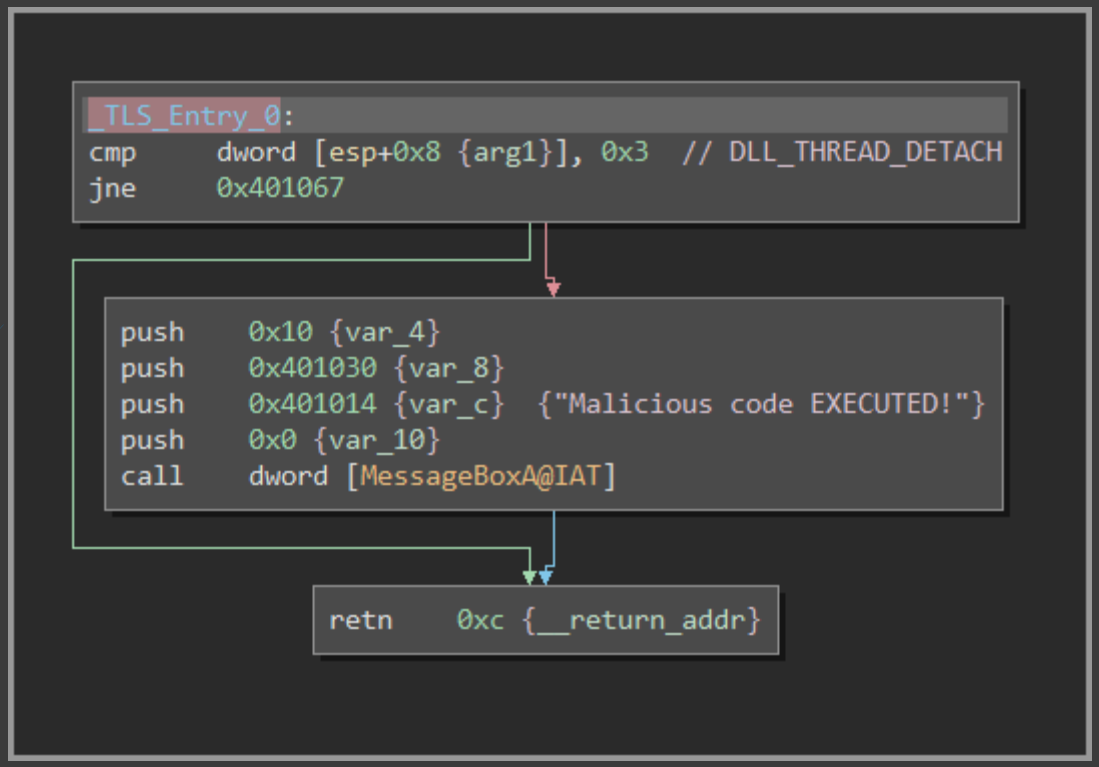
الحالة رقم 02:

نقوم بتحميل الملف case02.exe في المنقح، ما نلاحظه هو وجود شرط لتنفيذ الأوامر **فقط بتحقق شرط** (بارامترات Reason) اذا كان يساوي قيمة 1 و هي القيمة الثابتة تحت تسمية DLL_PROCESS_ATTACH



الحالة رقم 03:

نقوم بتحميل الملف case03.exe **في المنقح، ما نلاحظه هو وجود شرط لتنفيذ الأوامر فقط بتحقق شرط (بارامترات** Reason**)**
اذا كان يساوي قيمة 3 **و هي القيمة الثابتة تحت تسمية** DLL_THREAD_DETACH



الحالة رقم 04:

نقوم بتحميل الملف case04.dll **في المنقح** (Ollydbg)**، بعد تعديل خيار تجاهل توقف المنقح في نقطة الدخول و جعله يتوقف مبكرا** (system breakpoint) **لتفادي تنفيذ أي أوامر مبكرة كوسيلة معروفة مضادة لـ** TLS CALLBACK.

ما نلاحظه هو ان المنقح OllyDbg **لكي يستطيع تنقيح المكتبات يعتمد على ملفات تنفيذية مساعدة** loaddll.exe **يحمل من خلالها المكتبات، بمعنى انه ينقح الملفات المساعدة لكي يصل الى أوامر المكتبات.**

و في حالة مكتبة (case04.dll) **تحتوي على** TLS CALLBACK Function **فإن محتواها يتم تنفيذه حتى وان ان كان خيار المنقح** System breakpoint **مفعل.**

و في الأخير مثال كامل:

```
#include <windows.h>

#pragma comment(linker, "/INCLUDE:__tls_used")
#pragma comment(linker, "/INCLUDE:_tls_entry")
#pragma data_seg(".CRT$XLB" )

VOID NTAPI TLSCallback(PVOID DllHandle, DWORD Reason, PVOID Reserved);

extern "C" {
   PIMAGE_TLS_CALLBACK tls_entry = TLSCallback;
}

VOID NTAPI TLSCallback(PVOID DllHandle, DWORD Reason, PVOID Reserved) {

      switch(Reason)  {
       case DLL_PROCESS_ATTACH:
        // Initialize once for each new process.
         MessageBoxA(NULL,
                        "Malicious code EXECUTED! (DLL_PROCESS_ATTACH)",
                        "PoC",
                        MB_ICONERROR);
          break;

       case DLL_THREAD_ATTACH:
        // Do thread-specific initialization.
         MessageBoxA(NULL,
                        "Malicious code EXECUTED! (DLL_THREAD_ATTACH)",
                        "PoC",
                        MB_ICONERROR);
          break;
```

```
        case DLL_THREAD_DETACH:
        // Do thread-specific cleanup.
         MessageBoxA(NULL,
                     "Malicious code EXECUTED! (DLL_THREAD_DETACH)",
                     "PoC",
                     MB_ICONERROR);
          break;

       case DLL_PROCESS_DETACH:
        // Perform any necessary cleanup.
         MessageBoxA(NULL,
                     "Malicious code EXECUTED! (DLL_PROCESS_DETACH)",
                     "PoC",
                     MB_ICONERROR);
          break;
    }
}

int main() {

    return 0;
}
```