2021/03/02_PE_第11课_线程局部存储

笔记本： PE
创建时间： 2021/3/2 星期二 16:27
作者： ileemi

- 多线程编程
- 线程局部存储
  - 动态使用
  - 静态使用
- Tls 结构体
- 定位TLS

软件断点在CPU的设计中是以线程为单位的，但是在实际使用中和线程没关系。

多线程编程

> 所线程编程，存在 "资源竞争" 的问题。

解决资源竞争的办法：

- 加锁：会降低程序的执行效率
- 使用局部存储（每个线程分配对应的资源，最后将资源合并）

> 根据线程的数量进行具体的判断是使用加锁还是局部存储。

线程局部存储

> 线程局部存储（Thread local storage）：各线程独立，属于线程自己的全局变量（可在线程内部跨函数）。

> PE文件中会记录Tls的相关信息。**可执行文件**可以**静态使用Tls**，而 **".dll" 文件**中不推荐静态使用Tls，推荐**动态使用Tls**（dll加载在主模块之后，Tls的存储空间可能会被主模块使用完，可以判断是否申请成功）。

> Tls 在数据目录的第9项，同时在PE文件中会多出一个对应的节 ".tls"（存储Tls的结构信息）。

不同的线程访问同一个变量时，访问的地址不一样，代码一样。使用之前需要为每个线程申请一段存储空间。

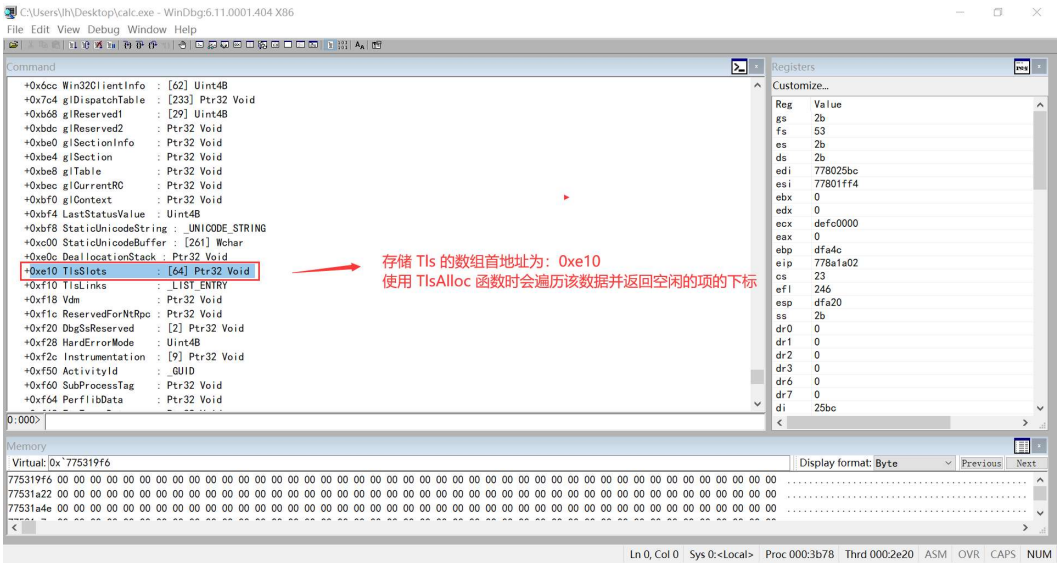
操作系统提供对应的API：

- TlsAlloc：检索可用的数组单元并返回其索引
- TlsFree：释放此单元

- TlsSetValue：设置值
- TlsGetValue：获取值

使用 TlsAlloc 函数最多申请4个字节的空间，但是可以保存指针（指向结构体）。

Tls 数组是有限的，每一个线程都有一个对应的TEB（利用TEB对变量进行存储），互不干扰。



动态使用

代码示例：

```
#include <windows.h>
#include <stdio.h>

#define THREAD_NUM  50
#define COUNT_MAX  40000

//10000

//线程局部存储(Thread local storage) 动态使用
int g_number = 0;

DWORD WINAPI WorkThread(LPVOID lpParameter) {
  DWORD dwIndex =  TlsAlloc(); //TEB  fs:[1480h] 32  00401000
00402000
  TlsSetValue(dwIndex, 0);
  LPVOID dwValue = NULL;

  for (int i = 0; i < COUNT_MAX; i++) {
    dwValue = TlsGetValue(dwIndex);
    *(DWORD*)&dwValue += 1;
    TlsSetValue(dwIndex, dwValue);
    //printf("%s TID:%d dwValue:%d\n", __FUNCTION__,
```

```
    GetCurrentThreadId(), dwValue);
  }
  TlsFree(dwIndex);

  return (DWORD)dwValue;
}

int main()
{
  HANDLE hThread[THREAD_NUM];
  //TlsAlloc  TlsFree  TlsSetValue  TlsGetValue
  for (int i = 0; i < THREAD_NUM; i++) {
    hThread[i] = CreateThread(NULL, 0, WorkThread, NULL, 0, NULL);
  }

  WaitForMultipleObjects(THREAD_NUM, hThread, TRUE, INFINITE);
  for (int i = 0; i < THREAD_NUM; i++) {
    DWORD dwExitCode = 0;
    GetExitCodeThread(hThread[i], &dwExitCode);
    g_number += dwExitCode;
  }

  printf("g_number:%d\n", g_number);
  system("pause");
  return 0;
}
```

静态使用

> 不在调用API，在目标变量前添加 "__declspec(thread)"， 申请内存、获取下标、设置值都由编译器去完成。

静态使用Tls，定义的Tls变量，编译器申请Tls空间的时机在OEP之前。

示例代码1：

```
#include <windows.h>
#include <stdio.h>

#define THREAD_NUM  50
#define COUNT_MAX  40000

//线程局部存储(Thread local storage) 静态使用
__declspec(thread) int g_number = 0;

DWORD WINAPI WorkThread(LPVOID lpParameter) {
```

```
  for (int i = 0; i < COUNT_MAX; i++) {
    g_number++;
  }
  return g_number;
}

int main() {
  HANDLE hThread[THREAD_NUM];
  //TlsAlloc  TlsFree  TlsSetValue  TlsGetValue
  for (int i = 0; i < THREAD_NUM; i++) {
    hThread[i] = CreateThread(NULL, 0, WorkThread, NULL, 0, NULL);
  }

  WaitForMultipleObjects(THREAD_NUM, hThread, TRUE, INFINITE);
  for (int i = 0; i < THREAD_NUM; i++) {
    DWORD dwExitCode = 0;
    GetExitCodeThread(hThread[i], &dwExitCode);
    g_number += dwExitCode;
  }

  printf("g_number:%d\n", g_number);
  system("pause");
  return 0;
}
```

Tls 结构体

> **IMAGE_TLS_DIRECTORY32**

```
#define IMAGE_DIRECTORY_ENTRY_TLS 9

typedef struct _IMAGE_TLS_DIRECTORY32 {
  // TLS初始化数据的起始地址
  DWORD StartAddressOfRawData; // 重要
  // TLS初始化数据的结束地址，两个正好定位一个范围，范围中存放初始的值
  DWORD EndAddressOfRawData; // 重要
  // TLS索引的位置，指向一个指针数组 PDWOD
  DWORD AddressOfIndex; // 重要
  // TLS初始化回调函数的数组指针，以0结尾，回调跟 dllmain一样，TLS函数
执行在主线程前
  DWORD AddressOfCallBacks; // 重要
   // 填充0的个数
  DWORD SizeOfZeroFill;
  union {
```

```
    DWORD Characteristics; // 保留
    struct {
      DWORD Reserved0 : 20;
      DWORD Alignment : 4;
      DWORD Reserved1 : 8;
    } DUMMYSTRUCTNAME;
  } DUMMYUNIONNAME;
} IMAGE_TLS_DIRECTORY32;

// Tls 函数的回调表，回调函数会在OEP前执行代码
typedef VOID (NTAPI* PIMAGE_TLS_CALLBACK) (
  PVOID DllHandle,
  DWORD Reason,
  PVOID Reserved
);
```

> 存储类微软官方文档
>
> 线程本地存储 (TLS)文档说明

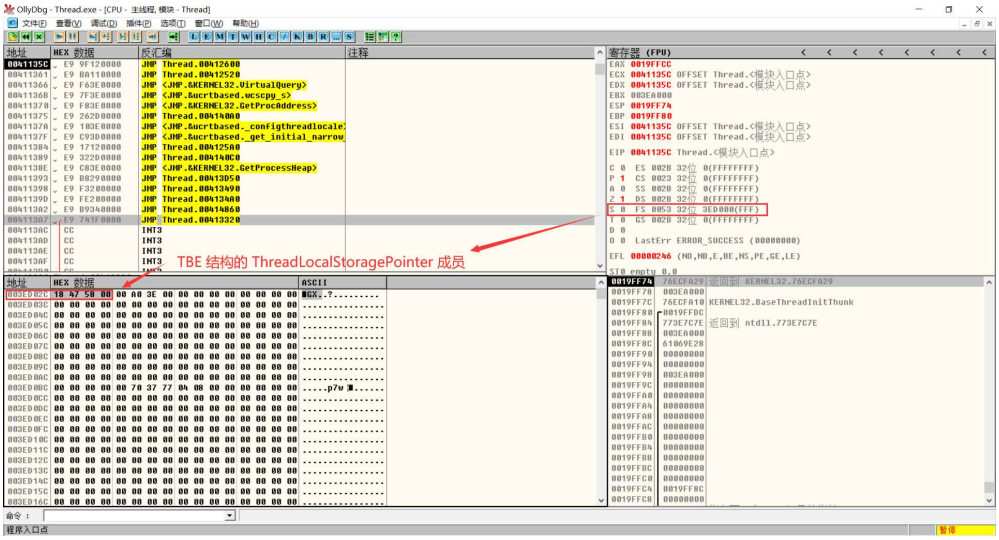
注意：

- 静态使用TLS才和PE有关系，动态使用TLS和PE没有关系。
- 在 Windows Vista 之前的 Windows 操作系统上， __declspec(thread) 有一些限制。 如果 DLL 将任何数据或对象声明为 __declspec(thread)，则如果动态加载，则可能导致保护错误。 在用 LoadLibrary加载 DLL 后，只要代码引用数据，就会导致系统故障 __declspec(thread)。 由于线程的全局变量空间在运行时进行分配，此空间的大小基于对应用程序的要求加静态链接的所有 DLL 的要求的计算。 使用时 LoadLibrary，不能扩展此空间以允许使用声明的线程本地变量 __declspec(thread)。 如果 DLL 可能使用加载，请在 DLL 中使用 TLS Api （如 TlsAlloc）分配 tls LoadLibrary 。

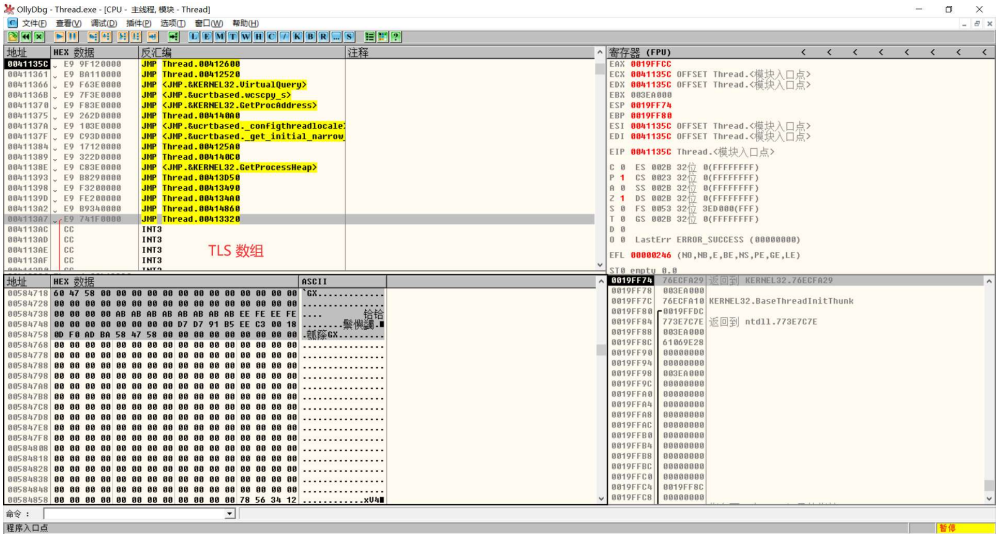
TLS 多用于加壳和反调试、代码注入（反调试、解密）。TLS回调函数在系统断点之后（Dll需要提前加载好，回调函数中有可能需要使用API），在OEP之前执行的。

定位TLS

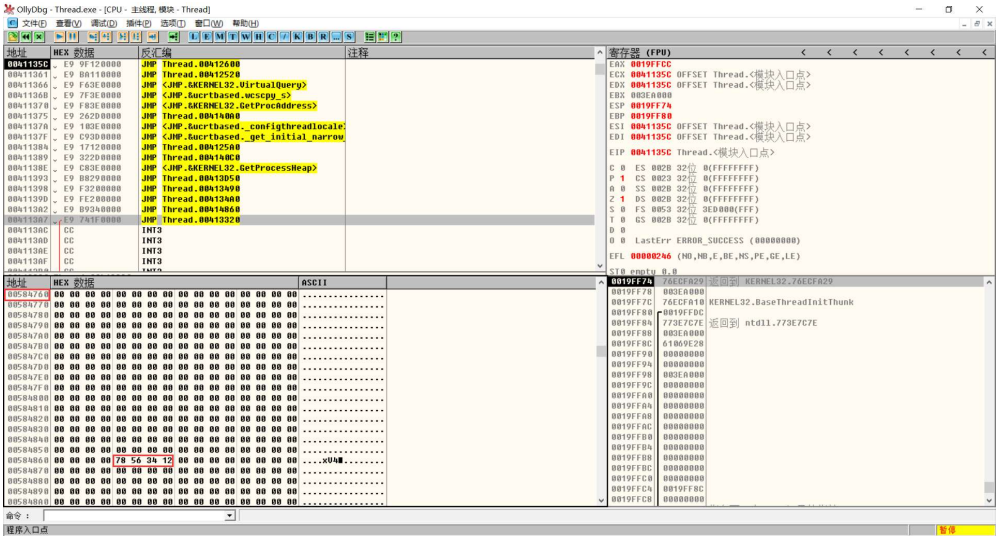
- 定位TEB的 ThreadLocalStoragePointer 成员：



- 定位TLS数组：



- 访问TLS数组第0项，根据偏移访问指定的变量：



```
10  //10000
11
12  //线程局部存储(Thread local storage) 静态使用
13
14  int fun1() {
15      printf("g_number fun1:%d\n", GetCurrentThreadId());
16      //MessageBoxA(NULL, "Init1", "51asm", MB_OK);
17      return 10;
18  }
19
20  __declspec(thread) int g_number = 0x12345678;
21
22  void NTAPI Init1(PVOID DllHandle, DWORD Reason, PVOID Reserved) {
23      printf("g_number Init1:%d\n", GetCurrentThreadId());
24      //MessageBoxA(NULL, "Init1", "51asm", MB_OK);
25      if (IsDebuggerPresent()) {
26          ExitProcess(0);
27      }
28      g_number = 10;
29  }
```