Understanding the PEB Loader Data Structure

The PEB_LDR_DATA structure is a Windows Operating System structure that contains information about all of the loaded modules in the current process.

The OS links to it in the [Process Envirnoment Block]() at offset 0x0C.

Shellcode will typically walk the [PEB_LDR_DATA]() structure and the linked [LDR_MODULE]() structures in order to find the base address of loaded dlls.

When you look at these structures for the first time, it can be allot to try to digest especially if you are not familiar with the further embedded types such as UNICODE_STRING and LIST_ENTRY.

The following graphic depicts teh core of what you need to know. (note I changed the name of the LIST_ENTRY here to mLIST so it didnt conflict with my header files)

```
typedef struct LDR_MODULE
{
    /* 0x00 */ mLIST InLoadOrder;
    //              4 byte Forward Link
    //              4 byte Backward Link
    /* 0x08 */ mLIST InMemOrder;
    /* 0x10 */ mLIST InInitOrder;
    /* 0x18 */ uint32_t DllBase;
    /* 0x1c */ uint32_t EntryPoint;
    /* 0x1f */ uint32_t Reserved;
    /* 0x24 */ UNICODE_STRING FullDllName;
    //              2 byte Length
    //              2 Byte MaxLength
    //              4 byte pointer to Unicode string
    /* 0x2c */ UNICODE_STRING BaseDllName;
}

typedef struct PEB_LDR_DATA
{
     /* 0x00 */ uint32_t Length;
     /* 0x04 */ uint8_t Initialized[4];
     /* 0x08 */ uint32_t SsHandle;
     /* 0x0c */ mLIST InLoadOrder;
     /* 0x14 */ mLIST InMemOrder;
     /* 0x1c */ mLIST InInitOrder;
     /* 0x24 */ uint8_t EntryInProgress;
}
```

Trying to follow code which makes use of these structures can be equally confusing until you figure out how the lists are interconnected as well.

First, lets start with some high level concepts.

1. Peb loader data is the head of the list. It contains both forward and backward links to the other elements.
2. Each dll gets its own Loader Module structure. It is these structures which are linked to the other entries.
3. Windows organizes the loaded dll list in 3 ways. According to the order the dlls
    1. were loaded by the windows loader
    2. are found in the memory layout
    3. were initilized
4. On windows XP, certain core dlls are always found at specific offsets in the list. Shellcode often takes advantage of this when they are locating dlls. The important bits are:

```
5. inloadorder = process, ntdll, kernel32, ...
6. inmemorder  = process, ntdll,kernel32, ...
7. ininitorder = ntdll, kernel32, ... (no process entry)
```

The way the actual structures work, makes sense once you understand their layout. Consider the following which represents a complete PEB Loader Data and Loader Module list for a simple process.

```
PEB Loader Data
00241EA0  00000028
00241EA4  BAADF001
00241EA8  00000000
00241EAC  00241EE0  inloadorder.flink
00241EB0  00242010             .blink
00241EB4  00241EE8  inmemorder.flink
00241EB8  00242018
00241EBC  00241F58  ininitorder.flink
00241EC0  00242020


loader module entry 1
00241EE0  00241F48
00241EE4  00241EAC
00241EE8  00241F50
00241EEC  00241EB4
00241EF0  00000000
00241EF4  00000000
00241EF8  00400000  shellcod.00400000
00241EFC  00401000  shellcod.
00241F00  00006000
00241F04  006E006C
00241F08  00020780  UNICODE "C:\shellcode.exe_"
00241F0C  001E001C
00241F10  000207D0  UNICODE "shellcode.exe_"


loader module entry 2
00241F48  00242010
00241F4C  00241EE0
00241F50  00242018
00241F54  00241EE8
00241F58  00242020
00241F5C  00241EBC
00241F60  7C900000  ntdll.7C900000
```

```
00241F64  7C9120F8  ntdll.
00241F68  000B2000
00241F6C  0208003A
00241F70  7C980048  UNICODE "C:\WINDOWS\system32\ntdll.dll"
00241F74  00140012
00241F78  7C92040C  UNICODE "ntdll.dll"


loader module entry 3
00242010  00241EAC    ;flink points back to peb.inloadorder.flink
00242014  00241F48    ;points back to entry 2 inloadorder.flink
00242018  00241EB4
0024201C  00241F50
00242020  00241EBC
00242024  00241F58
00242028  7C800000  kernel32.7C800000
0024202C  7C80B64E  kernel32.
00242030  000F6000
00242034  00420040
00242038  00241FB0  UNICODE "C:\WINDOWS\system32\kernel32.dll"
0024203C  001A0018
00242040  00241FD8  UNICODE "kernel32.dll"
```

Perhaps the easiest way to become familiar with the layout of these lists is to open up a simple executable in Olly, click the dump window to make it active, and press control+G to goto expression. Type in **fs:[30]** Which will bring you to the parent PEB structure. Right click and view the data as Long->Address.

You can even double click on the first entry in the address column to have it display the offsets relative to the offset you clicked. From here you can right click on entry 0x0C and choose follow in dump which will take you to the PEB_LDR_DATA structure.

In this manner you can interactively follow the lists and see how the data changes.

Now lets explore the listing given above.

You can easily now see that each LIST_ENTRY field links to the next by following the offsets. (The hex number on the left is the memory address, the next hex number is the data value at that address. If there is any data in the 3rd column, it is either a comment or a data dereference by olly)

If you look closley you will notice a couple things.

- Each list.flink points to the next dlls corrosponding list.flink. (IE The InLoadOrder list links to the next items InLoadOrder list)
- At the end of the list, the last items forward link, points back to the Peb loader data list head.
- The process entry (for the .exe) is not linked into the .InInitilizationOrder list
- Each entries back link, points to the last items forward link.

One other thing that makes sense in hindsight, but was confusing at the time is how the offset for the basedll name, or module base address changes depending on which list you are walking.

If we were walking the InInitilizationOrder List, you would see something like this

```
PEB Loader
00241EA0  00000028
00241EA4  BAADF001
```

```
00241EA8  00000000
00241EAC  00241EE0  inloadorder.flink
00241EB0  00242010
00241EB4  00241EE8  inmemorder.flink
00241EB8  00242018
00241EBC  00241F58  ininitorder.flink
00241EC0  00242020

in init order entry 1
00241F58  00242020
00241F5C  00241EBC
00241F60  7C900000  ntdll.7C900000
00241F64  7C9120F8  ntdll.
00241F68  000B2000
00241F6C  0208003A
00241F70  7C980048  UNICODE "C:\WINDOWS\system32\ntdll.dll"
00241F74  00140012
00241F78  7C92040C  UNICODE "ntdll.dll"
```

See how the module base address is now at flink+0x8 ?

This is because the list you are walking is already 0x10 bytes into the loader module list structure by the time you get there. If you had been walking the InLoadOrder list, then the dll base would be at offset 0x18

```
$ ==>    >00242010  ;start of ldr_module structure, InLoadOrderList.flink
$+4      >00241EE0
$+8      >00242018
$+C      >00241EE8
$+10     >00242020
$+14     >00241EBC
$+18     >7C900000  ntdll.7C900000
$+1C     >7C9120F8  ntdll.
$+20     >000B2000
$+24     >0208003A
$+28     >7C980048  UNICODE "C:\WINDOWS\system32\ntdll.dll"
$+2C     >00140012
$+30     >7C92040C  UNICODE "ntdll.dll"

$ ==>    >00242020  ;0x10 bytes into ldr_module, InInitOrder.flink
$+4      >00241EBC
$+8      >7C900000  ntdll.7C900000
$+C      >7C9120F8  ntdll.
$+10     >000B2000
$+14     >0208003A
$+18     >7C980048  UNICODE "C:\WINDOWS\system32\ntdll.dll"
$+1C     >00140012
$+20     >7C92040C  UNICODE "ntdll.dll"
```

Initially this can be a source of confusion, but once you see it in action, it makes sense.

i guess those are the main things I wanted to show about working with the loader data lists. Looking at just the structures and blobs of hex data is not always a very friendly exercise. I googled a bit and couldnt find any documents like this so figured I would put this out there to help others along.

-dzzie