漏洞银行二进制安全系列讲座（二） —— 加密解密与漏洞利用

第十二讲：PE文件结构讲解与特征码免杀原理

主讲：K1ght（漏洞银行安全专家）

讲师互动丨课后交流丨资料共享 二进制群：**565893809**

漏洞银行微信公众号：**BUG_BANK**

漏洞银行 | BUGBANK
www.bugbank.cn

主要内容

01
PE结构讲解

02
特征码免杀原理

01
PE结构讲解

PE结构

PE（Portable Executable）文件简介

PE（Portable Executable）文件是Windows操作系统下使用的可执行文件格式。它是微软在UNIX平台的COFF（通用对象文件格式）基础上制作而成。最初设计用来提高程序在不同操作系统上的移植性，但实际上这种文件格式仅用在Windows系列操作系统下。

PE文件是指32位可执行文件，也称为PE32。64位的可执行文件称为PE+或PE32+,是PE(PE32)的一种扩展形式（请注意不是PE64)。

PE结构

当一个PE文件被执行时，PE装载器首先检查DOS header里的PE header的偏移量。如果找到，则直接跳转到PE header的位置。

当PE装载器跳转到PE header后，第二步要做的就是检查PE header是否有效。如果该PE header有效，就跳转到PE header的尾部。

紧跟PE header尾部的是节表。PE装载器执行完第二步后开始读取节表中的节段信息，并采用文件映射（在执行一个PE文件的时候，Windows并不在一开始就将整个文件读入内存，而是采用与内存映射的机制，也就是说，Windows装载器在装载的时候仅仅建立好虚拟地址和PE文件之间的映射关系，只有真正执行到某个内存页中的指令或者访问某一页中的数据时，这个页面才会被从磁盘提交到物理内存，这种机制使文件装入的速度和文件大小没有太大的关系）的方法将这些节段映射到内存，同时附上节表里指定节段的读写属性。

PE文件映射入内存后，PE装载器将继续处理PE文件中类似 import table （输入表）的逻辑部分

这四个步骤便是PE文件的执行顺序，具体细节读者可以参考相关文档。

PE结构

文件中使用偏移（offset），内存中使用VA（Virtual Address，虚拟地址）来表示位置。

VA指进程虚拟内存的绝对地址，RVA(Relative Virtual Address，相对虚拟地址)是指从某基准位置（ImageBase）开始的相对地址。VA与RVA满足下面的换算关系：

RVA+ImageBase=VA

PE头内部信息大多是RVA形式存在。原因在于（主要是DLL）加载到进程虚拟内存的特定位置时，该位置可能已经加载了其他的PE文件（DLL）。此时必须通过重定向（Relocation）将其加载到其他空白的位置，若PE头信息使用的是VA，则无法正常访问。因此使用RVA来重定向信息，即使发生了重定向，只要相对于基准位置的相对位置没有变化，就能正常访问到指定信息，不会出现任何问题。

当PE文件被执行时，PE装载器会为进程分配4G的虚拟地址空间，然后把程序所占用的磁盘空间作为虚拟内存映射到这个4GB的虚拟地址空间中。一般情况下，会映射到虚拟地址空间中的0X400000的位置。

PE结构

- PE头结构：DOS头+NT头
- 其中比较重要的有e_magic和e_lfanew，由图可知
- e_magic的值为4D5A，e_lfanew的值为000000C0（注意不是C0000000，详见我的上一篇文章)
- WORD占2个字节，LONG占4个字节，刚好是30个WORD和1个LONG，从00000000到0000003F
- 即使没有DOS存根，文件也能正常执行

```
typedef struct _IMAGE_DOS_HEADER { // DOS的.EXE头部
  USHORT e_magic; // DOS签名"MZ-->Mark Zbikowski（设计了DOS的工程师）"
  USHORT e_cblp; // 文件最后页的字节数
  USHORT e_cp; // 文件页数
  USHORT e_crlc; // 重定义元素个数
  USHORT e_cparhdr; // 头部尺寸，以段落为单位
  USHORT e_minalloc; // 所需的最小附加段
  USHORT e_maxalloc; // 所需的最大附加段
  USHORT e_ss; // 初始的SS值（相对偏移量）
  USHORT e_sp; // 初始的SP值
  USHORT e_csum; // 校验和
  USHORT e_ip; // 初始的IP值
  USHORT e_cs; // 初始的CS值（相对偏移量）
  USHORT e_lfarlc; // 重分配表文件地址
  USHORT e_ovno; // 覆盖号
  USHORT e_res[4]; // 保留字
  USHORT e_oemid; // OEM标识符（相对e_oeminfo）
  USHORT e_oeminfo; // OEM信息
  USHORT e_res2[10]; // 保留字
  LONG e_lfanew; // 指示NT头的偏移（根据不同文件拥有可变值）
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

PE结构

- PE头：文件头+可选头
- 1.Machine 每个CPU拥有唯一的Machine码，兼容32位 Intel X86芯片的Machine码为14C（如图）。以下是定义在winnt.h文件中的Machine码：
- e_magic的值为4D5A，e_lfanew的值为000000C0（注意不是C0000000，详见我的上一篇文章)
- WORD占2个字节，LONG占4个字节，刚好是30个WORD和1个LONG，从00000000到0000003F

```
typedef struct _IMAGE_NT_HEADERS {
        DWORD Signature;
        IMAGE_FILE_HEADER FileHeader;
        IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

Signature：类似于DOS头中的e_magic，其高16位是0，低16是0x4550
IMAGE_FILE_HEADER：IMAGE_FILE_HEADER是PE文件头，定义如下：
    typedef struct _IMAGE_FILE_HEADER {
        WORD    Machine;
        WORD    NumberOfSections;
        DWORD   TimeDateStamp;
        DWORD   PointerToSymbolTable;
        DWORD   NumberOfSymbols;
        WORD    SizeOfOptionalHeader;
        WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE结构

- NT头
- **1.Machine** 每个CPU拥有唯一的Machine码，兼容32位

  Intel X86芯片的Machine码为14C

```
#define IMAGE_FILE_MACHINE_UNKNOWN           0
#define IMAGE_FILE_MACHINE_I386              0x014c  // Intel 386.
#define IMAGE_FILE_MACHINE_AMD64             0x8664  // AMD64 (K8)
#define IMAGE_FILE_MACHINE_IA64              0x0200  // Intel 64
```

```
typedef struct _IMAGE_NT_HEADERS {
        DWORD Signature;
        IMAGE_FILE_HEADER FileHeader;
        IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

Signature：类似于DOS头中的e_magic，其高16位是0，低16是0x4550
IMAGE_FILE_HEADER：IMAGE_FILE_HEADER是PE文件头，定义如下：
    typedef struct _IMAGE_FILE_HEADER {
        WORD    Machine;
        WORD    NumberOfSections;
        DWORD   TimeDateStamp;
        DWORD   PointerToSymbolTable;
        DWORD   NumberOfSymbols;
        WORD    SizeOfOptionalHeader;
        WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE结构

- NT头
- **2.NumberOfEsctions**
- PE文件把代码，数据，资源等依据属性分类到各节中储存。
- NumberOfEsctions指文件中存在的节段（又称节区）数量，也就是节表中的项数。该值一定要大于0，且当定义的节段数与实际不符时，将发生运行错误。

```
typedef struct _IMAGE_NT_HEADERS {
        DWORD Signature;
        IMAGE_FILE_HEADER FileHeader;
        IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

Signature：类似于DOS头中的e_magic，其高16位是0，低16是0x4550
IMAGE_FILE_HEADER：IMAGE_FILE_HEADER是PE文件头，定义如下：
    typedef struct _IMAGE_FILE_HEADER {
        WORD    Machine;
        WORD    NumberOfSections;
        DWORD   TimeDateStamp;
        DWORD   PointerToSymbolTable;
        DWORD   NumberOfSymbols;
        WORD    SizeOfOptionalHeader;
        WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE结构

- NT头
- 3.SizeOfOptionalHeader
- IMAGE_NT_HEADERS结构最后一个成员IMAGE_OPTIONAL_HEADER32。
- SizeOfOptionalHeader用来指出IMAGE_OPTIONAL_HEADER32结构体的长度。PE装载器需要查看SizeOfOptionalHeader的值，从而识别IMAGE_OPTIONAL_HEADER32结构体的大小。
- PE32+格式文件中使用的是IMAGE_OPTIONAL_HEADER64结构体，这两个结构体尺寸是不相同的，所以需要在SizeOfOptionalHeader中指明大小。

```
typedef struct _IMAGE_NT_HEADERS {
        DWORD Signature;
        IMAGE_FILE_HEADER FileHeader;
        IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

Signature：类似于DOS头中的e_magic，其高16位是0，低16是0x4550
IMAGE_FILE_HEADER：IMAGE_FILE_HEADER是PE文件头，定义如下：
    typedef struct _IMAGE_FILE_HEADER {
        WORD    Machine;
        WORD    NumberOfSections;
        DWORD   TimeDateStamp;
        DWORD   PointerToSymbolTable;
        DWORD   NumberOfSymbols;
        WORD    SizeOfOptionalHeader;
        WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE结构

- NT头
- 4.Characteristics
- 该段用于标识文件的属性，文件是否是可运行的状态，是否为DLL文件等信息。

```
typedef struct _IMAGE_NT_HEADERS {
        DWORD Signature;
        IMAGE_FILE_HEADER FileHeader;
        IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

Signature：类似于DOS头中的e_magic，其高16位是0，低16是0x4550
IMAGE_FILE_HEADER：IMAGE_FILE_HEADER是PE文件头，定义如下：
    typedef struct _IMAGE_FILE_HEADER {
        WORD    Machine;
        WORD    NumberOfSections;
        DWORD   TimeDateStamp;
        DWORD   PointerToSymbolTable;
        DWORD   NumberOfSymbols;
        WORD    SizeOfOptionalHeader;
        WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE结构

- IMAGE_OPTIONAL_HEADER32：
- 可选头

```
typedef struct _IMAGE_OPTIONAL_HEADER {
        WORD    Magic;
        BYTE    MajorLinkerVersion;
        BYTE    MinorLinkerVersion;
        DWORD   SizeOfCode;
        DWORD   SizeOfInitializedData;
        DWORD   SizeOfUninitializedData;
        DWORD   AddressOfEntryPoint;
        DWORD   BaseOfCode;
        DWORD   BaseOfData;
        DWORD   ImageBase;
        DWORD   SectionAlignment;
        DWORD   FileAlignment;
        WORD    MajorOperatingSystemVersion;
        WORD    MinorOperatingSystemVersion;
        WORD    MajorImageVersion;
        WORD    MinorImageVersion;
        WORD    MajorSubsystemVersion;
        WORD    MinorSubsystemVersion;
        DWORD   Win32VersionValue;
        DWORD   SizeOfImage;
        DWORD   SizeOfHeaders;
        DWORD   CheckSum;
        WORD    Subsystem;
        WORD    DllCharacteristics;
        DWORD   SizeOfStackReserve;
        DWORD   SizeOfStackCommit;
        DWORD   SizeOfHeapReserve;
        DWORD   SizeOfHeapCommit;
        DWORD   LoaderFlags;
        DWORD   NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

- 1.Magic
- 为IMAGE_OPTIONAL_HEADER32时，magic码为10B，为IMAGE_OPTIONAL_HEADER64时，magic码为20B

```
typedef struct _IMAGE_OPTIONAL_HEADER {
        WORD    Magic;
        BYTE    MajorLinkerVersion;
        BYTE    MinorLinkerVersion;
        DWORD   SizeOfCode;
        DWORD   SizeOfInitializedData;
        DWORD   SizeOfUninitializedData;
        DWORD   AddressOfEntryPoint;
        DWORD   BaseOfCode;
        DWORD   BaseOfData;
        DWORD   ImageBase;
        DWORD   SectionAlignment;
        DWORD   FileAlignment;
        WORD    MajorOperatingSystemVersion;
        WORD    MinorOperatingSystemVersion;
        WORD    MajorImageVersion;
        WORD    MinorImageVersion;
        WORD    MajorSubsystemVersion;
        WORD    MinorSubsystemVersion;
        DWORD   Win32VersionValue;
        DWORD   SizeOfImage;
        DWORD   SizeOfHeaders;
        DWORD   CheckSum;
        WORD    Subsystem;
        WORD    DllCharacteristics;
        DWORD   SizeOfStackReserve;
        DWORD   SizeOfStackCommit;
        DWORD   SizeOfHeapReserve;
        DWORD   SizeOfHeapCommit;
        DWORD   LoaderFlags;
        DWORD   NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

- **1.Magic**

  为IMAGE_OPTIONAL_HEADER32时，magic码为10B，为IMAGE_OPTIONAL_HEADER64时，magic码为20B

- **2.AddressOfEntryPoint**

  AddressOfEntryPoint持有EP的RVA值。该值指出程序最先执行的代码起始地址，相当重要。

- **3.ImageBase**

  一般来说，使用开发工具（VB/VC++/Delphi）创建好EXE文件后，其ImageBase值为00400000，DLL文件的ImageBase值为10000000（当然也可以指定其他值）。

  执行PE文件时，PE装载器先创建进程，再将文件载入内存，然后把EIP寄存器的值设置为ImageBase+AddressOfEntryPoint

```
typedef struct _IMAGE_OPTIONAL_HEADER {
        WORD    Magic;
        BYTE    MajorLinkerVersion;
        BYTE    MinorLinkerVersion;
        DWORD   SizeOfCode;
        DWORD   SizeOfInitializedData;
        DWORD   SizeOfUninitializedData;
        DWORD   AddressOfEntryPoint;
        DWORD   BaseOfCode;
        DWORD   BaseOfData;
        DWORD   ImageBase;
        DWORD   SectionAlignment;
        DWORD   FileAlignment;
        WORD    MajorOperatingSystemVersion;
        WORD    MinorOperatingSystemVersion;
        WORD    MajorImageVersion;
        WORD    MinorImageVersion;
        WORD    MajorSubsystemVersion;
        WORD    MinorSubsystemVersion;
        DWORD   Win32VersionValue;
        DWORD   SizeOfImage;
        DWORD   SizeOfHeaders;
        DWORD   CheckSum;
        WORD    Subsystem;
        WORD    DllCharacteristics;
        DWORD   SizeOfStackReserve;
        DWORD   SizeOfStackCommit;
        DWORD   SizeOfHeapReserve;
        DWORD   SizeOfHeapCommit;
        DWORD   LoaderFlags;
        DWORD   NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

◆ **4.SectionAlignment，FileAlignment**

PE文件的Body部分被划分成若干节段，这些节段储存着不同类别的数据。

FileAlignment指定了节段在磁盘文件中的最小单位，而SectionAlignment则指定了节区在内存中的最小单位（SectionAlignment必须大于或者等于FileAlignment）

◆ **5.SizeOfImage**

当PE文件加载到内存时，SizeOfImage指定了PE Image在虚拟内存中所占用的空间大小，一般文件大小与加载到内存中的大小是不同的（节段头中定义了各节装载的位置与占有内存的大小，后面会讲到）

◆ **6.SizeOfHeader**

SizeOfHeader用来指出整个PE头大小。该值必须是FileAlignment的整数倍。第一节段所在位置与SizeOfHeader距文件开始偏移的量相同。

```
typedef struct _IMAGE_OPTIONAL_HEADER {
        WORD    Magic;
        BYTE    MajorLinkerVersion;
        BYTE    MinorLinkerVersion;
        DWORD   SizeOfCode;
        DWORD   SizeOfInitializedData;
        DWORD   SizeOfUninitializedData;
        DWORD   AddressOfEntryPoint;
        DWORD   BaseOfCode;
        DWORD   BaseOfData;
        DWORD   ImageBase;
        DWORD   SectionAlignment;
        DWORD   FileAlignment;
        WORD    MajorOperatingSystemVersion;
        WORD    MinorOperatingSystemVersion;
        WORD    MajorImageVersion;
        WORD    MinorImageVersion;
        WORD    MajorSubsystemVersion;
        WORD    MinorSubsystemVersion;
        DWORD   Win32VersionValue;
        DWORD   SizeOfImage;
        DWORD   SizeOfHeaders;
        DWORD   CheckSum;
        WORD    Subsystem;
        WORD    DllCharacteristics;
        DWORD   SizeOfStackReserve;
        DWORD   SizeOfStackCommit;
        DWORD   SizeOfHeapReserve;
        DWORD   SizeOfHeapCommit;
        DWORD   LoaderFlags;
        DWORD   NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

◆ **7.Subsystem**

Subsystem值用来区分系统驱动文件（*.sys)与普通可执行文件（*.exe，*.dll）。

```
#define IMAGE_SUBSYSTEM_UNKNOWN              0   // Unknown subsystem.
#define IMAGE_SUBSYSTEM_NATIVE               1   // Image doesn&#039;t require a subsystem.   系统驱动
#define IMAGE_SUBSYSTEM_WINDOWS_GUI          2   // Image runs in the Windows GUI subsystem.  窗口应用程序
#define IMAGE_SUBSYSTEM_WINDOWS_CUI          3   // Image runs in the Windows character subsystem.  控制台应用程序
#define IMAGE_SUBSYSTEM_OS2_CUI              5   // image runs in the OS/2 character subsystem.
#define IMAGE_SUBSYSTEM_POSIX_CUI            7   // image runs in the Posix character subsystem.
#define IMAGE_SUBSYSTEM_NATIVE_WINDOWS       8   // image is a native Win9x driver.
#define IMAGE_SUBSYSTEM_WINDOWS_CE_GUI       9   // Image runs in the Windows CE subsystem.
#define IMAGE_SUBSYSTEM_EFI_APPLICATION      10  //
#define IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11   //
#define IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER   12  //
#define IMAGE_SUBSYSTEM_EFI_ROM              13
#define IMAGE_SUBSYSTEM_XBOX                 14
#define IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION 16
```

PE结构

- **8.DataDirectory**
- **数据目录，定义如下：**

typedef struct _IMAGE_DATA_DIRECTORY {

DWORD　VirtualAddress;

DWORD　Size;

} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

可以看出，有地址（VirtualAddress）有大小（Size），数组定义的一定是一个区域，数组每项都有被定义的值，不同项对应不同数据结构，比如导入表，导出表等

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT          1   // Import Directory
```

```
typedef struct _IMAGE_OPTIONAL_HEADER {
        WORD    Magic;
        BYTE    MajorLinkerVersion;
        BYTE    MinorLinkerVersion;
        DWORD   SizeOfCode;
        DWORD   SizeOfInitializedData;
        DWORD   SizeOfUninitializedData;
        DWORD   AddressOfEntryPoint;
        DWORD   BaseOfCode;
        DWORD   BaseOfData;
        DWORD   ImageBase;
        DWORD   SectionAlignment;
        DWORD   FileAlignment;
        WORD    MajorOperatingSystemVersion;
        WORD    MinorOperatingSystemVersion;
        WORD    MajorImageVersion;
        WORD    MinorImageVersion;
        WORD    MajorSubsystemVersion;
        WORD    MinorSubsystemVersion;
        DWORD   Win32VersionValue;
        DWORD   SizeOfImage;
        DWORD   SizeOfHeaders;
        DWORD   CheckSum;
        WORD    Subsystem;
        WORD    DllCharacteristics;
        DWORD   SizeOfStackReserve;
        DWORD   SizeOfStackCommit;
        DWORD   SizeOfHeapReserve;
        DWORD   SizeOfHeapCommit;
        DWORD   LoaderFlags;
        DWORD   NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

Magic：表示可选头的类型。
MajorLinkerVersion和MinorLinkerVersion：链接器的版本号。
SizeOfCode：代码段的长度，如果有多个代码段，则是代码段长度的总和。
SizeOfInitializedData：初始化的数据长度。
SizeOfUninitializedData：未初始化的数据长度。
AddressOfEntryPoint：程序入口的RVA，对于exe这个地址可以理解为WinMain的RVA。
对于DLL，这个地址可以理解为DllMain的RVA，
如果是驱动程序，可以理解为DriverEntry的RVA。
BaseOfCode：代码段起始地址的RVA。
BaseOfData：数据段起始地址的RVA。
ImageBase：映象（加载到内存中的PE文件）的基地址，这个基地址是建议，对于DLL来说，如果无法加载到这个地址，系统会自动为其选择地址。
SectionAlignment：节对齐，PE中的节被加载到内存时会按照这个域指定的值来对齐，比如这个值是0x1000，那么每个节的起始地址的低12位都为0。
FileAlignment：节在文件中按此值对齐，SectionAlignment必须大于或等于FileAlignment。
MajorOperatingSystemVersion、MinorOperatingSystemVersion：所需操作系统的版本号，随着操作系统版本越来越多，这个好像不是那么重要了。
MajorImageVersion、MinorImageVersion：映象的版本号，这个是开发者自己指定的，由连接器填写。
MajorSubsystemVersion、MinorSubsystemVersion：所需子系统版本号。
Win32VersionValue：保留，必须为0。
SizeOfImage：映象的大小，PE文件加载到内存中空间是连续的，这个值指定占用虚拟空间的大小。
SizeOfHeaders：所有文件头（包括节表）的大小，这个值是以FileAlignment对齐的。
CheckSum：映象文件的校验和。
Subsystem：运行该PE文件所需的子系统。

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD    Magic;
    BYTE    MajorLinkerVersion;
    BYTE    MinorLinkerVersion;
    DWORD   SizeOfCode;
    DWORD   SizeOfInitializedData;
    DWORD   SizeOfUninitializedData;
    DWORD   AddressOfEntryPoint;
    DWORD   BaseOfCode;
    DWORD   BaseOfData;
    DWORD   ImageBase;
    DWORD   SectionAlignment;
    DWORD   FileAlignment;
    WORD    MajorOperatingSystemVersion;
    WORD    MinorOperatingSystemVersion;
    WORD    MajorImageVersion;
    WORD    MinorImageVersion;
    WORD    MajorSubsystemVersion;
    WORD    MinorSubsystemVersion;
    DWORD   Win32VersionValue;
    DWORD   SizeOfImage;
    DWORD   SizeOfHeaders;
    DWORD   CheckSum;
    WORD    Subsystem;
    WORD    DllCharacteristics;
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    DWORD   SizeOfHeapReserve;
    DWORD   SizeOfHeapCommit;
    DWORD   LoaderFlags;
    DWORD   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

PE结构

节段（区）头：

节段头是由IMAGE_SECTION_HEADER结构体组成的数组，每个结构体对应一个节段。

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

PE结构

Name：区块名。这是一个由8位的ASCII 码名，用来定义区块的名称。多数区块名都习惯性以一个“.”作为开头（例如：.text）

Virtual Size：该区块表对应的区块的大小，这是区块的数据在没有进行对齐处理前的实际大小。

Virtual Address：该区块装载到内存中的RVA 地址。这个地址是按照内存页来对齐的，因此它的数值总是 SectionAlignment 的值的整数倍。在Microsoft 工具中，第一个快的默认 RVA 总为1000h。在OBJ 中，该字段没有意义地，并被设为0。

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

PE结构

SizeOfRawData：该区块在磁盘中所占的大小。在可执行文件中，该字段是已经被FileAlignment 潜规则处理过的长度。

PointerToRawData：该区块在磁盘中的偏移。这个数值是从文件头开始算起的偏移量。

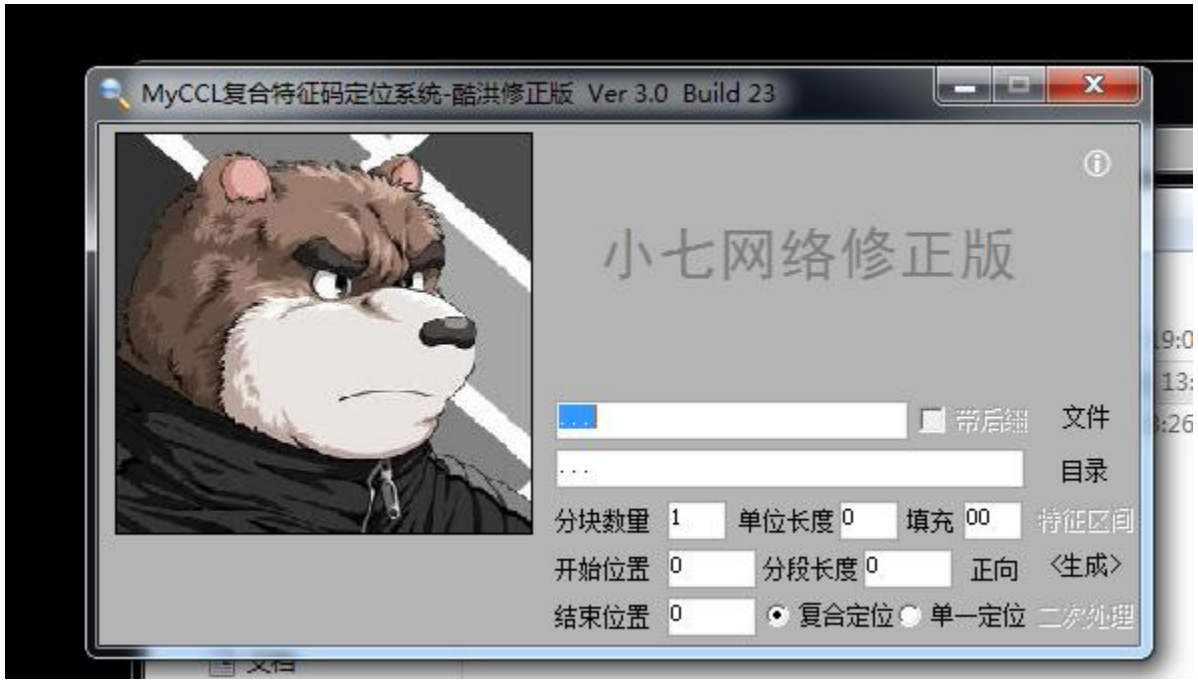
Characteristics：该区块的属性。该字段是按位来指出区块的属性（如代码/数据/可读/可写等）的标志。

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

02
特征码免杀原理

特征码免杀原理

利用MyCCL进行特征码免杀



加入二进制安全研究社群　**QQ群号：565893809**

获取**免费课件**｜结交**讲师伙伴**｜紧跟**后续课程**

微信公众号：BUG_BANK

漏洞银行 | BUGBANK
www.bugbank.cn