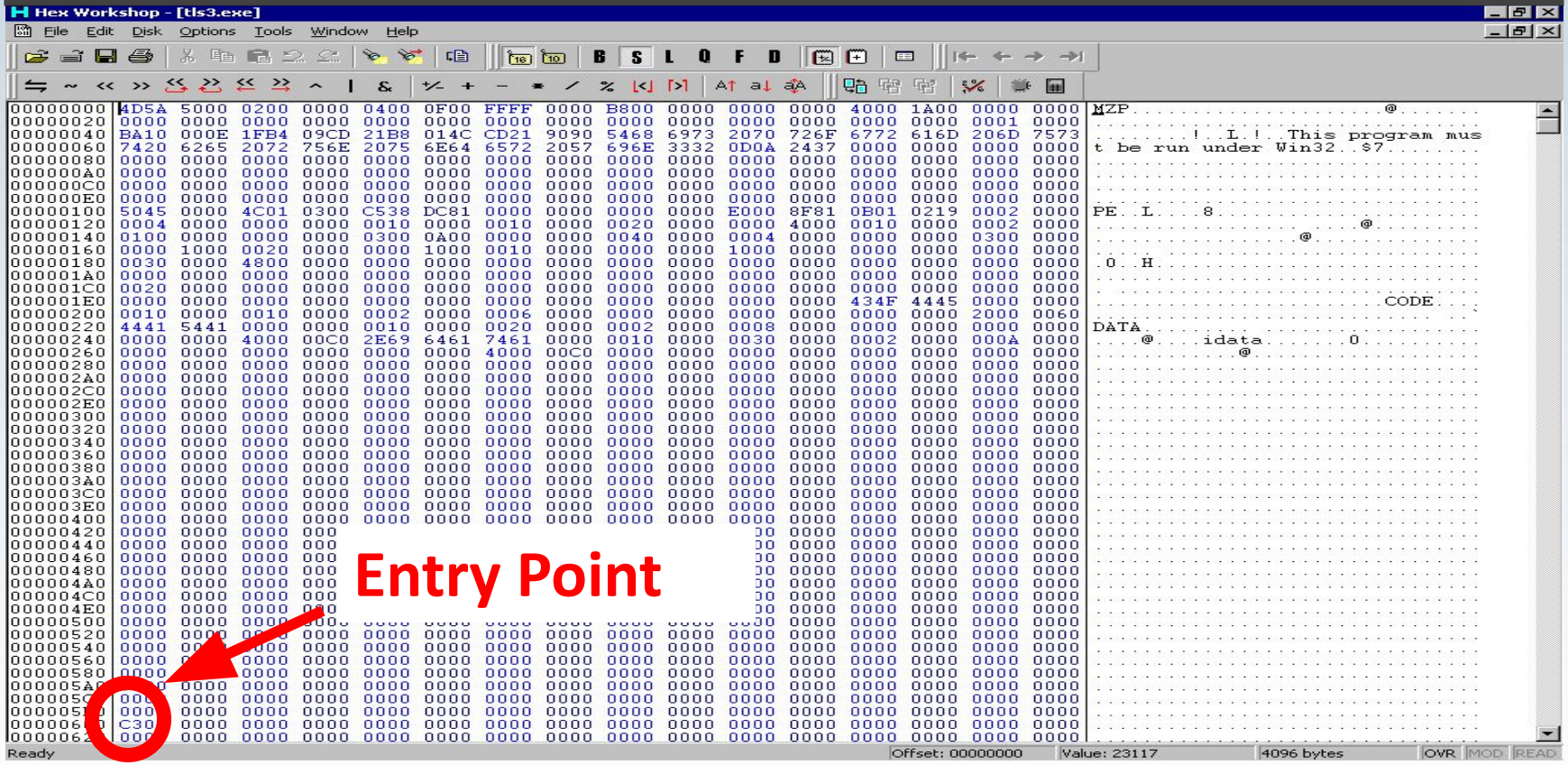
Fun With Thread Local Sto

Peter Ferrie
Senior Anti-virus Researcher
2 July, 2008

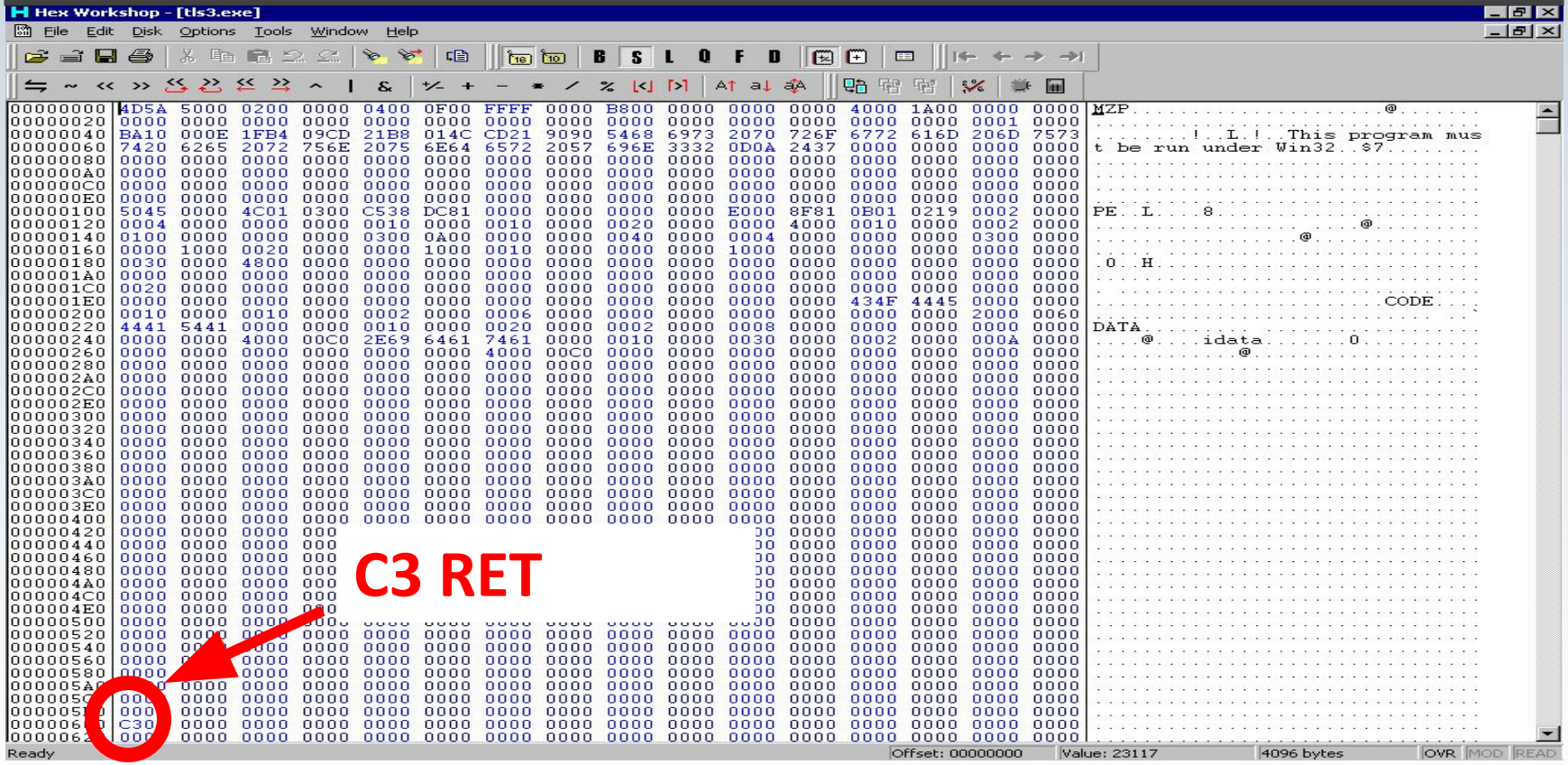
You Can Call Me Al

Thread Local Storage callbacks were discovered in 2000.
However, widespread use didn't occur until 2004.
Now, it should be the first place to look for code,
since it runs before the main entrypoint.
And that can make all the difference…

Empty!


Hex Workshop - [tls3.exe]
File Edit Disk Options Tools Window Help
Entry Point
Ready
Offset: 00000000
Value: 23117
4096 bytes
OVR MOD READ

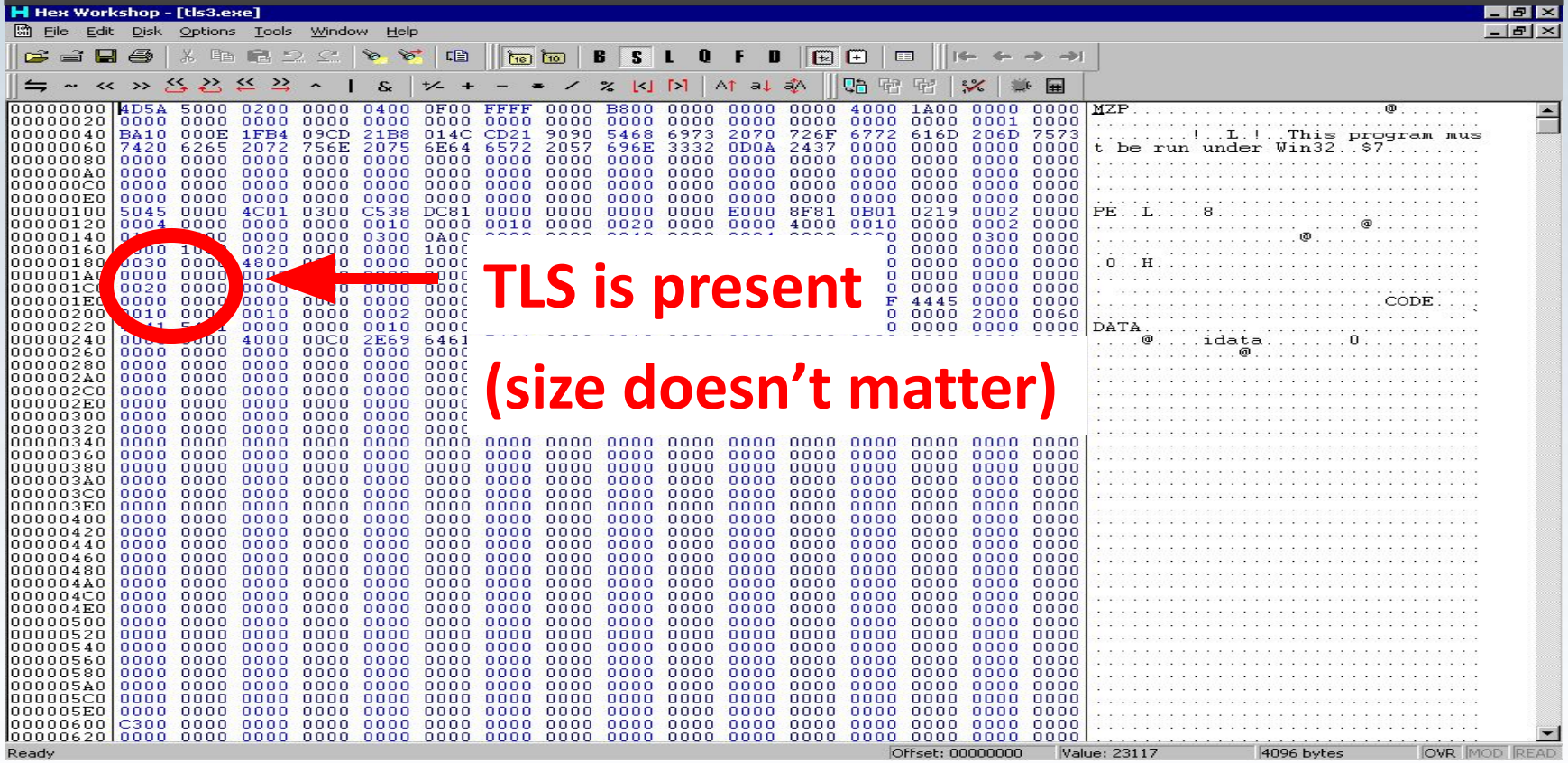
Empty!


Hex Workshop - [tls3.exe]
File Edit Disk Options Tools Window Help
C3 RET
Ready
Offset: 00000000
Value: 23117
4096 bytes
OVR MOD READ

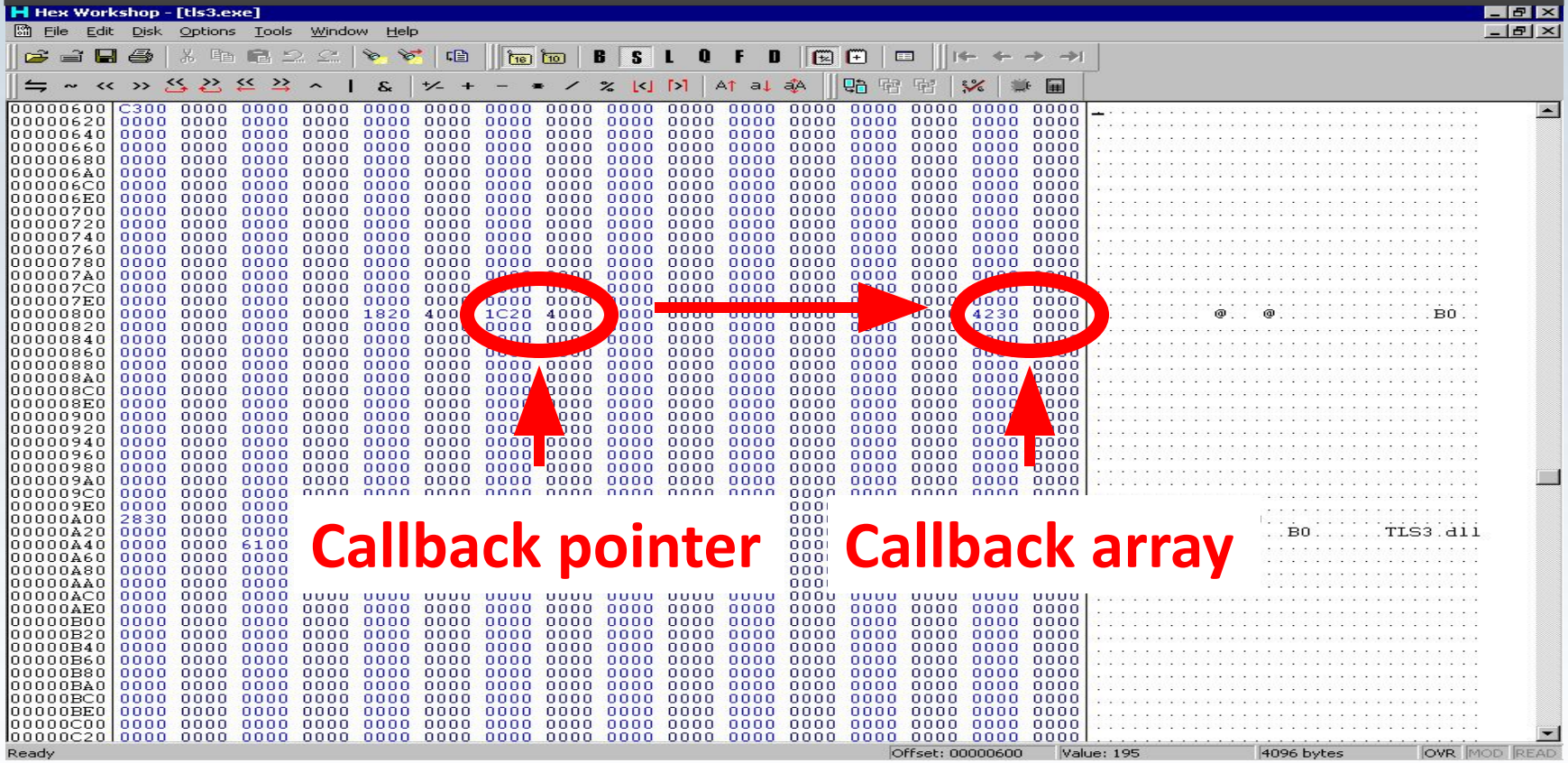
Empty!

So the main file does nothing.
If we assume that the structure is normal,
then we could check the thread local storage table.
Just in case.

Empty!


Hex Workshop - [tls3.exe]
File Edit Disk Options Tools Window Help
TLS is present
(size doesn't matter)
Ready
Offset: 00000000
Value: 23117
4096 bytes

Empty!


Hex Workshop - [tls3.exe]
File Edit Disk Options Tools Window Help
Callback pointer
Callback array
TLS3 dll
Ready
Offset: 00000600
Value: 195
4096 bytes

Empty!

So the search moves to the callbacks,
of which there is only one, but it looks peculiar.
It's not a virtual address.

The One and Only

IDA - tls3.exe

File Edit Jump Search View Debug Options Window

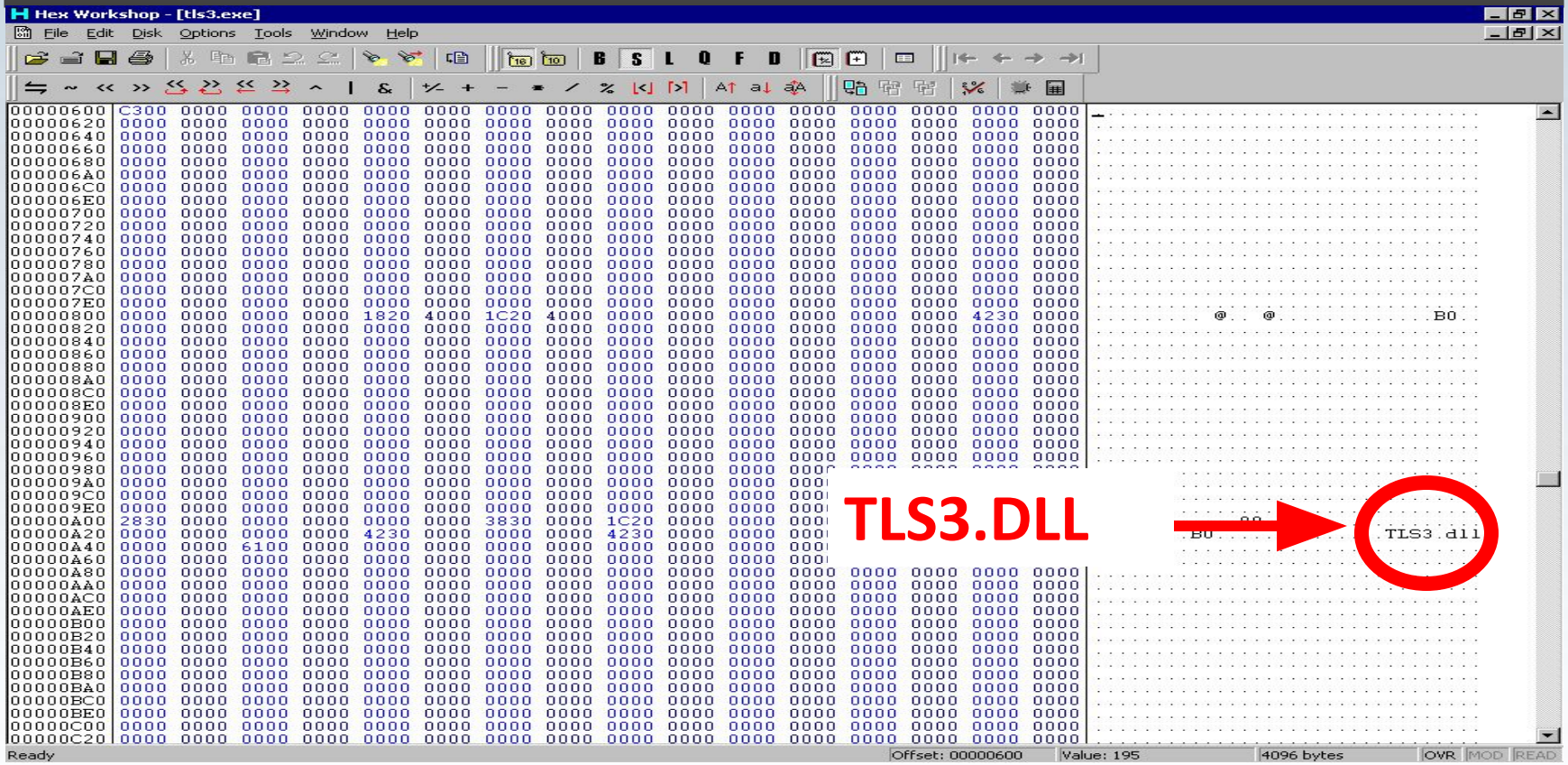
IDA View-A

```
DATA:00402000 ; Section 2. (virtual address 00002000)
DATA:00402000 ; Virtual size                   : 00001000 (   4096.)
DATA:00402000 ; Section size in file           : 00000200 (    512.)
DATA:00402000 ; Offset to raw data for section: 00000800
DATA:00402000 ; Flags C0000040: Data Readable Writable
DATA:00402000 ; Alignment     : default
DATA:00402000 ; ===========================================================================
DATA:00402000
DATA:00402000 ; Segment type: Pure data
DATA:00402000 ; Segment permissions: Read/Write
DATA:00402000 DATA            segment para public 'DATA' use32
DATA:00402000                 assume cs:DATA
DATA:00402000                 ;org 402000h
DATA:00402000 TlsDirectory    TLS_DIR_ENTRY <0, 0, offset TlsIndex, offset TlsCallbacks, 0, 0>
DATA:00402000                                         ; DATA XREF: HEADER:pe_header↑o HEADER:00400220↑o
DATA:00402018 TlsIndex        dd 0                    ; DATA XREF: DATA:TlsDirectory↑o
DATA:0040201C ;
DATA:0040201C ; Imports from TLS3.dll
DATA:0040201C ;
DATA:0040201C TlsCallbacks    dd 3042h                ; DATA XREF: DATA:TlsDirectory↑o
DATA:0040201C                                         ; .idata:import_directory↓o
DATA:00402020 TlsCallbacksEnd dd 0
DATA:00402024                 align 1000h
DATA:00402024 DATA            ends
DATA:00402024
```

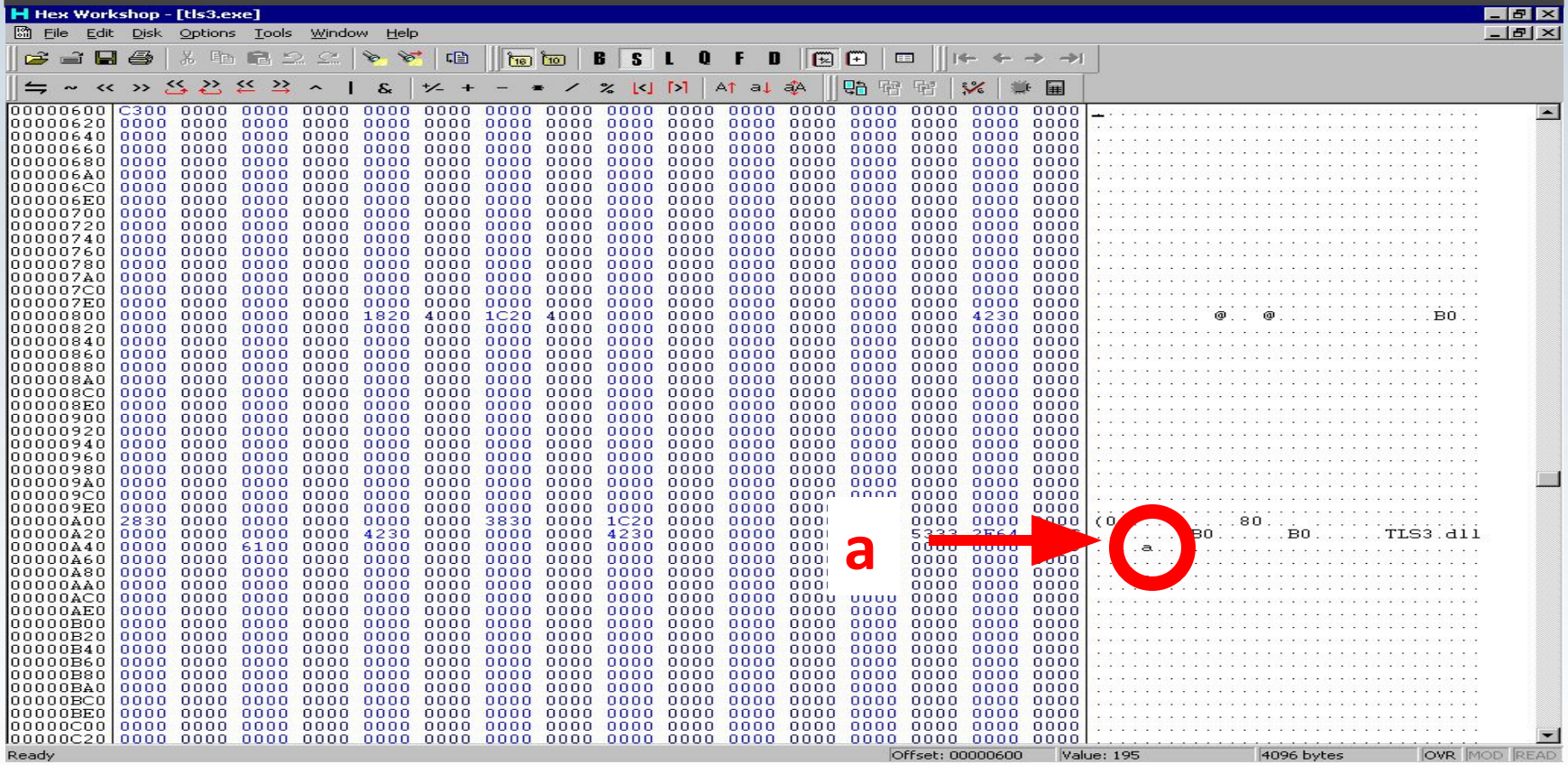
Imported TLS callbac

We know that the TLS callback array can be altered at runtime.
We know that the TLS callbacks can point outside of the image.
Now we are looking at a new way to achieve that.
Imports are resolved before TLS callbacks are called.
So TLS callbacks can be imported addresses!
Let's check the import table.

The Search Goes On



The Search Goes On



The Search Goes On

So the search moves to TLS3.DLL,
and the mysterious function called ‘a’.

‘A’ function

IDA - tls3.dll

File Edit Jump Search View Debug Options Window

IDA View-A

```
CODE:00401000 ; File Name   : C:\Users\Peter\z\tls3.dll
CODE:00401000 ; Format      : Portable executable for 80386 (PE)
CODE:00401000 ; Imagebase   : 400000
CODE:00401000 ; Section 1. (virtual address 00001000)
CODE:00401000 ; Virtual size                  : 00001000 (    4096.)
CODE:00401000 ; Section size in file          : 00000200 (     512.)
CODE:00401000 ; Offset to raw data for section: 00000600
CODE:00401000 ; Flags 60000020: Text Executable Readable
CODE:00401000 ; Alignment     : default
CODE:00401000 ; Exported entry   1. a
CODE:00401000 ; ===========================================================================
CODE:00401000
CODE:00401000 ; Segment type: Pure code
CODE:00401000 ; Segment permissions: Read/Execute
CODE:00401000 CODE            segment para public 'CODE' use32
CODE:00401000                 assume cs:CODE
CODE:00401000                 ;org 401000h
CODE:00401000                 assume es:_reloc, ss:_reloc, ds:CODE, fs:nothing, gs:nothing
CODE:00401000
CODE:00401000 ; ============== S U B R O U T I N E =======================================
CODE:00401000
CODE:00401000
CODE:00401000                 public a
CODE:00401000 a               proc near               ; DATA XREF: HEADER:pe_header↑o
CODE:00401000                                         ; HEADER:pe_section_table↑o ...
CODE:00401000                 push    0               ; uType
CODE:00401002                 push    offset Caption  ; "demo"
CODE:00401007                 push    offset Text     ; "run"
CODE:0040100C                 push    0               ; hWnd
CODE:0040100E                 call    j_MessageBoxA
CODE:0040100E a               endp ; sp-analysis failed
CODE:0040100E
CODE:00401013
CODE:00401013 ; ============== S U B R O U T I N E =======================================
CODE:00401013
CODE:00401013
CODE:00401013 ; BOOL __stdcall start(HINSTANCE hinstDLL,DWORD fdwReason,LPVOID lpReserved)
CODE:00401013                 public start
CODE:00401013 start           proc near               ; DATA XREF: HEADER:pe_header↑o
CODE:00401013                 mov     al, 1           ; DllEntryPoint
CODE:00401015                 retn
CODE:00401015 start           endp
CODE:00401015
```

The ‘Aha’ Moment

So that’s how it’s done.

If we let it run…

Surprise!

Not OK

The code runs.

Really Not OK

Just a little something to add to the workload.