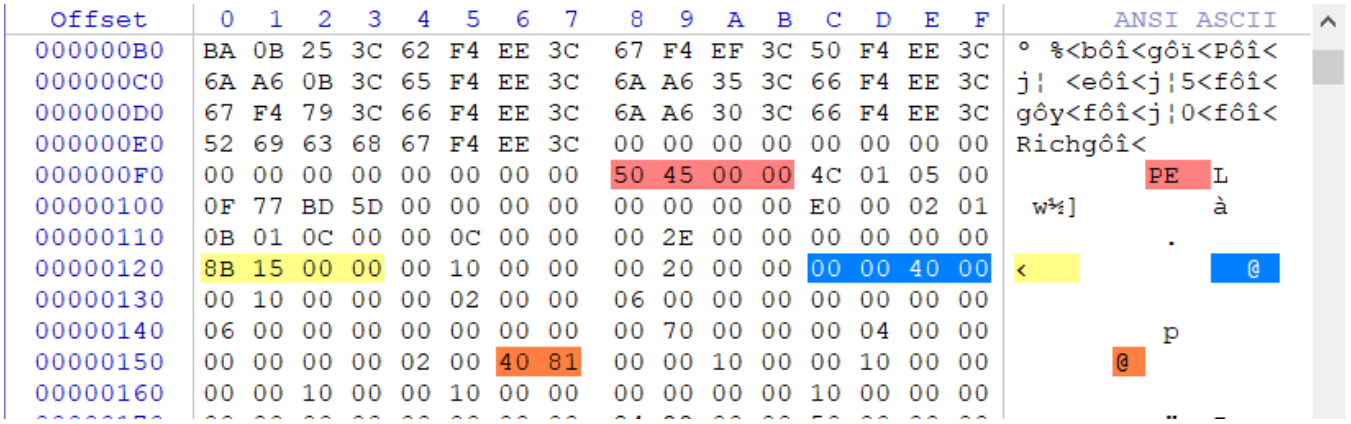
本章将继续探索内核中解析PE文件的相关内容，PE文件中FOA与VA,RVA之间的转换也是很重要的，所谓的FOA是文件中的地址，VA则是内存装入后的虚拟地址，RVA是内存基址与当前地址的相对偏移，本章还是需要用到`《内核解析PE结构导出表》`中所封装的`KernelMapFile()`映射函数，在映射后对其PE格式进行相应的解析，并实现转换函数。

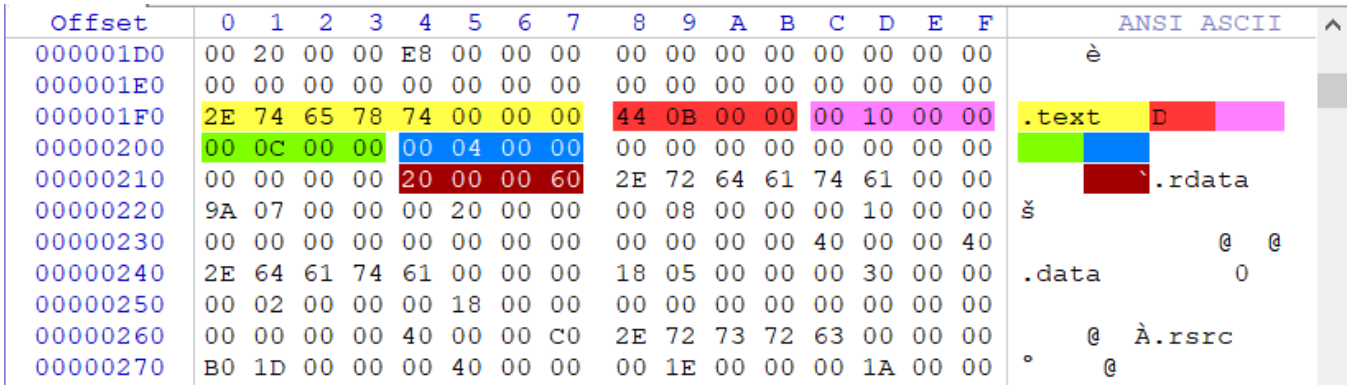
首先先来演示一下内存VA地址与FOA地址互相转换的方式，通过使用WinHEX打开一个二进制文件，打开后我们只需要关注如下蓝色注释为映像建议装入基址，黄色注释为映像装入后的RVA偏移。



通过上方的截图结合PE文件结构图我们可得知`0000158B`为映像装入内存后的RVA偏移，紧随其后的`00400000`则是映像的建议装入基址，为什么是建议而不是绝对？别急后面慢来来解释。

> 通过上方的已知条件我们就可以计算出程序实际装入内存后的入口地址了，公式如下：
> VA(实际装入地址) = ImageBase(基址) + RVA(偏移) => 00400000 + 0000158B = 0040158B

找到了程序的OEP以后，接着我们来判断一下这个`0040158B`属于那个节区，以.text节区为例，下图我们通过观察区段可知，第一处橙色位置`00000B44（节区尺寸）`，第二处紫色位置`00001000（节区RVA）`，第三处`00000C00（文件对齐尺寸）`，第四处`00000400（文件中的偏移）`，第五处`60000020（节区属性）`。

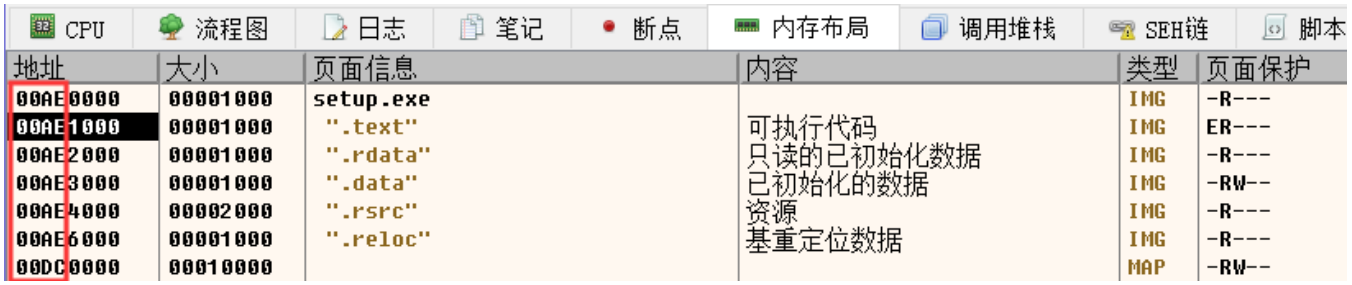


得到了上方text节的相关数据，我们就可以判断程序的OEP到底落在了那个节区中，这里以.text节为例子，计算公式如下：

> 虚拟地址开始位置：节区基地址 + 节区RVA => 00400000 + 00001000 = 00401000
> 虚拟地址结束位置：text节地址 + 节区尺寸 => 00401000 + 00000B44 = 00401B44

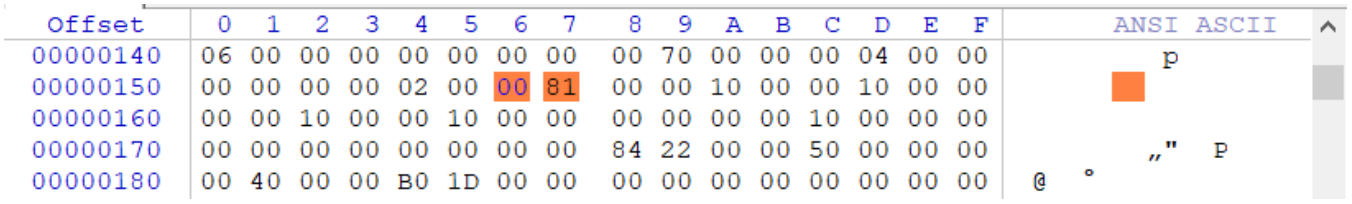
经过计算得知 .text 节所在区间（401000 - 401B44） 你的装入VA地址`0040158B`只要在区间里面就证明在本节区中，此处的VA地址是在`401000 - 401B44`区间内的，则说明它属于.text节。

经过上面的公式计算我们知道了程序的OEP位置是落在了.text节，此时你兴致勃勃的打开x64DBG想去验证一下公式是否计算正确不料，这地址根本不是400000开头啊，这是什么鬼？



上图中出现的这种情况就是关于随机基址的问题，在新版的VS编译器上存在一个选项是否要启用随机基址(默认启用)，至于这个随机基址的作用，猜测可能是为了防止缓冲区溢出之类的烂七八糟的东西。

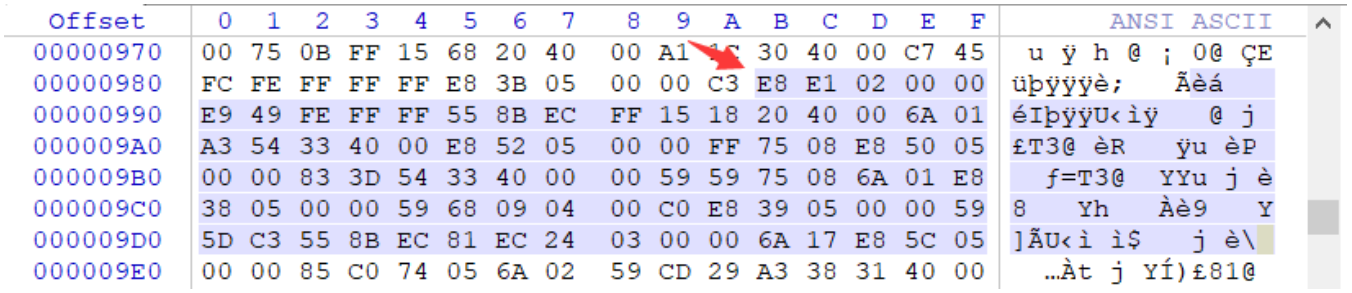
为了方便我们调试，我们需要手动干掉它，其对应到PE文件中的结构为 IMAGE_NT_HEADERS -> IMAGE_OPTIONAL_HEADER -> DllCharacteristics 相对于PE头的偏移为90字节，只需要修改这个标志即可，修改方式 `x64：6081 改 2081` 相对于 `x86：4081 改 0081` 以X86程序为例，修改后如下图所示。



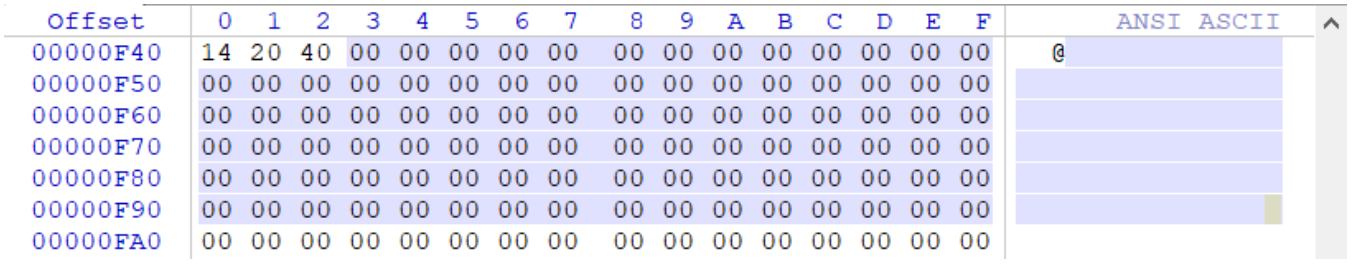
经过上面对标志位的修改，程序再次载入就能够停在 `0040158B` 的位置，也就是程序的OEP，接下来我们将通过公式计算出该OEP对应到文件中的位置。

> .text(节首地址) = ImageBase + 节区RVA => 00400000 + 00001000 = 00401000
> VA(虚拟地址) = ImageBase + RVA(偏移) => 00400000 + 0000158B = 0040158B
> RVA(相对偏移) = VA - (.text节首地址) => 0040158B - 00401000 = 58B
> FOA(文件偏移) = RVA + .text节对应到文件中的偏移 => 58B + 400 = 98B

经过公式的计算，我们找到了虚拟地址 `0040158B` 对应到文件中的位置是 `98B`，通过WinHEX定位过去，即可看到OEP处的机器码指令了。



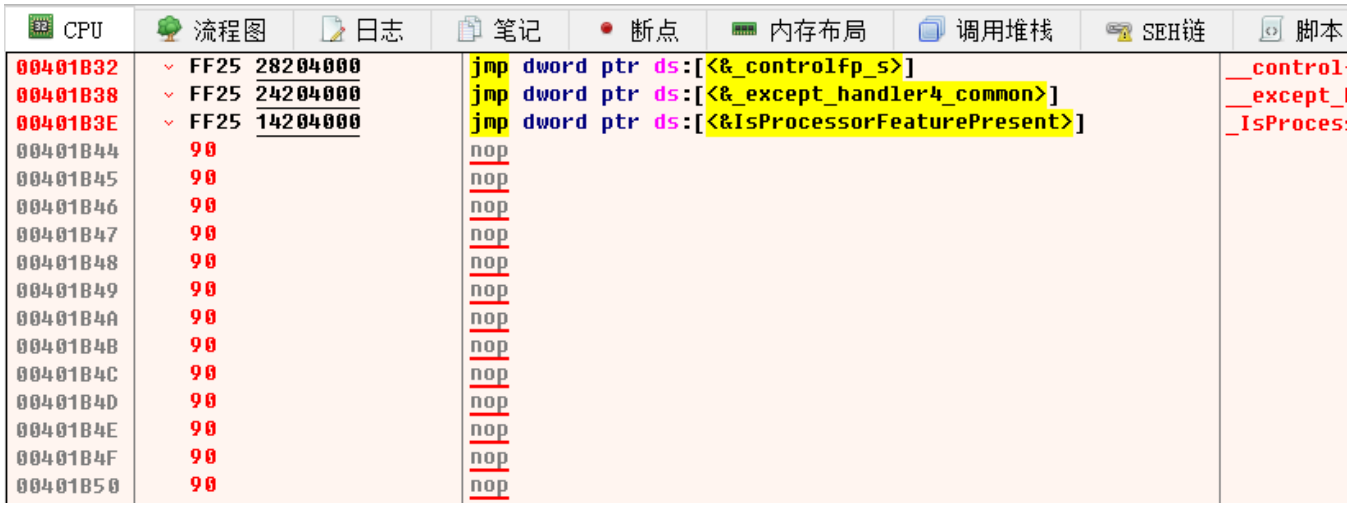
接着我们来计算一下.text节区的结束地址，通过文件的偏移加上文件对齐尺寸即可得到.text节的结束地址 `400+C00=1000`，那么我们主要就在文件偏移为(98B - 1000)在该区间中找空白的地方，此处我找到了在文件偏移为1000之前的位置有一段空白区域，如下图：



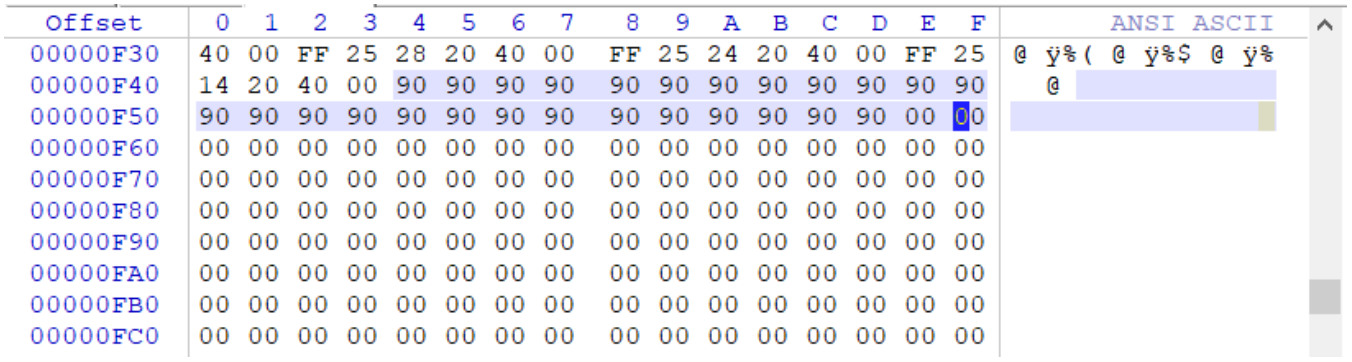
接着我么通过公式计算一下文件偏移为 `0xF43` 的位置，其对应到VA虚拟地址是多少，公式如下：

> .text(节首地址) = ImageBase + 节区RVA => 00400000 + 00001000 = 00401000
> VPK(实际大小) = (text节首地址 - ImageBase) - 实际偏移 => 401000-400000-400 = C00
> VA(虚拟地址) = FOA(.text节) + ImageBase + VPK => F43+400000+C00 = 401B43

计算后直接X64DBG跳转过去，我们从 `00401B44` 的位置向下全部填充为90(nop)，然后直接保存文件。



再次使用WinHEX查看文件偏移为 `0xF43` 的位置，会发现已经全部替换成了90指令，说明计算正确。



到此文件偏移与虚拟偏移的转换就结束了，那么这些功能该如何实现呢，接下来将以此实现这些转换细节。

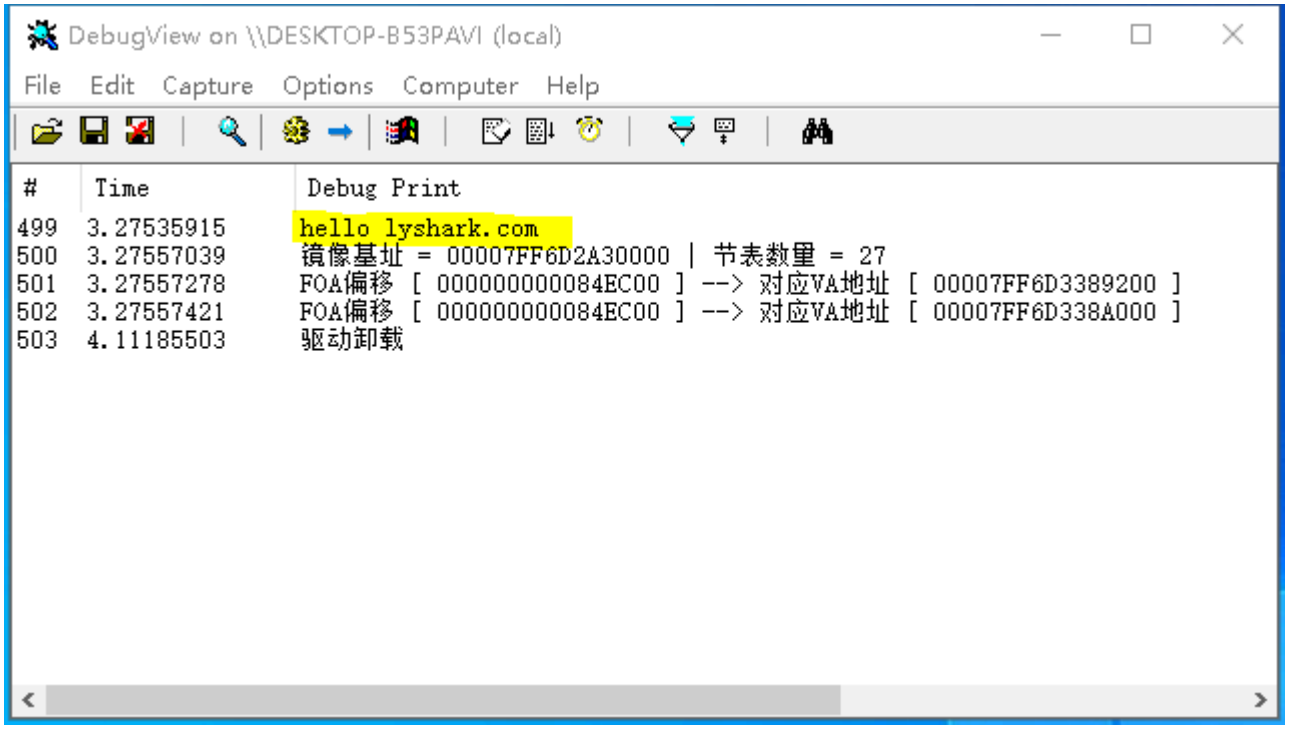
FOA转换为VA

首先来实现将 `FOA` 地址转换为 `VA` 地址，这段代码实现起来很简单，如下所示，此处将 `dwFOA` 地址 `0x84EC00` 转换为对应内存的虚拟地址。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    NTSTATUS status = STATUS_SUCCESS;
    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    PVOID pBaseAddress = NULL;
    UNICODE_STRING FileName = { 0 };

    // 初始化字符串
    RtlInitUnicodeString(&FileName, L"\\??\\C:\\Windows\\System32\\ntoskrnl.exe");

    // 内存映射文件
```

```
    status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress);
    if (!NT_SUCCESS(status))
    {
        return 0;
    }

    // 获取PE头数据集
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pBaseAddress;
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader->e_lfanew);
    PIMAGE_SECTION_HEADER pSection = IMAGE_FIRST_SECTION(pNtHeaders);
    PIMAGE_FILE_HEADER pFileHeader = &pNtHeaders->FileHeader;

    DWORD64 dwFOA = 0x84EC00;

    DWORD64 ImageBase = pNtHeaders->OptionalHeader.ImageBase;
    DWORD NumberOfSectinsCount = pNtHeaders->FileHeader.NumberOfSections;
    DbgPrint("镜像基址 = %p | 节表数量 = %d \n", ImageBase, NumberOfSectinsCount);

    for (int each = 0; each < NumberOfSectinsCount; each++)
    {
        DWORD64 PointerRawStart = pSection[each].PointerToRawData;
     // 文件偏移开始位置
        DWORD64 PointerRawEnds = pSection[each].PointerToRawData + pSection[each].SizeOfRawData;  // 文件偏移结束位置
        // DbgPrint("文件开始偏移 = %p | 文件结束偏移 = %p \n", PointerRawStart, PointerRawEnds);

        if (dwFOA >= PointerRawStart && dwFOA <= PointerRawEnds)
        {
            DWORD64 RVA = pSection[each].VirtualAddress + (dwFOA - pSection[each].PointerToRawData);     // 计算出RVA
            DWORD64 VA = RVA + pNtHeaders->OptionalHeader.ImageBase;
            // 计算出VA
            DbgPrint("FOA偏移 [ %p ] --> 对应VA地址 [ %p ] \n", dwFOA, VA);
        }
    }

    ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
    ZwClose(hSection);
    ZwClose(hFile);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行效果如下所示，此处之所以出现两个结果是因为没有及时返回，一般我们取第一个结果就是最准确的；



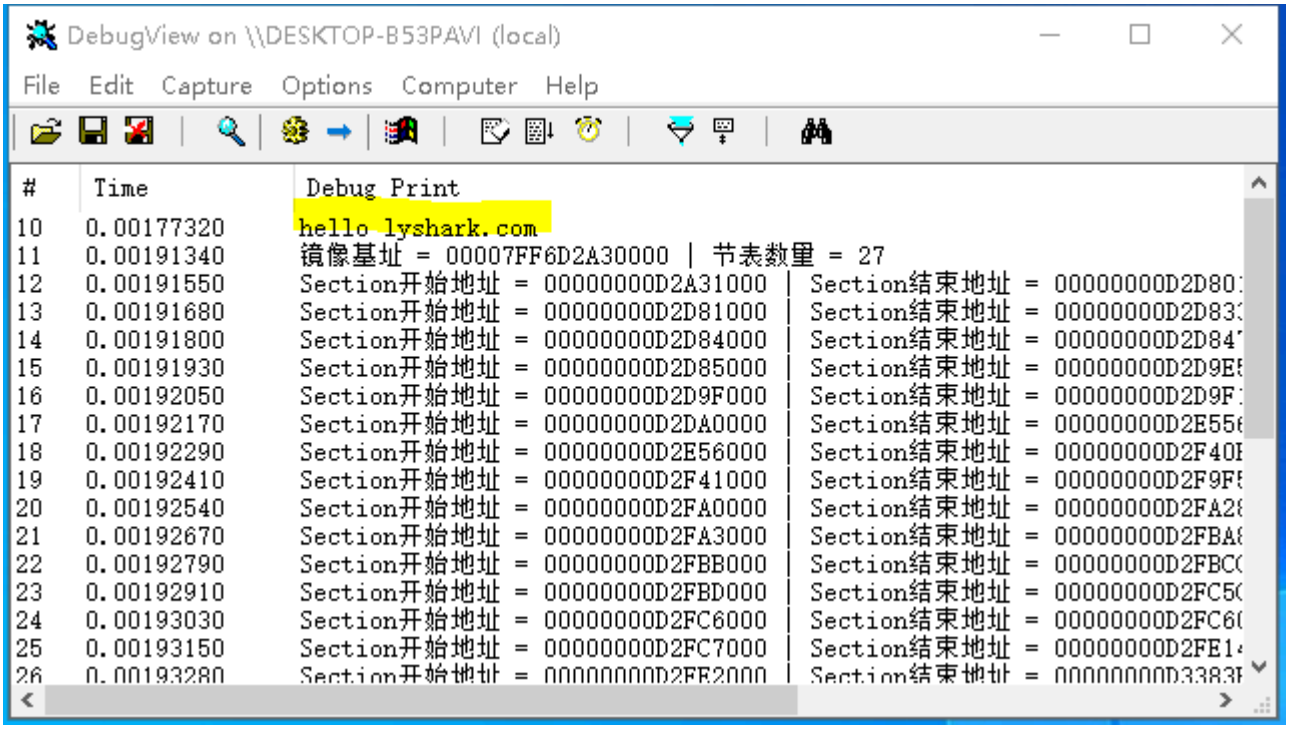
VA转换为FOA

将VA内存地址转换为FOA文件偏移，代码与如上基本保持一致。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    NTSTATUS status = STATUS_SUCCESS;
    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    PVOID pBaseAddress = NULL;
    UNICODE_STRING FileName = { 0 };

    // 初始化字符串
    RtlInitUnicodeString(&FileName, L"\\??\\C:\\Windows\\System32\\ntoskrnl.exe");

    // 内存映射文件
    status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress);
    if (!NT_SUCCESS(status))
    {
        return 0;
    }

    // 获取PE头数据集
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pBaseAddress;
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader-
>e_lfanew);
    PIMAGE_SECTION_HEADER pSection = IMAGE_FIRST_SECTION(pNtHeaders);
    PIMAGE_FILE_HEADER pFileHeader = &pNtHeaders->FileHeader;

    DWORD64 dwVA = 0x00007FF6D3389200;
    DWORD64 ImageBase = pNtHeaders->OptionalHeader.ImageBase;
```

```
    DWORD NumberOfSectinsCount = pNtHeaders->FileHeader.NumberOfSections;
    DbgPrint("镜像基址 = %p | 节表数量 = %d \n", ImageBase, NumberOfSectinsCount);

    for (DWORD each = 0; each < NumberOfSectinsCount; each++)
    {
        DWORD Section_Start = ImageBase + pSection[each].VirtualAddress;
             // 获取节的开始地址
        DWORD Section_Ends = ImageBase + pSection[each].VirtualAddress +
pSection[each].Misc.VirtualSize; // 获取节的结束地址

        DbgPrint("Section开始地址 = %p | Section结束地址 = %p \n", Section_Start,
Section_Ends);

        if (dwVA >= Section_Start && dwVA <= Section_Ends)
        {
            DWORD RVA = dwVA - pNtHeaders->OptionalHeader.ImageBase;
           // 计算RVA
            DWORD FOA = pSection[each].PointerToRawData + (RVA -
pSection[each].VirtualAddress);       // 计算FOA

            DbgPrint("VA偏移 [ %p ] --> 对应FOA地址 [ %p ] \n", dwVA, FOA);
        }
    }

    ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
    ZwClose(hSection);
    ZwClose(hFile);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行效果如下所示，此处没有出现想要的结果是因为我们当前的VA内存地址并非实际装载地址，仅仅是PE磁盘中的地址，此处如果换成内存中的PE则可以提取出正确的结果；



RVA转换为FOA

将相对偏移地址转换为FOA文件偏移地址，此处仅仅只是多了一步`pNtHeaders->OptionalHeader.ImageBase + dwRVA`RVA转换为VA的过程其转换结果与VA转FOA一致。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    NTSTATUS status = STATUS_SUCCESS;
    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    PVOID pBaseAddress = NULL;
    UNICODE_STRING FileName = { 0 };

    // 初始化字符串
    RtlInitUnicodeString(&FileName, L"\\??\\C:\\Windows\\System32\\ntoskrnl.exe");

    // 内存映射文件
    status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress);
    if (!NT_SUCCESS(status))
    {
        return 0;
    }

    // 获取PE头数据集
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pBaseAddress;
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader-
>e_lfanew);
    PIMAGE_SECTION_HEADER pSection = IMAGE_FIRST_SECTION(pNtHeaders);
    PIMAGE_FILE_HEADER pFileHeader = &pNtHeaders->FileHeader;
```

```
    DWORD64 dwRVA = 0x89200;
    DWORD64 ImageBase = pNtHeaders->OptionalHeader.ImageBase;
    DWORD NumberOfSectinsCount = pNtHeaders->FileHeader.NumberOfSections;
    DbgPrint("镜像基址 = %p | 节表数量 = %d \n", ImageBase, NumberOfSectinsCount);

    for (DWORD each = 0; each < NumberOfSectinsCount; each++)
    {
        DWORD Section_Start = pSection[each].VirtualAddress;
 // 计算RVA开始位置
        DWORD Section_Ends = pSection[each].VirtualAddress +
pSection[each].Misc.VirtualSize; // 计算RVA结束位置

        if (dwRVA >= Section_Start && dwRVA <= Section_Ends)
        {
            DWORD VA = pNtHeaders->OptionalHeader.ImageBase + dwRVA;
         // 得到VA地址
            DWORD FOA = pSection[each].PointerToRawData + (dwRVA -
pSection[each].VirtualAddress);    // 得到FOA
            DbgPrint("RVA偏移 [ %p ] --> 对应FOA地址 [ %p ] \n", dwRVA, FOA);
        }
    }

    ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
    ZwClose(hSection);
    ZwClose(hFile);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行效果如下所示;