2024腾讯游戏安全竞赛决赛题解

参赛人员信息

分析

自行加载loader.sys, 找到用户名为administrator的KEY，作为答案提交（1分）

Key 和 User 默认读 `C:\card.txt`，如果找不到或者是错误，那么加载会失败，于是想到 hook NtCreateFile 获取到文件句柄，再 hook NtReadFile 找到文件内容写入的地址。

```
#include <ntifs.h>
#include <ntdef.h>
#include <ntstatus.h>
#include <ntddk.h>
#define MAX_BACKTRACE_DEPTH 20
#define SYMBOL L"\\??\\xia0ji2333"
#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)

UINT64 BaseAddr=NULL, DLLSize=0;

void DeleteDevice(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start delete device\n"), __LINE__);
    if (pDriver->DeviceObject) {
        UNICODE_STRING Sym;
        RtlInitUnicodeString(&Sym, SYMBOL);//CreateFile
        kprintf(("Line %d:xia0ji233: Delete Symbol\n"), __LINE__);
        IoDeleteSymbolicLink(&Sym);
        kprintf(("Line %d:xia0ji233: Delete Device\n"), __LINE__);
        IoDeleteDevice(pDriver->DeviceObject);
    }
    kprintf(("Line %d:xia0ji233: end delete device\n"), __LINE__);
}

HANDLE FileHandler = NULL;

char newcode[] = {
    0x48,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//mov rax,xxx
    0xFF,0xE0//jmp rax
};
char oldcode[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,
};

char newcode2[] = {
    0x48,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//mov rax,xxx
```

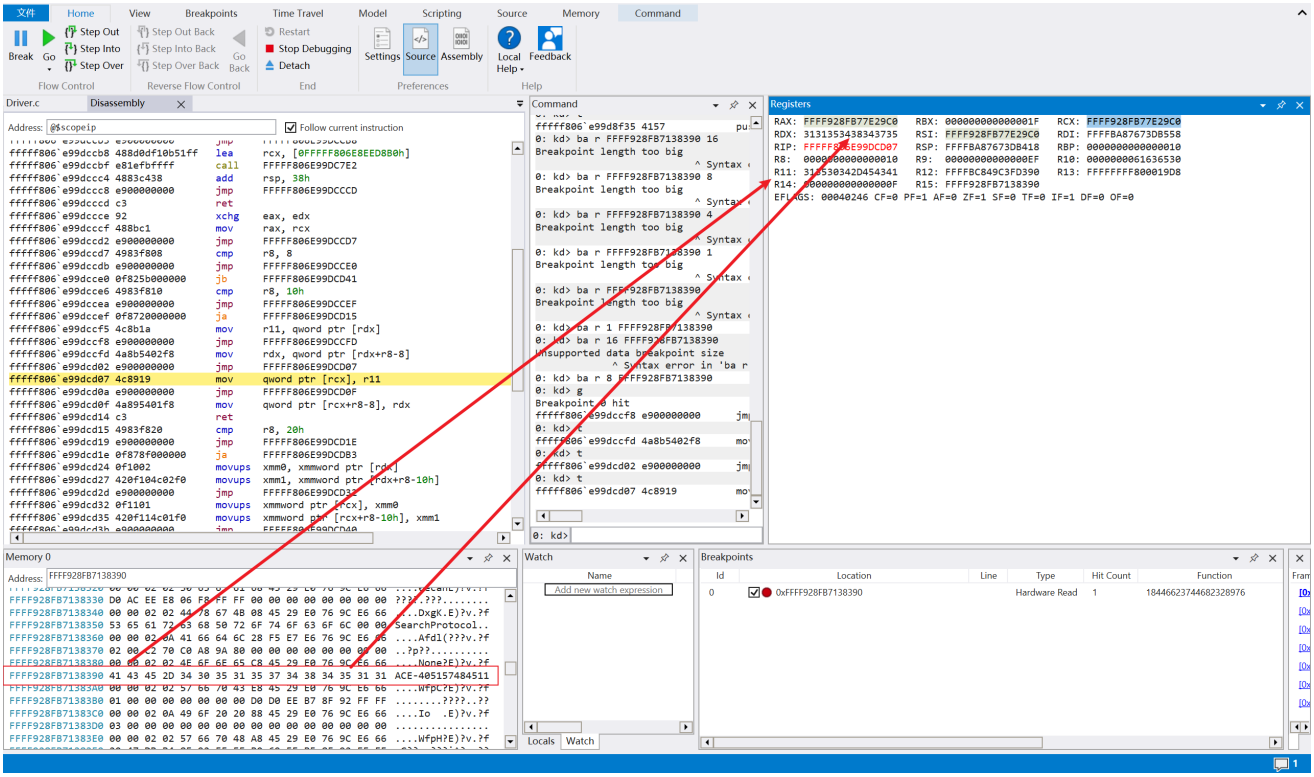
```
    0xFF,0xE0//jmp rax
};
char oldcode2[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,
};

char* target;
char* target2;
KIRQL WPOFFx64()
{
    KIRQL irql = KeRaiseIrqlToDpcLevel();
    UINT64 cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}

void WPONx64(KIRQL irql)
{
    UINT64 cr0 = __readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

NTSTATUS Unhook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = oldcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Unhook2() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode2); i++) {
        target2[i] = oldcode2[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Hook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = newcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Hook2() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode2); i++) {
```

```
        target2[i] = newcode2[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

PDRIVER_OBJECT g_Object = NULL;
typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;//驱动的进入点 DriverEntry
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;//驱动的满路径
    UNICODE_STRING BaseDllName;//不带路径的驱动名字
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    union {
        struct {
            ULONG TimeDateStamp;
        };
        struct {
            PVOID LoadedImports;
        };
    };
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;



typedef NTSTATUS (* FuncPtr) (
    _Out_ PHANDLE FileHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_ POBJECT_ATTRIBUTES ObjectAttributes,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _In_opt_ PLARGE_INTEGER AllocationSize,
    _In_ ULONG FileAttributes,
    _In_ ULONG ShareAccess,
    _In_ ULONG CreateDisposition,
    _In_ ULONG CreateOptions,
    _In_reads_bytes_opt_(EaLength) PVOID EaBuffer,
    _In_ ULONG EaLength
    );
typedef NTSTATUS (* FuncPtr2 )(
    _In_ HANDLE FileHandle,
    _In_opt_ HANDLE Event,
    _In_opt_ PIO_APC_ROUTINE ApcRoutine,
    _In_opt_ PVOID ApcContext,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _Out_writes_bytes_(Length) PVOID Buffer,
```

```
    _In_ ULONG Length,
    _In_opt_ PLARGE_INTEGER ByteOffset,
    _In_opt_ PULONG Key
    );

ULONG  myCreateFile(_Out_ PHANDLE FileHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_ POBJECT_ATTRIBUTES ObjectAttributes,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _In_opt_ PLARGE_INTEGER AllocationSize,
    _In_ ULONG FileAttributes,
    _In_ ULONG ShareAccess,
    _In_ ULONG CreateDisposition,
    _In_ ULONG CreateOptions,
    _In_reads_bytes_opt_(EaLength) PVOID EaBuffer,
    _In_ ULONG EaLength) {

    Unhook();
    FuncPtr func = (FuncPtr)target;

    NTSTATUS s =
func(FileHandle,DesiredAccess,ObjectAttributes,IoStatusBlock,AllocationSize,FileAttributes,Share
Access,CreateDisposition,CreateOptions,EaBuffer,EaLength);
    if (!wcscmp(ObjectAttributes->ObjectName->Buffer, L"\\??\\C:\\card.txt")) {
        kprintf(("call NtCreateFile(%p,%p,%S,%p,%p,%p,%p,%p,%p,%p,%p)\n"),
FileHandle,DesiredAccess,ObjectAttributes->ObjectName-
>Buffer,IoStatusBlock,AllocationSize,FileAttributes,ShareAccess,CreateDisposition,CreateOptions,
EaBuffer,EaLength);
        DbgBreakPoint();
        FileHandler = *FileHandle;
    }

    //DbgBreakPoint();


    Hook();

    return s;
}

ULONG myReadFile(
    _In_ HANDLE FileHandle,
    _In_opt_ HANDLE Event,
    _In_opt_ PIO_APC_ROUTINE ApcRoutine,
    _In_opt_ PVOID ApcContext,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _Out_writes_bytes_(Length) PVOID Buffer,
    _In_ ULONG Length,
    _In_opt_ PLARGE_INTEGER ByteOffset,
    _In_opt_ PULONG Key) {

    Unhook2();
    FuncPtr2 func = (FuncPtr2)target2;

    if (FileHandler && FileHandler == FileHandle) {
        kprintf(("call NtReadFile(%p,%p,%p,%p,%p,%p,%p,%p,%p)\n"), FileHandle, Event,
ApcRoutine, ApcContext, IoStatusBlock, Buffer, Length, ByteOffset, Key);
        DbgBreakPoint();
```

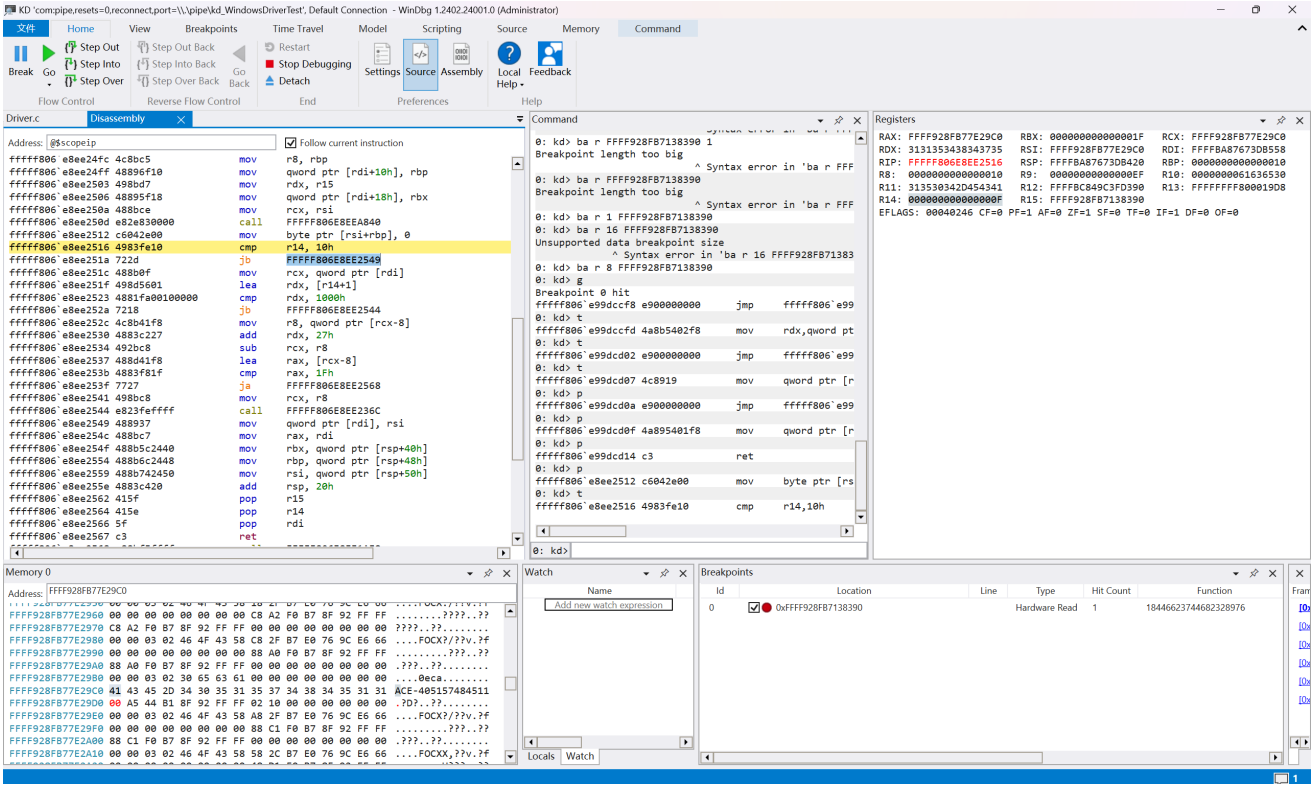
```
        FileHandler = 0;
    }
    //DbgBreakPoint();
    NTSTATUS s =
func(FileHandle,Event,ApcRoutine,ApcContext,IoStatusBlock,Buffer,Length,ByteOffset,Key);

    Hook2();

    return s;


}

void DriverUnload(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start unload\n"), __LINE__);
    Unhook();
    Unhook2();
    DeleteDevice(pDriver);
}


NTSTATUS DriverEntry(
    _In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING RegistryPath
) {
    DriverObject->DriverUnload = DriverUnload;
    kprintf(("Line %d:xia0ji233: RegistryPath = %S\n"), __LINE__, RegistryPath->Buffer);
    target = NtCreateFile;
    target2 = NtReadFile;
    kprintf(("Line %d:xia0ji233: NtCreateFile=%p NtReadFile=%p\n"), __LINE__, target,target2);
    g_Object = DriverObject;
    if (target&&target2) {
        for (int i = 0; i < sizeof(oldcode); i++) {
            oldcode[i] = target[i];
            oldcode2[i] = target2[i];
        }
        *(UINT64*)(newcode + 2) = myCreateFile;
        *(UINT64*)(newcode2 + 2) = myReadFile;
        Hook();
        Hook2();
    }
    else {
        kprintf(("xia0ji233:hahaha"));
    }
    return 0;
}
```

跟到后面可以找到文件内容（ReadFile第六个参数），在文件内容处下硬件读取断点，可以找到这里文件内容被被写入两个寄存器，随后又写入另外的内存（RCX所指向的内存）。

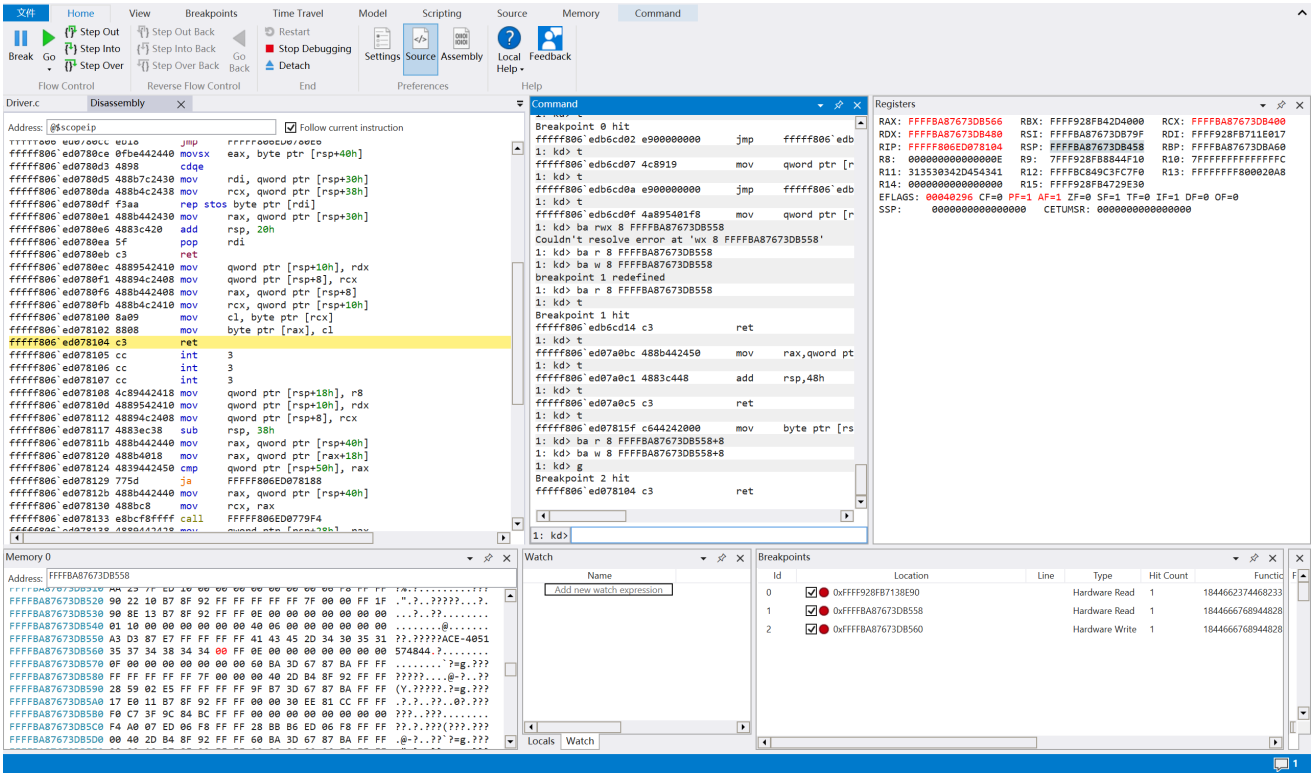


然后就直接 ret 了，这里将内存窗口转到 RCX 指向的地址，然后程序跳出来。

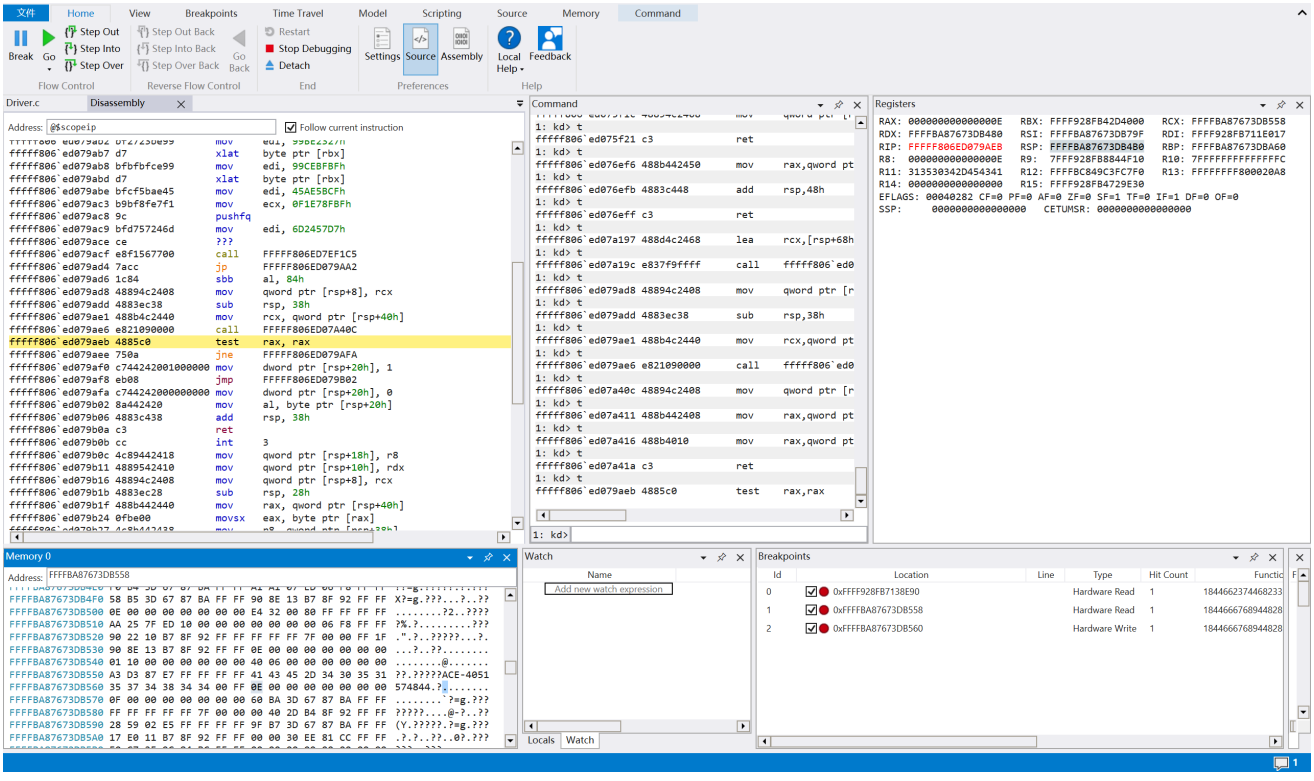
紧接着给末尾添 0 字节，然后将 r14 与 0x10 比较，小于跳出



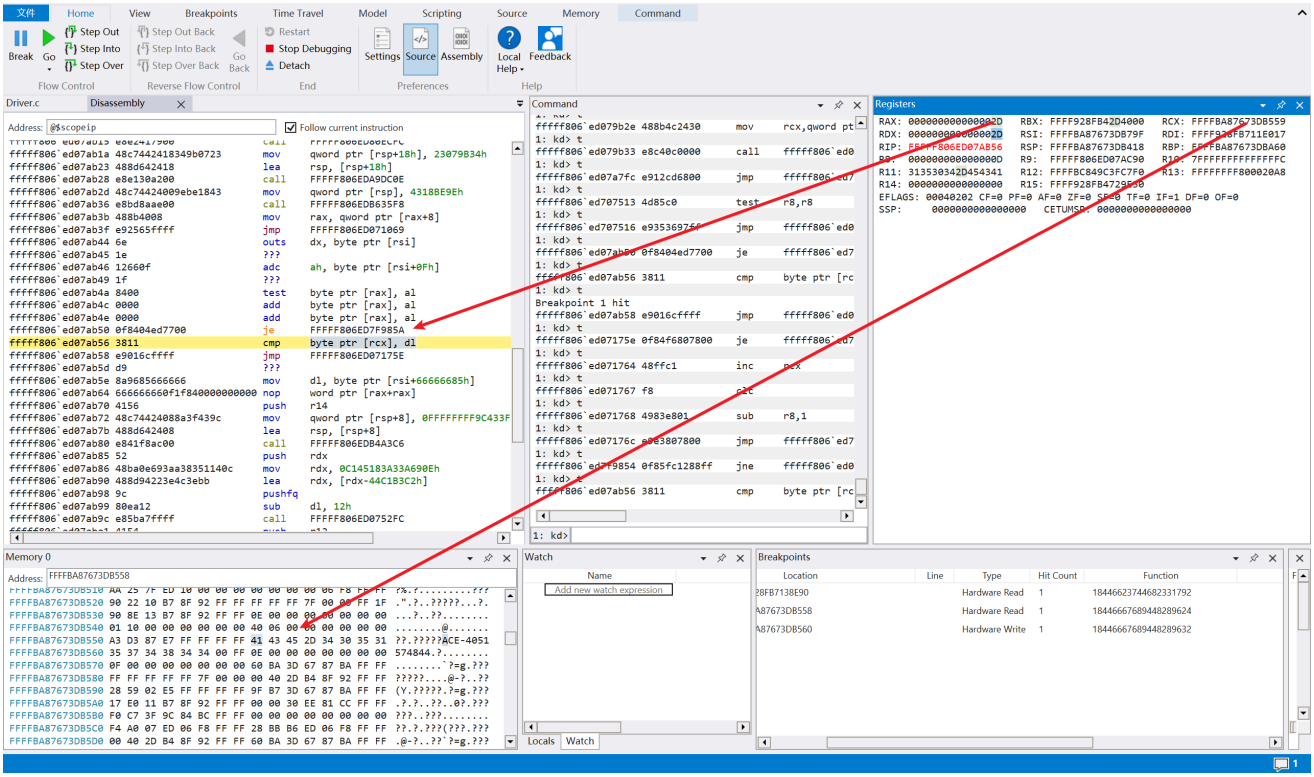
这一次输出失败了，于是考虑换长度但是改一个字符，发现在另外的位置写了零字节



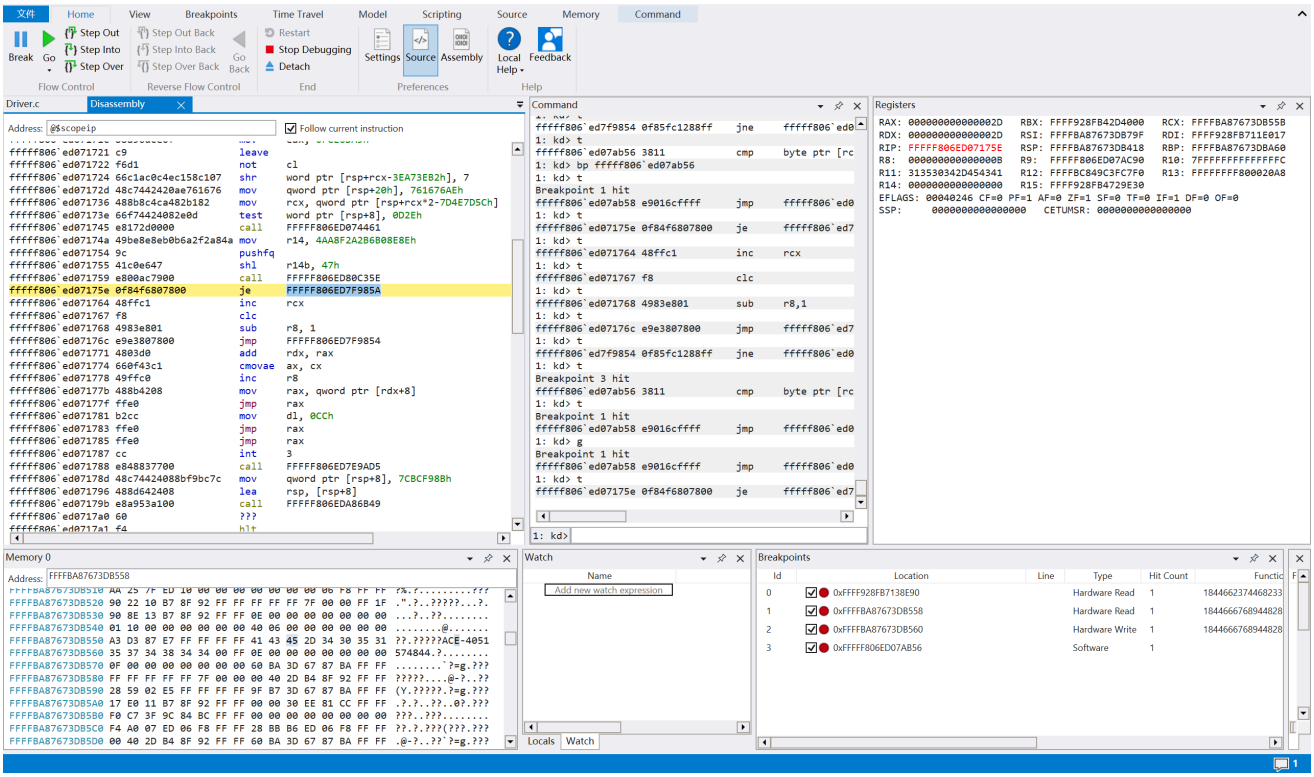
在跟的过程中，发现后面的一个内存就是长度



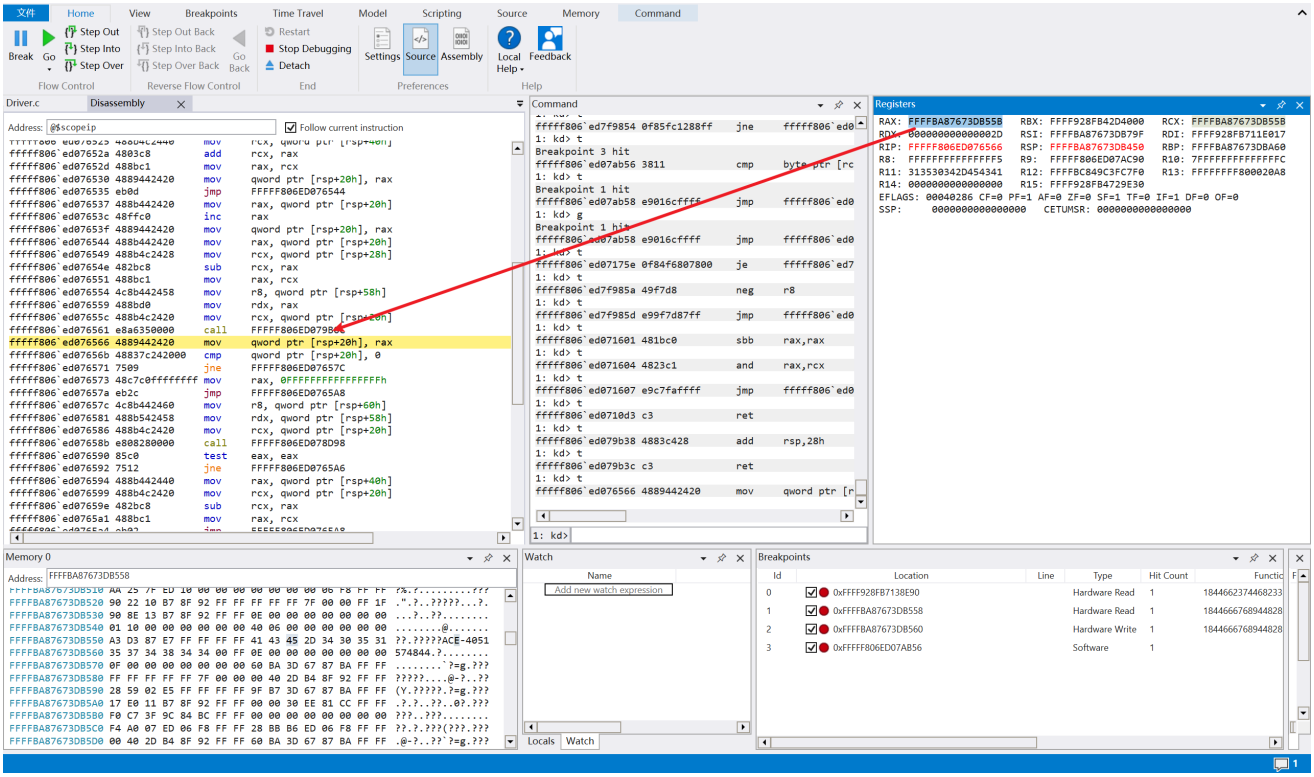
然后接着跟，会跟到一个找 - 的代码



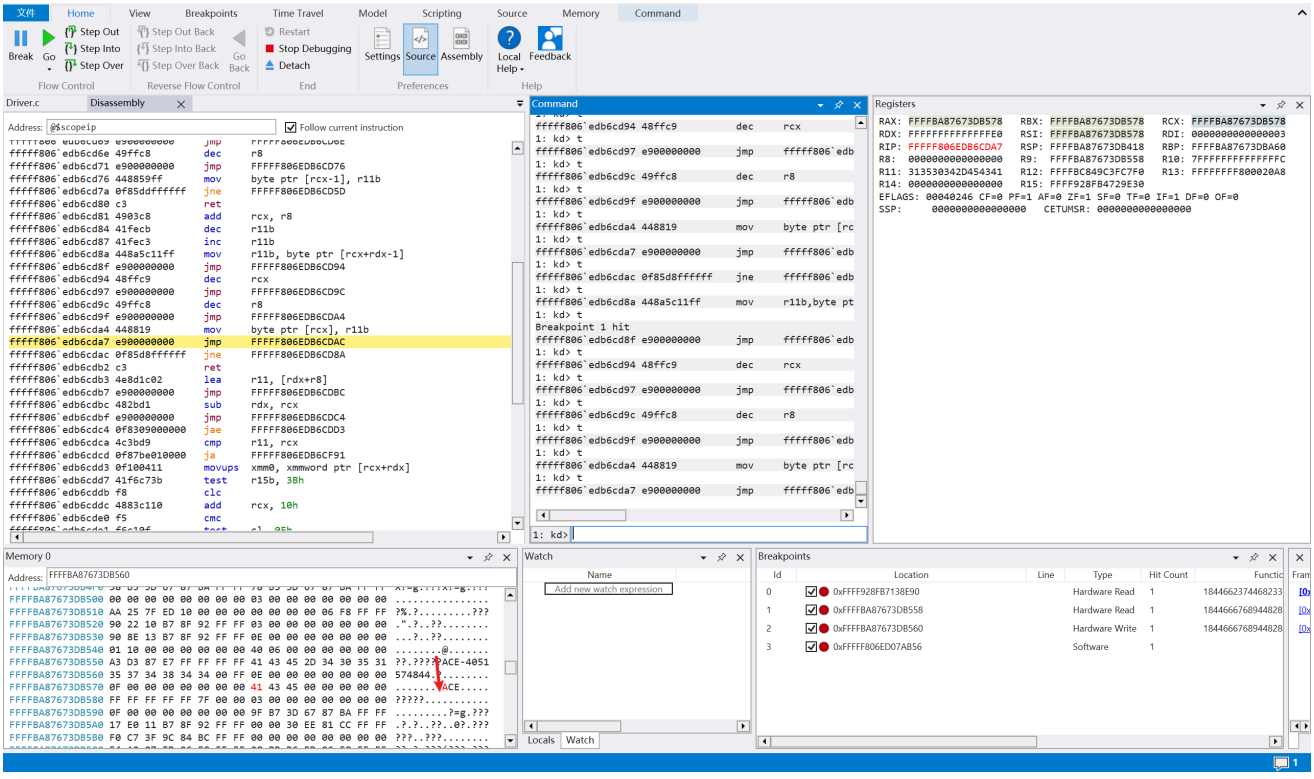
很容易理解，因为 - 就是分隔 user 和 key 的，必然有一个遍历在找 - 的位置，那就直接跳到它找到了 - 的位置，发现有一个大跳转



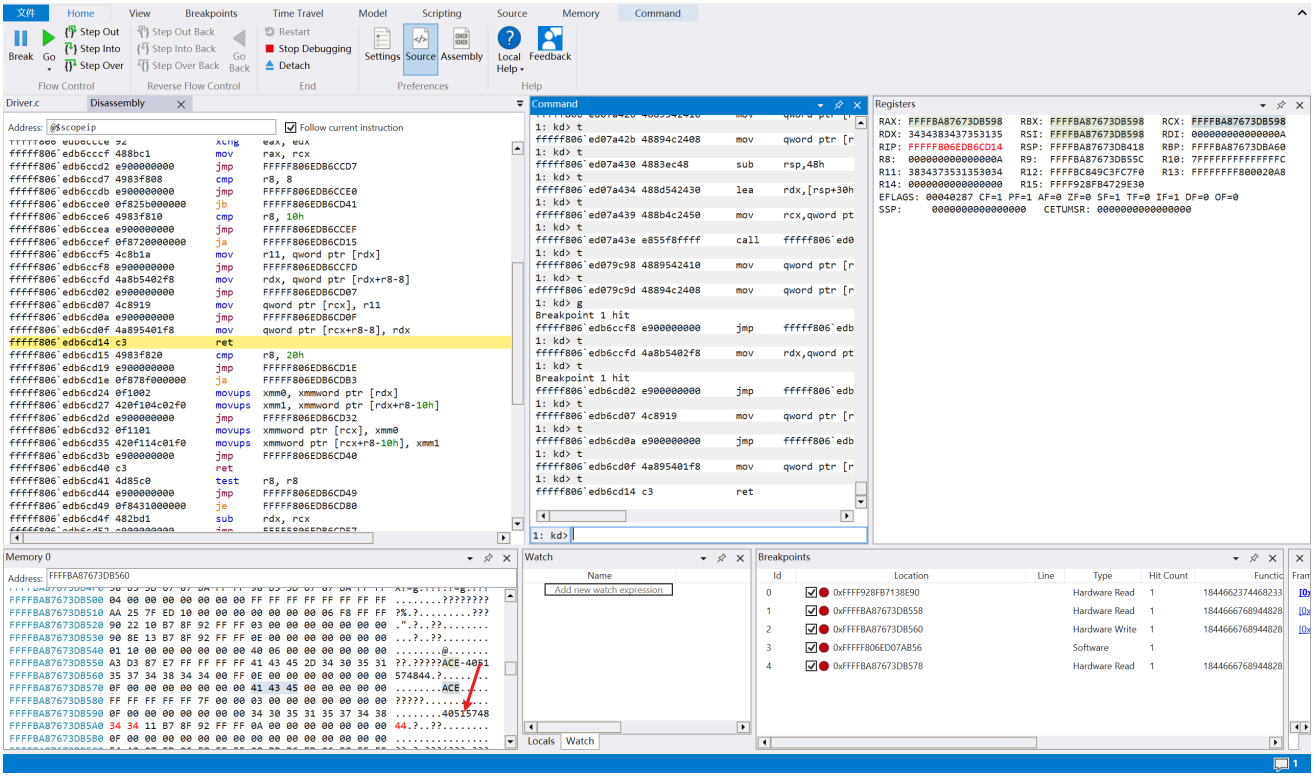
随后会把 - 所处的地址存入栈中



紧接着跟，发现把User拷贝到下面的内存了

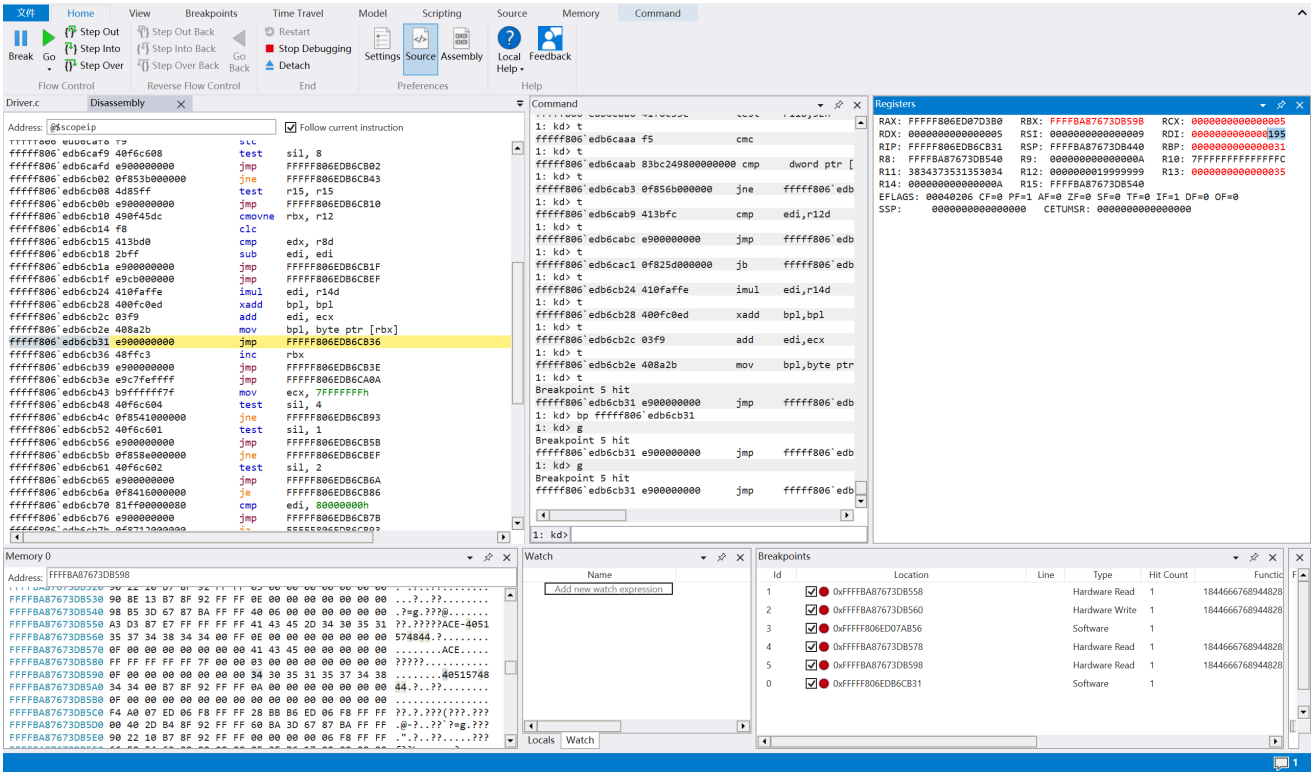


随后将key也写入下方的内存



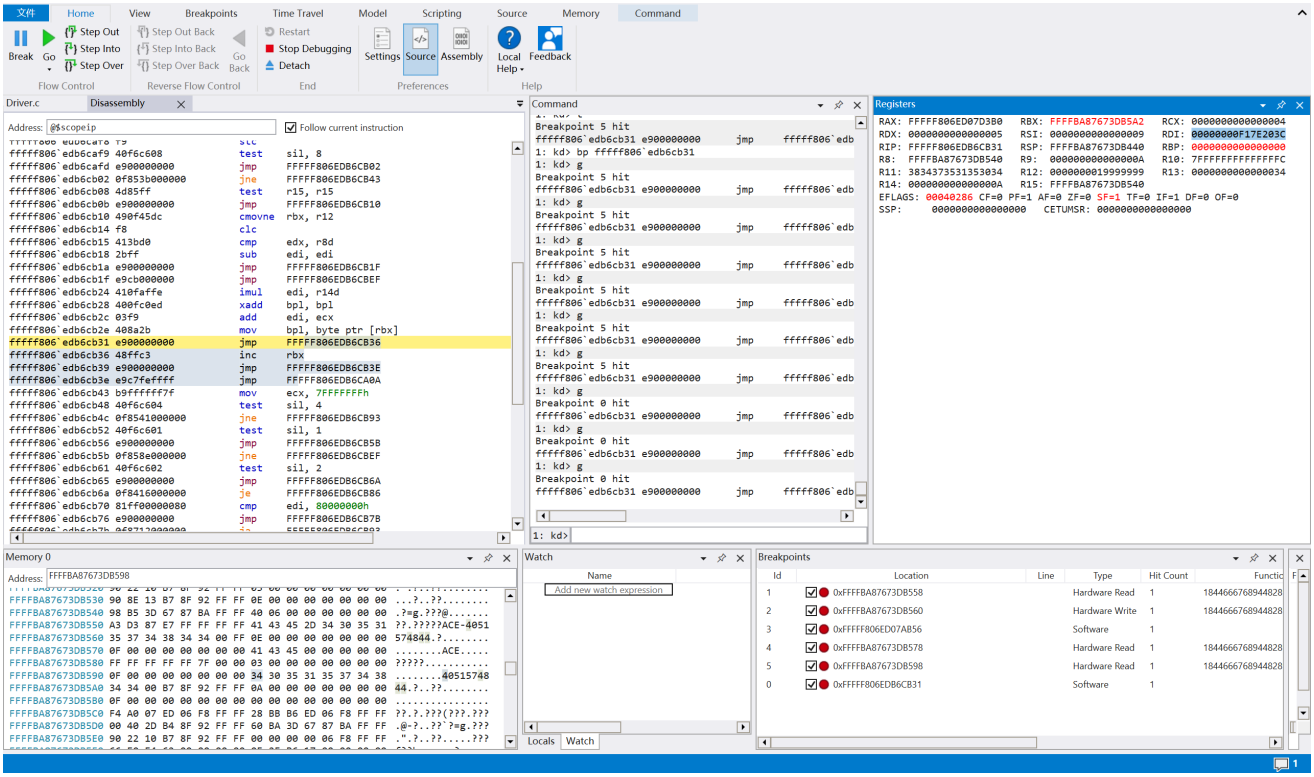
然后开始循环 key，判断如果 -0x30 是否 >10，应该 key 只能是数字的判断。

于是我在循环这里下了一个软件断点，发现 RDI 会存储当前已遍历的十进制。

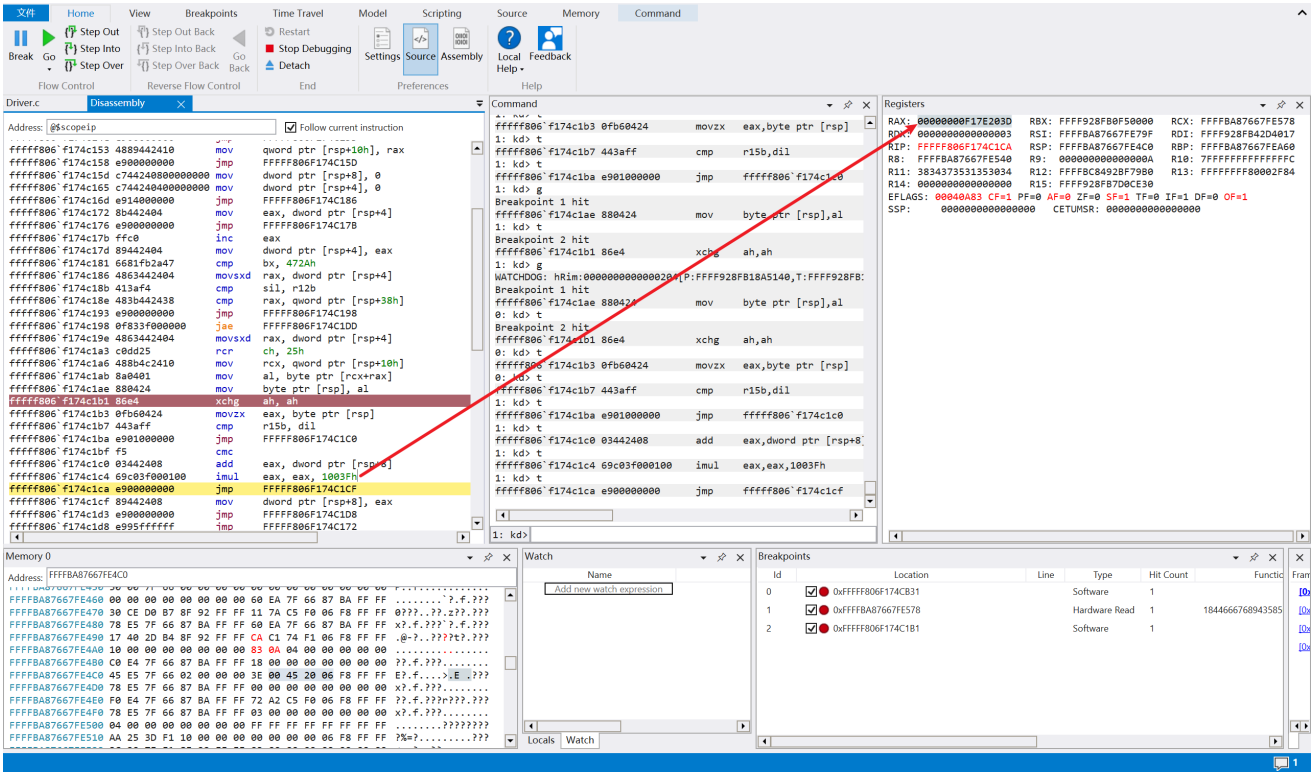


例如现在已经遍历到了 405，那么 rdi=0x195=405。

经过数次的循环，

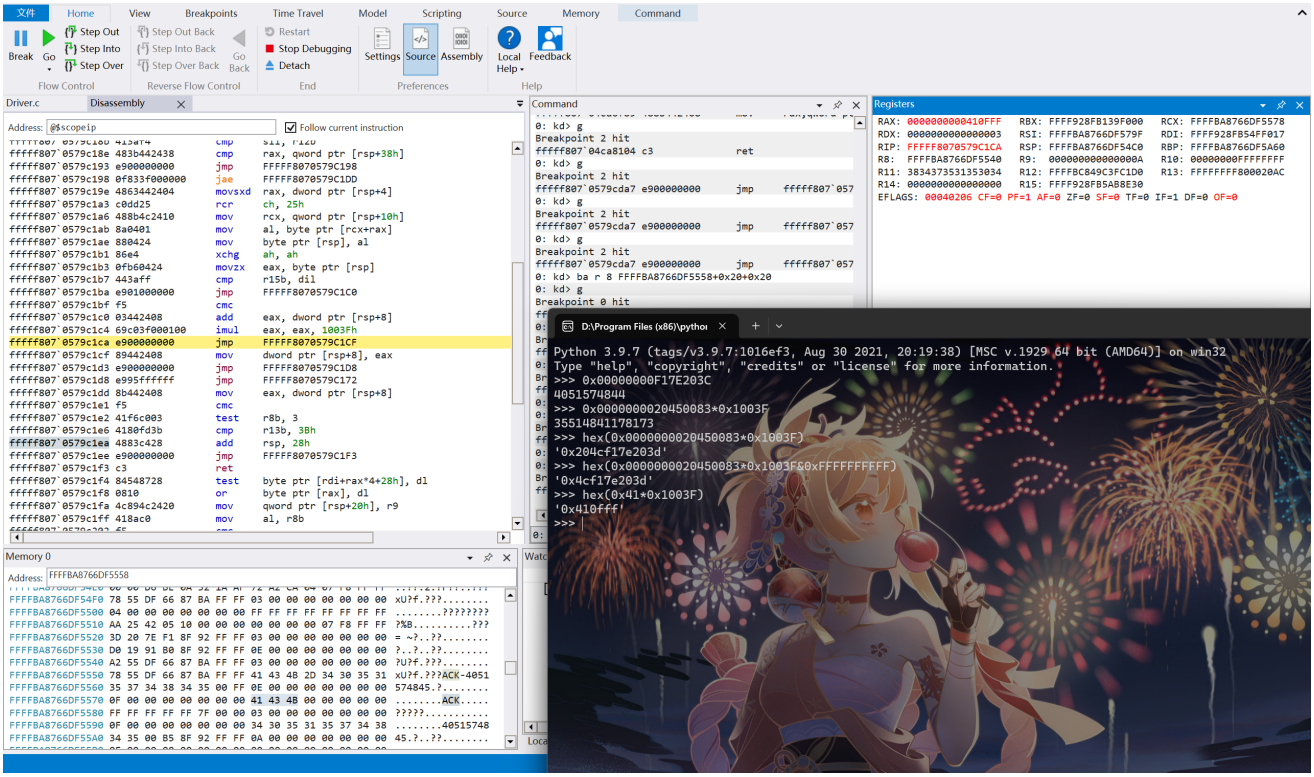


可以发现 RDI=0xF17E203C，就是 4051574844，那么接下来就往下面跟看看跟 User 有什么样的关系。



取出ACE之后算出一个值 0x0000000020450083，最后很 0x1003F 相乘，得到 00000000F17E203D，刚好是题目所给 key 的十六进制表示。

后面通过一定的调试分析，发现一个规律



似乎它会把每一个字符加起来然后 `*0x1003F`，并且一定是 `int`。

先验证一遍ACE是否正确

发现果然如此，那么照样子算出 key 为 `4007951923`。

编写一个keygen，能生成对于任何用户名的KEY（1分）

根据上面的分析，不难写出 keygen 程序

```c
#include<stdio.h>
#include<string.h>
int main(){
    char user[]="xia0ji233";
    int n=strlen(user);
    unsigned key=0;
    for(int i=0;i<n;i++){
        key=(key+user[i])*0x1003F;
    }
    printf("%u\n",key);
}
```

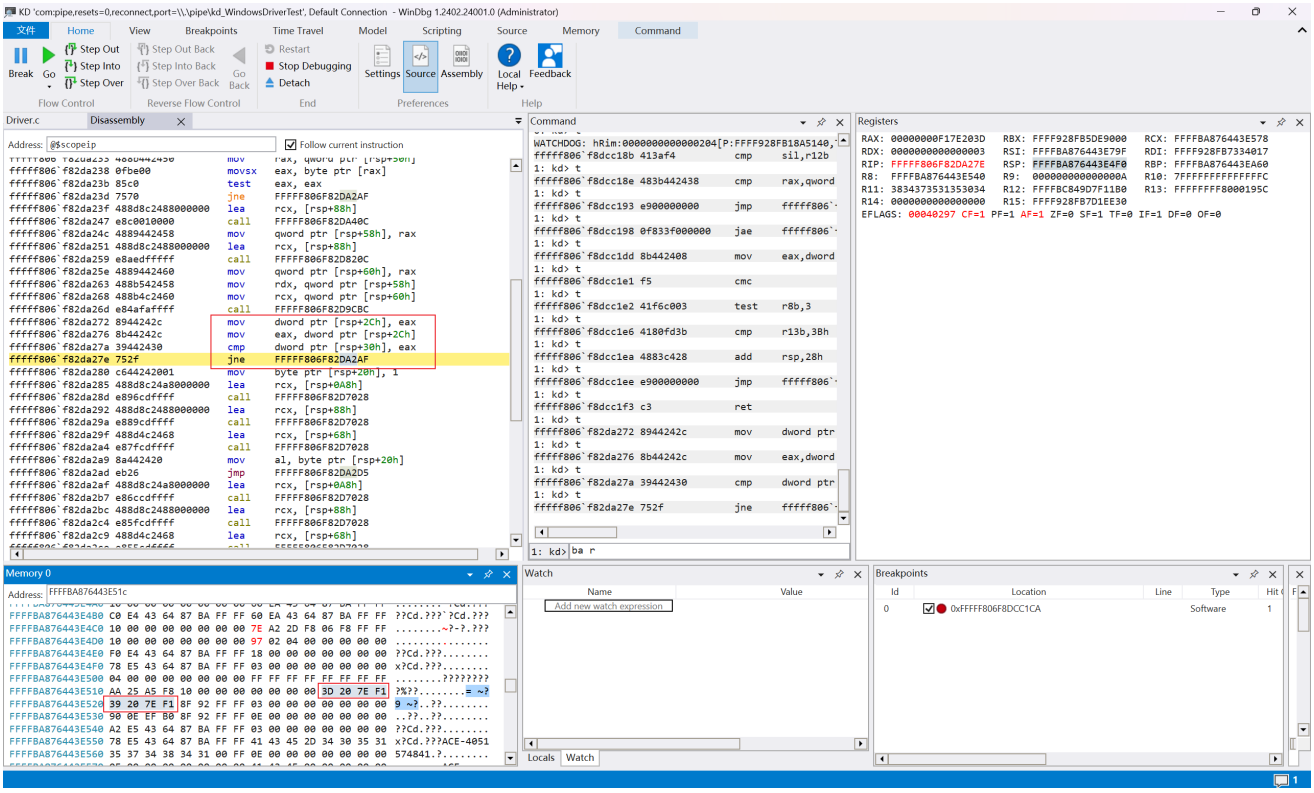
运行即可获得 key 的输出。

编写一个exp，在exp程序运行后，对于任意的用户名-key，Loader.sys均能正确启动（1分）

有这么几种方法：监控内核线程，在 shellcode 执行的时候拦截，把比较是否相等的代码patch掉。监控文件读写，在读文件的时候，判断如果是目标文件，让它返回正确的结果。

最后还是选择在文件处拦截，然后趁它读文件的时候遍历驱动模块改它代码，上面分析的 imul 关键指令在 `Loader.sys+0xafc1c4` 的位置

再调试一遍，决出关键一步，找到了 cmp 指令



如图所示的内存分别为实际输入的数值和通过 user 计算得到的key的数值，随后取出相比较，不相等显然跳转到 Fail 分支，因此这里改成 NOP 让它不跳转任意情况下跳转成功。

此时的 sys 基地址为 `0xFFFFF806F82D0000`，与该指令相减得到 `0xa27e` 的偏移，只需把这两个字节改成 0x90 即可达到任意的 user key 可以成功加载驱动。

我先使用了hook的方式去劫持，确保劫持的函数没错，再通过修改劫持时机和方式让方法满足要求

```c
#include <ntifs.h>
```

```
#include <ntdef.h>
#include <ntstatus.h>
#include <ntddk.h>
#define MAX_BACKTRACE_DEPTH 20
#define SYMBOL L"\\??\\xia0ji2333"
#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)

UINT64 BaseAddr=NULL, DLLSize=0;

void DeleteDevice(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start delete device\n"), __LINE__);
    if (pDriver->DeviceObject) {
        UNICODE_STRING Sym;
        RtlInitUnicodeString(&Sym, SYMBOL);//CreateFile
        kprintf(("Line %d:xia0ji233: Delete Symbol\n"), __LINE__);
        IoDeleteSymbolicLink(&Sym);
        kprintf(("Line %d:xia0ji233: Delete Device\n"), __LINE__);
        IoDeleteDevice(pDriver->DeviceObject);
    }
    kprintf(("Line %d:xia0ji233: end delete device\n"), __LINE__);
}

HANDLE FileHandler = NULL;

char newcode[] = {
    0x48,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//mov rax,xxx
    0xFF,0xE0//jmp rax
};
char oldcode[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,
};

char newcode2[] = {
    0x48,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//mov rax,xxx
    0xFF,0xE0//jmp rax
};
char oldcode2[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,
};

char* target;
char* target2;
KIRQL WPOFFx64()
{
    KIRQL irql = KeRaiseIrqlToDpcLevel();
    UINT64 cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}

void WPONx64(KIRQL irql)
{
    UINT64 cr0 = __readcr0();
```

```
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

NTSTATUS Unhook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = oldcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Unhook2() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode2); i++) {
        target2[i] = oldcode2[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Hook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = newcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

NTSTATUS Hook2() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode2); i++) {
        target2[i] = newcode2[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}

PDRIVER_OBJECT g_Object = NULL;
typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;//驱动的进入点 DriverEntry
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;//驱动的满路径
    UNICODE_STRING BaseDllName;//不带路径的驱动名字
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
```

```
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    union {
        struct {
            ULONG TimeDateStamp;
        };
        struct {
            PVOID LoadedImports;
        };
    };
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;



typedef NTSTATUS (* FuncPtr) (
    _Out_ PHANDLE FileHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_ POBJECT_ATTRIBUTES ObjectAttributes,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _In_opt_ PLARGE_INTEGER AllocationSize,
    _In_ ULONG FileAttributes,
    _In_ ULONG ShareAccess,
    _In_ ULONG CreateDisposition,
    _In_ ULONG CreateOptions,
    _In_reads_bytes_opt_(EaLength) PVOID EaBuffer,
    _In_ ULONG EaLength
    );
typedef NTSTATUS (* FuncPtr2 )(
    _In_ HANDLE FileHandle,
    _In_opt_ HANDLE Event,
    _In_opt_ PIO_APC_ROUTINE ApcRoutine,
    _In_opt_ PVOID ApcContext,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _Out_writes_bytes_(Length) PVOID Buffer,
    _In_ ULONG Length,
    _In_opt_ PLARGE_INTEGER ByteOffset,
    _In_opt_ PULONG Key
    );

NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(ULONG, PVOID, ULONG, PULONG);
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER Reserved[3];
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    ULONG BasePriority;
    HANDLE ProcessId;
    HANDLE InheritedFromProcessId;
} SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
typedef struct _SYSTEM_THREAD_INFORMATION {
    LARGE_INTEGER KernelTime;
```

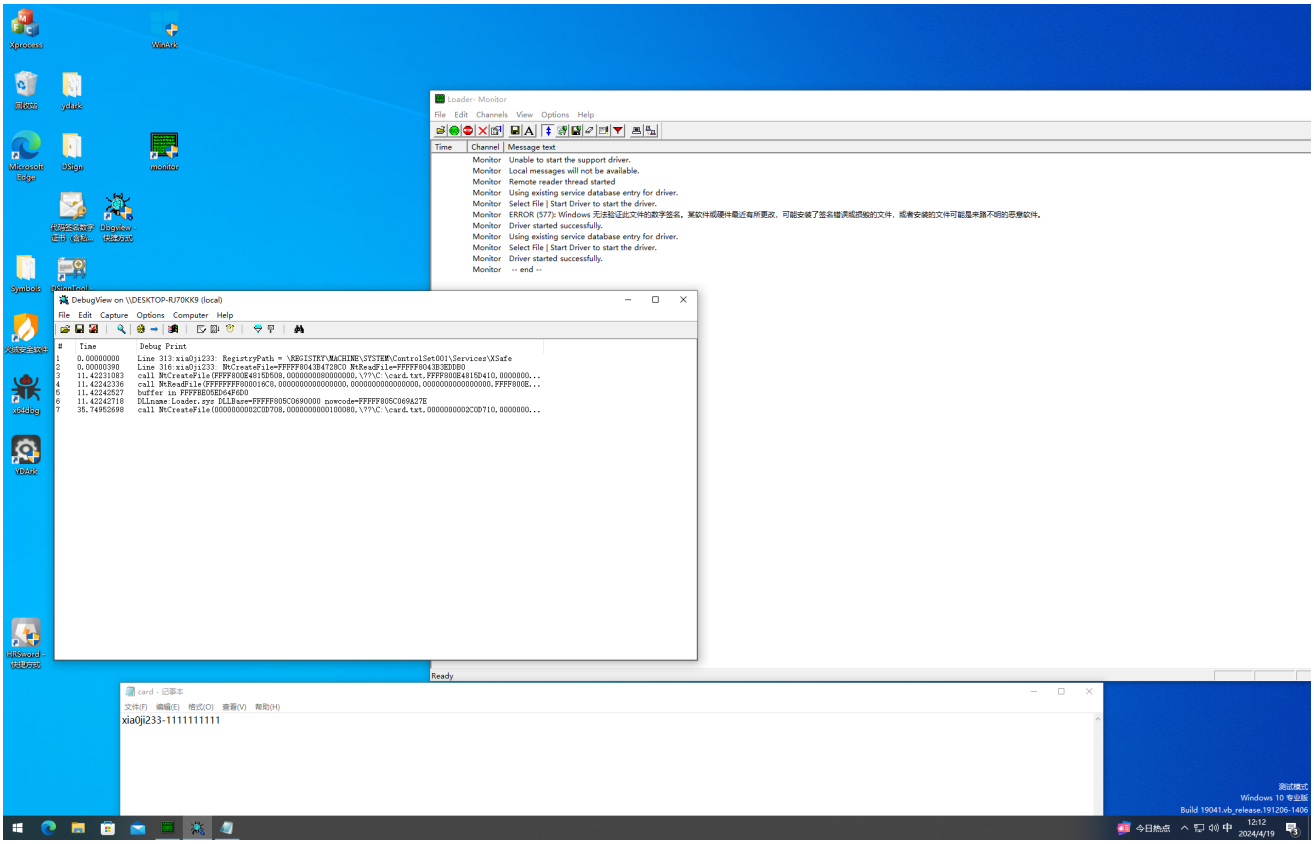
```
    LARGE_INTEGER UserTime;
    LARGE_INTEGER CreateTime;
    ULONG WaitTime;
    PVOID StartAddress;
    CLIENT_ID ClientId;
    ULONG Priority;
    LONG BasePriority;
    ULONG ContextSwitchCount;
    LONG State;
    LONG WaitReason;
} SYSTEM_THREAD_INFORMATION, *PSYSTEM_THREAD_INFORMATION;
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemProcessInformation = 5
} SYSTEM_INFORMATION_CLASS;
#define SystemModuleInformation 11

ULONG  myCreateFile(_Out_ PHANDLE FileHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_ POBJECT_ATTRIBUTES ObjectAttributes,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _In_opt_ PLARGE_INTEGER AllocationSize,
    _In_ ULONG FileAttributes,
    _In_ ULONG ShareAccess,
    _In_ ULONG CreateDisposition,
    _In_ ULONG CreateOptions,
    _In_reads_bytes_opt_(EaLength) PVOID EaBuffer,
    _In_ ULONG EaLength) {

    Unhook();
    FuncPtr func = (FuncPtr)target;

    NTSTATUS s =
func(FileHandle,DesiredAccess,ObjectAttributes,IoStatusBlock,AllocationSize,FileAttributes,Share
Access,CreateDisposition,CreateOptions,EaBuffer,EaLength);
    if (!wcscmp(ObjectAttributes->ObjectName->Buffer, L"\\??\\C:\\card.txt")) {
        kprintf(("call NtCreateFile(%p,%p,%S,%p,%p,%p,%p,%p,%p,%p,%p)\n"),
FileHandle,DesiredAccess,ObjectAttributes->ObjectName-
>Buffer,IoStatusBlock,AllocationSize,FileAttributes,ShareAccess,CreateDisposition,CreateOptions,
EaBuffer,EaLength);
        //DbgBreakPoint();
        FileHandler = *FileHandle;
    }

    //DbgBreakPoint();


    Hook();

    return s;
}

MDLWriteMemory(PVOID pBaseAddress, PVOID pWriteData, SIZE_T writeDataSize)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    pMdl = MmCreateMdl(NULL, pBaseAddress, writeDataSize);
    if (NULL == pMdl)
    {
```

```
        return FALSE;
    }
    MmBuildMdlForNonPagedPool(pMdl);
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
    }
    RtlCopyMemory(pNewAddress, pWriteData, writeDataSize);
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);
    return TRUE;
}


VOID PatchInstr()
{
    LDR_DATA_TABLE_ENTRY *TE, *Tmp;
    TE = (LDR_DATA_TABLE_ENTRY*)g_Object->DriverSection;
    PLIST_ENTRY LinkList;
    ;
    int i = 0;
    LinkList = TE->InLoadOrderLinks.Flink;
    UNICODE_STRING name;
    RtlInitUnicodeString(&name,L"Loader.sys");
    ;
    while (LinkList != &TE->InLoadOrderLinks)
    {
        Tmp = (LDR_DATA_TABLE_ENTRY*)LinkList;

        if (RtlEqualUnicodeString(&Tmp->BaseDllName, &name,FALSE)) {
            kprintf(("DLLname:%S DLLBase=%p nowcode=%p\n"), Tmp->BaseDllName.Buffer,Tmp-
>DllBase,(ULONG64)(Tmp->DllBase) + 0xa27e);
            char buffer[] = { 0x90,0x90 };
            MDLWriteMemory((ULONG64)(Tmp->DllBase) + 0xa27e, buffer, 2);
            return;
        }
        LinkList = LinkList->Flink;
        i++;
    }


}

ULONG myReadFile(
    _In_ HANDLE FileHandle,
    _In_opt_ HANDLE Event,
    _In_opt_ PIO_APC_ROUTINE ApcRoutine,
    _In_opt_ PVOID ApcContext,
    _Out_ PIO_STATUS_BLOCK IoStatusBlock,
    _Out_writes_bytes_(Length) PVOID Buffer,
    _In_ ULONG Length,
    _In_opt_ PLARGE_INTEGER ByteOffset,
    _In_opt_ PULONG Key) {

    Unhook2();
    FuncPtr2 func = (FuncPtr2)target2;
```

```
    if (FileHandler && FileHandler == FileHandle) {
        kprintf(("call NtReadFile(%p,%p,%p,%p,%p,%p,%p,%p,%p)\n"), FileHandle, Event,
ApcRoutine, ApcContext, IoStatusBlock, Buffer, Length, ByteOffset, Key);
        kprintf(("buffer in %p\n"), Buffer);
        PatchInstr();
        FileHandler = 0;
    }
    //DbgBreakPoint();
    NTSTATUS s =
func(FileHandle,Event,ApcRoutine,ApcContext,IoStatusBlock,Buffer,Length,ByteOffset,Key);

    Hook2();

    return s;


}

void DriverUnload(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start unload\n"), __LINE__);
    Unhook();
    Unhook2();
    DeleteDevice(pDriver);
}


NTSTATUS DriverEntry(
    _In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING RegistryPath
) {
    DriverObject->DriverUnload = DriverUnload;
    kprintf(("Line %d:xia0ji233: RegistryPath = %S\n"), __LINE__, RegistryPath->Buffer);
    target = NtCreateFile;
    target2 = NtReadFile;
    kprintf(("Line %d:xia0ji233: NtCreateFile=%p NtReadFile=%p\n"), __LINE__, target,target2);
    g_Object = DriverObject;
    if (target&&target2) {
        for (int i = 0; i < sizeof(oldcode); i++) {
            oldcode[i] = target[i];
            oldcode2[i] = target2[i];
        }
        *(UINT64*)(newcode + 2) = myCreateFile;
        *(UINT64*)(newcode2 + 2) = myReadFile;
        Hook();
        Hook2();
    }
    else {
        kprintf(("xia0ji233:hahaha"));
    }
    return 0;
}

int Filter(ULONG Start)
{
    LDR_DATA_TABLE_ENTRY *TE, *Tmp;
    TE = (LDR_DATA_TABLE_ENTRY*)g_Object->DriverSection;
    PLIST_ENTRY LinkList;
    ;
    int i = 0;
```

```
    LinkList = TE->InLoadOrderLinks.Flink;
    while (LinkList != &TE->InLoadOrderLinks)
    {
        Tmp = (LDR_DATA_TABLE_ENTRY*)LinkList;
        ULONG BASE = Tmp->DllBase;
        ULONG Size = Tmp->SizeOfImage;
        if (Start >= BASE && Start < BASE + Size) {
            return 0;
        }

        LinkList = LinkList->Flink;
        i++;
    }

    return 1;
}
```

可以看到，先加载我写的驱动，后加载题目驱动，无论 user-key 正确与否，都加载成功



于是这里把修改的函数套到文件读取里面去拦截，达到读该文件时遍历模块，找到指定模块则写入指令。

下面是真正的代码（编译文件为XSafe2.sys）

```
#include <ntifs.h>
#include <ntdef.h>
#include <ntstatus.h>
#include <ntddk.h>
#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)

typedef struct _CALLBACK_ENTRY
{
    LIST_ENTRY CallbackList;
    OB_OPERATION  Operations;
```

```
    ULONG Active;
    PVOID Handle;
    POBJECT_TYPE ObjectType;
    POB_PRE_OPERATION_CALLBACK  PreOperation;
    POB_POST_OPERATION_CALLBACK PostOperation;
    ULONG unknown;
} CALLBACK_ENTRY, *PCALLBACK_ENTRY;

typedef struct _LDR_DATA                        // 24 elements, 0xE0 bytes (sizeof)
{
    /*0x000*/     struct _LIST_ENTRY InLoadOrderLinks;                    // 2 elements, 0x10
bytes (sizeof)
    /*0x010*/     struct _LIST_ENTRY InMemoryOrderLinks;                  // 2 elements, 0x10
bytes (sizeof)
    /*0x020*/     struct _LIST_ENTRY InInitializationOrderLinks;          // 2 elements, 0x10
bytes (sizeof)
    /*0x030*/     VOID*        DllBase;
    /*0x038*/     VOID*        EntryPoint;
    /*0x040*/     ULONG32      SizeOfImage;
    /*0x044*/     UINT8        _PADDING0_[0x4];
    /*0x048*/     struct _UNICODE_STRING FullDllName;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x058*/     struct _UNICODE_STRING BaseDllName;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x068*/     ULONG32      Flags;
    /*0x06C*/     UINT16       LoadCount;
    /*0x06E*/     UINT16       TlsIndex;
    union                                                   // 2 elements, 0x10 bytes (sizeof)
    {
        /*0x070*/         struct _LIST_ENTRY HashLinks;                       // 2 elements,
0x10 bytes (sizeof)
        struct                                              // 2 elements, 0x10 bytes (sizeof)
        {
            /*0x070*/             VOID*        SectionPointer;
            /*0x078*/             ULONG32      CheckSum;
            /*0x07C*/             UINT8        _PADDING1_[0x4];
        };
    };
    union                                                   // 2 elements, 0x8 bytes (sizeof)
    {
        /*0x080*/         ULONG32      TimeDateStamp;
        /*0x080*/         VOID*        LoadedImports;
    };
    /*0x088*/     struct _ACTIVATION_CONTEXT* EntryPointActivationContext;
    /*0x090*/     VOID*        PatchInformation;
    /*0x098*/     struct _LIST_ENTRY ForwarderLinks;                      // 2 elements, 0x10
bytes (sizeof)
    /*0x0A8*/     struct _LIST_ENTRY ServiceTagLinks;                     // 2 elements, 0x10
bytes (sizeof)
    /*0x0B8*/     struct _LIST_ENTRY StaticLinks;                         // 2 elements, 0x10
bytes (sizeof)
    /*0x0C8*/     VOID*        ContextInformation;
    /*0x0D0*/     UINT64       OriginalBase;
    /*0x0D8*/     union _LARGE_INTEGER LoadTime;                          // 4 elements, 0x8
bytes (sizeof)
}LDR_DATA, *PLDR_DATA;
```

```
typedef struct _OBJECT_TYPE_INITIALIZER
                                                                                // 25 elements,
0x70 bytes (sizeof)
{
    /*0x000*/     UINT16       Length;
    union
                                                                                // 2 elements,
0x1 bytes (sizeof)
    {
        /*0x002*/         UINT8        ObjectTypeFlags;
        struct
                                                                              // 7 elements,
0x1 bytes (sizeof)
        {
            /*0x002*/             UINT8        CaseInsensitive : 1;

      // 0 BitPosition
            /*0x002*/             UINT8        UnnamedObjectsOnly : 1;

     // 1 BitPosition
            /*0x002*/             UINT8        UseDefaultObject : 1;

     // 2 BitPosition
            /*0x002*/             UINT8        SecurityRequired : 1;

     // 3 BitPosition
            /*0x002*/             UINT8        MaintainHandleCount : 1;

      // 4 BitPosition
            /*0x002*/             UINT8        MaintainTypeList : 1;

     // 5 BitPosition
            /*0x002*/             UINT8        SupportsObjectCallbacks : 1;

      // 6 BitPosition
        };
    };
    /*0x004*/     ULONG32      ObjectTypeCode;
    /*0x008*/     ULONG32      InvalidAttributes;
    /*0x00C*/     struct _GENERIC_MAPPING GenericMapping;
                                                                                //
4 elements, 0x10 bytes (sizeof)
    /*0x01C*/     ULONG32      ValidAccessMask;
    /*0x020*/     ULONG32      RetainAccess;
    /*0x024*/     enum _POOL_TYPE PoolType;
    /*0x028*/     ULONG32      DefaultPagedPoolCharge;
    /*0x02C*/     ULONG32      DefaultNonPagedPoolCharge;
    /*0x030*/     PVOID DumpProcedure;
    /*0x038*/     PVOID OpenProcedure;
    /*0x040*/     PVOID CloseProcedure;
    /*0x048*/     PVOID DeleteProcedure;
    /*0x050*/     PVOID ParseProcedure;
    /*0x058*/     PVOID SecurityProcedure;
    /*0x060*/     PVOID QueryNameProcedure;
    /*0x068*/     PVOID OkayToCloseProcedure;
}OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

typedef struct _EX_PUSH_LOCK                // 7 elements, 0x8 bytes (sizeof)
```

```
{
    union                                   // 3 elements, 0x8 bytes (sizeof)
    {
        struct                              // 5 elements, 0x8 bytes (sizeof)
        {
            /*0x000*/             UINT64       Locked : 1;         // 0 BitPosition
            /*0x000*/             UINT64       Waiting : 1;        // 1 BitPosition
            /*0x000*/             UINT64       Waking : 1;         // 2 BitPosition
            /*0x000*/             UINT64       MultipleShared : 1; // 3 BitPosition
            /*0x000*/             UINT64       Shared : 60;        // 4 BitPosition
        };
        /*0x000*/         UINT64       Value;
        /*0x000*/         VOID*        Ptr;
    };
};

typedef struct _MY_OBJECT_TYPE                  // 12 elements, 0xD0 bytes (sizeof)
{
    /*0x000*/     struct _LIST_ENTRY TypeList;              // 2 elements, 0x10 bytes (sizeof)
    /*0x010*/     struct _UNICODE_STRING Name;              // 3 elements, 0x10 bytes (sizeof)
    /*0x020*/     VOID*        DefaultObject;
    /*0x028*/     UINT8        Index;
    /*0x029*/     UINT8        _PADDING0_[0x3];
    /*0x02C*/     ULONG32      TotalNumberOfObjects;
    /*0x030*/     ULONG32      TotalNumberOfHandles;
    /*0x034*/     ULONG32      HighWaterNumberOfObjects;
    /*0x038*/     ULONG32      HighWaterNumberOfHandles;
    /*0x03C*/     UINT8        _PADDING1_[0x4];
    /*0x040*/     struct _OBJECT_TYPE_INITIALIZER TypeInfo; // 25 elements, 0x70 bytes (sizeof)
    /*0x0B0*/     struct _EX_PUSH_LOCK TypeLock;            // 7 elements, 0x8 bytes (sizeof)
    /*0x0B8*/     ULONG32      Key;
    /*0x0BC*/     UINT8        _PADDING2_[0x4];
    /*0x0C0*/     struct _LIST_ENTRY CallbackList;          // 2 elements, 0x10 bytes (sizeof)
}MY_OBJECT_TYPE, *PMY_OBJECT_TYPE;


NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath);
VOID UnloadDriver(PDRIVER_OBJECT DriverObject);
NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(ULONG, PVOID, ULONG, PULONG);
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER Reserved[3];
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    ULONG BasePriority;
    HANDLE ProcessId;
    HANDLE InheritedFromProcessId;
} SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
typedef struct _SYSTEM_THREAD_INFORMATION {
    LARGE_INTEGER KernelTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER CreateTime;
    ULONG WaitTime;
```

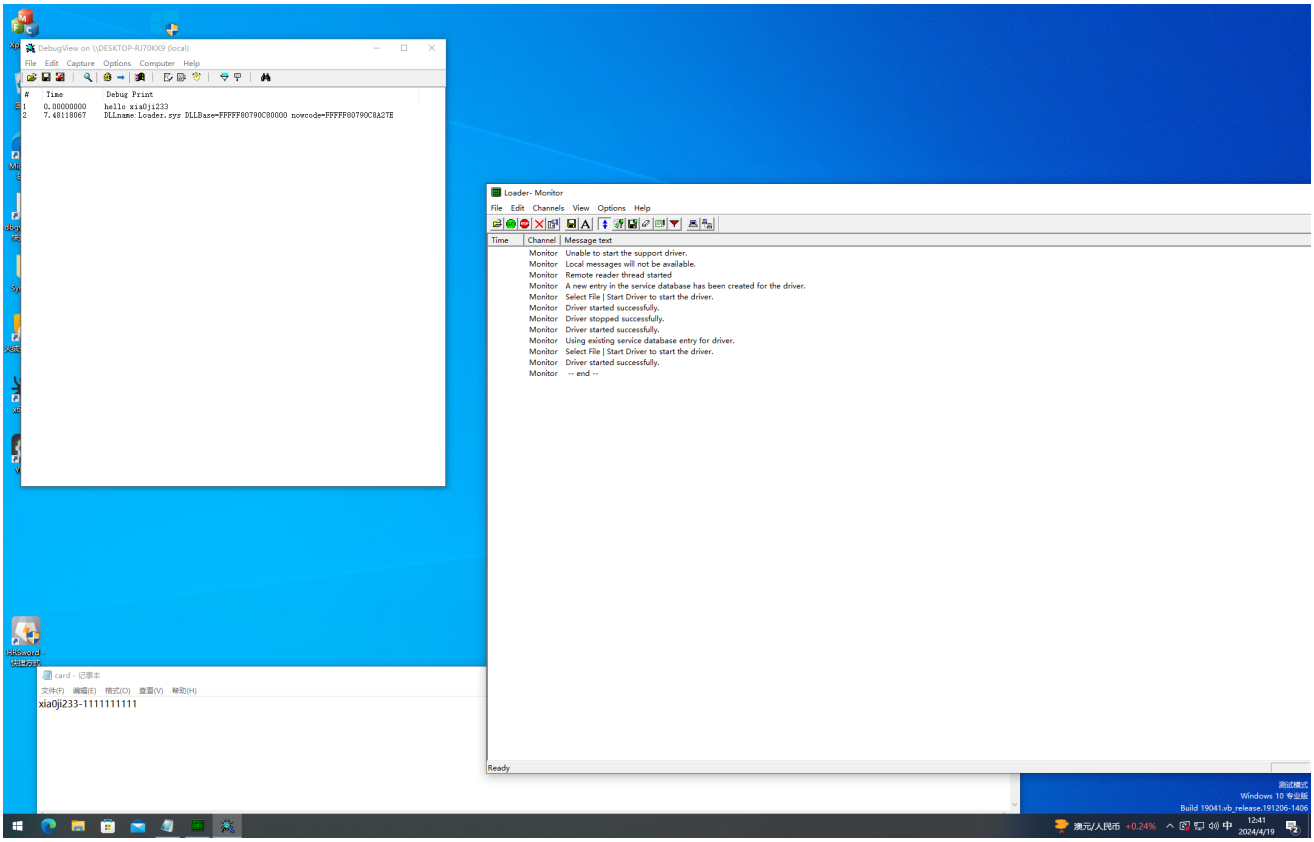
```
    PVOID StartAddress;
    CLIENT_ID ClientId;
    ULONG Priority;
    LONG BasePriority;
    ULONG ContextSwitchCount;
    LONG State;
    LONG WaitReason;
} SYSTEM_THREAD_INFORMATION, *PSYSTEM_THREAD_INFORMATION;
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemProcessInformation = 5
} SYSTEM_INFORMATION_CLASS;
#define SystemModuleInformation 11



PVOID obHandle;
DRIVER_INITIALIZE DriverEntry;

PDRIVER_OBJECT g_Object = NULL;
typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;//驱动的进入点 DriverEntry
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;//驱动的满路径
    UNICODE_STRING BaseDllName;//不带路径的驱动名字
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    union {
        struct {
            ULONG TimeDateStamp;
        };
        struct {
            PVOID LoadedImports;
        };
    };
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;


MDLWriteMemory(PVOID pBaseAddress, PVOID pWriteData, SIZE_T writeDataSize)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    pMdl = MmCreateMdl(NULL, pBaseAddress, writeDataSize);
    if (NULL == pMdl)
    {
        return FALSE;
    }
```

```
    MmBuildMdlForNonPagedPool(pMdl);
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
    }
    RtlCopyMemory(pNewAddress, pWriteData, writeDataSize);
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);
    return TRUE;
}


VOID PatchInstr()
{
    LDR_DATA_TABLE_ENTRY *TE, *Tmp;
    TE = (LDR_DATA_TABLE_ENTRY*)g_Object->DriverSection;
    PLIST_ENTRY LinkList;
    ;
    int i = 0;
    LinkList = TE->InLoadOrderLinks.Flink;
    UNICODE_STRING name;
    RtlInitUnicodeString(&name,L"Loader.sys");
    ;
    while (LinkList != &TE->InLoadOrderLinks)
    {
        Tmp = (LDR_DATA_TABLE_ENTRY*)LinkList;
        if (RtlEqualUnicodeString(&Tmp->BaseDllName, &name,FALSE)) {
            kprintf(("DLLname:%S DLLBase=%p nowcode=%p\n"), Tmp->BaseDllName.Buffer,Tmp-
>DllBase,(ULONG64)(Tmp->DllBase) + 0xa27e);
            char buffer[] = { 0x90,0x90 };
            MDLWriteMemory((ULONG64)(Tmp->DllBase) + 0xa27e, buffer, 2);
            return;
        }
        LinkList = LinkList->Flink;
        i++;
    }
}

// 文件回调
OB_PREOP_CALLBACK_STATUS FileObjectpreCall(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
{
    UNICODE_STRING DosName;
    PFILE_OBJECT fileo = OperationInformation->Object;
    HANDLE CurrentProcessId = PsGetCurrentProcessId();
    UNREFERENCED_PARAMETER(RegistrationContext);

    if (OperationInformation->ObjectType != *IoFileObjectType)
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效指针
    if (fileo->FileName.Buffer == NULL ||
        !MmIsAddressValid(fileo->FileName.Buffer) ||
        fileo->DeviceObject == NULL ||
        !MmIsAddressValid(fileo->DeviceObject))
```

```
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效路径
    if (!_wcsicmp(fileo->FileName.Buffer, L"\\Endpoint") ||
        !_wcsicmp(fileo->FileName.Buffer, L"?") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\.\\.") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\"))
    {
        return OB_PREOP_SUCCESS;
    }

    // 将对象转为DOS路径
    RtlVolumeDeviceToDosName(fileo->DeviceObject, &DosName);



    if (!wcscmp(fileo->FileName.Buffer, L"\\card.txt")) {
        PETHREAD pct=PsGetCurrentThread();
        PVOID addr=*(ULONG64*)((char*)pct + 0x450);
        PatchInstr();

        //EnumerateKernelThreads();
        DbgBreakPoint();
    }



    return OB_PREOP_SUCCESS;
}

VOID EnableObType(POBJECT_TYPE ObjectType)
{
    PMY_OBJECT_TYPE myobtype = (PMY_OBJECT_TYPE)ObjectType;
    myobtype->TypeInfo.SupportsObjectCallbacks = 1;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    UNREFERENCED_PARAMETER(driver);
    ObUnRegisterCallbacks(obHandle);
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status = STATUS_SUCCESS;
    PLDR_DATA ldr;

    kprintf(("hello xia0ji233\n"));
    g_Object = Driver;
    OB_CALLBACK_REGISTRATION obRegFileCallBack;
    OB_OPERATION_REGISTRATION opRegFileCallBack;

    // enable IoFileObjectType
    EnableObType(*IoFileObjectType);

    // bypass MmVerifyCallbackFunction
```

```
    ldr = (PLDR_DATA)Driver->DriverSection;
    ldr->Flags |= 0x20;

    // 初始化回调
    memset(&obRegFileCallBack, 0, sizeof(obRegFileCallBack));
    obRegFileCallBack.Version = ObGetFilterVersion();
    obRegFileCallBack.OperationRegistrationCount = 1;
    obRegFileCallBack.RegistrationContext = NULL;
    RtlInitUnicodeString(&obRegFileCallBack.Altitude, L"321000");
    obRegFileCallBack.OperationRegistration = &opRegFileCallBack;

    memset(&opRegFileCallBack, 0, sizeof(opRegFileCallBack));
    opRegFileCallBack.ObjectType = IoFileObjectType;
    opRegFileCallBack.Operations = OB_OPERATION_HANDLE_CREATE | OB_OPERATION_HANDLE_DUPLICATE;
    opRegFileCallBack.PreOperation = (POB_PRE_OPERATION_CALLBACK)&FileObjectpreCall;

    status = ObRegisterCallbacks(&obRegFileCallBack, &obHandle);
    if (!NT_SUCCESS(status))
    {
        kprintf(("注册回调错误 \n"));
        status = STATUS_UNSUCCESSFUL;
    }

    UNREFERENCED_PARAMETER(RegistryPath);
    Driver->DriverUnload = &UnDriver;
    return status;
}
```

该程序（附件中的XSafe2.sys）先加载，再加载Loader.sys同样可以任意user key加载成功并且不hook任何系统API和文件。



分析shellcode反复在读取哪个内存地址（2分）

shellcode加载之后，会检测双机调试

但是有一定的延迟，说明是主动检测的不是被动触发的，刚好遇上这个题，猜测是检测了一个调试器标志位。

不过这里感觉是得先确定一下 shellcode 的位置的，因为它带反调，不知道它怎么反调的，突然想到之前一位神仙把蓝屏代码删了导致电脑爆炸，于是我也想效仿一下看看可不可彳亍。

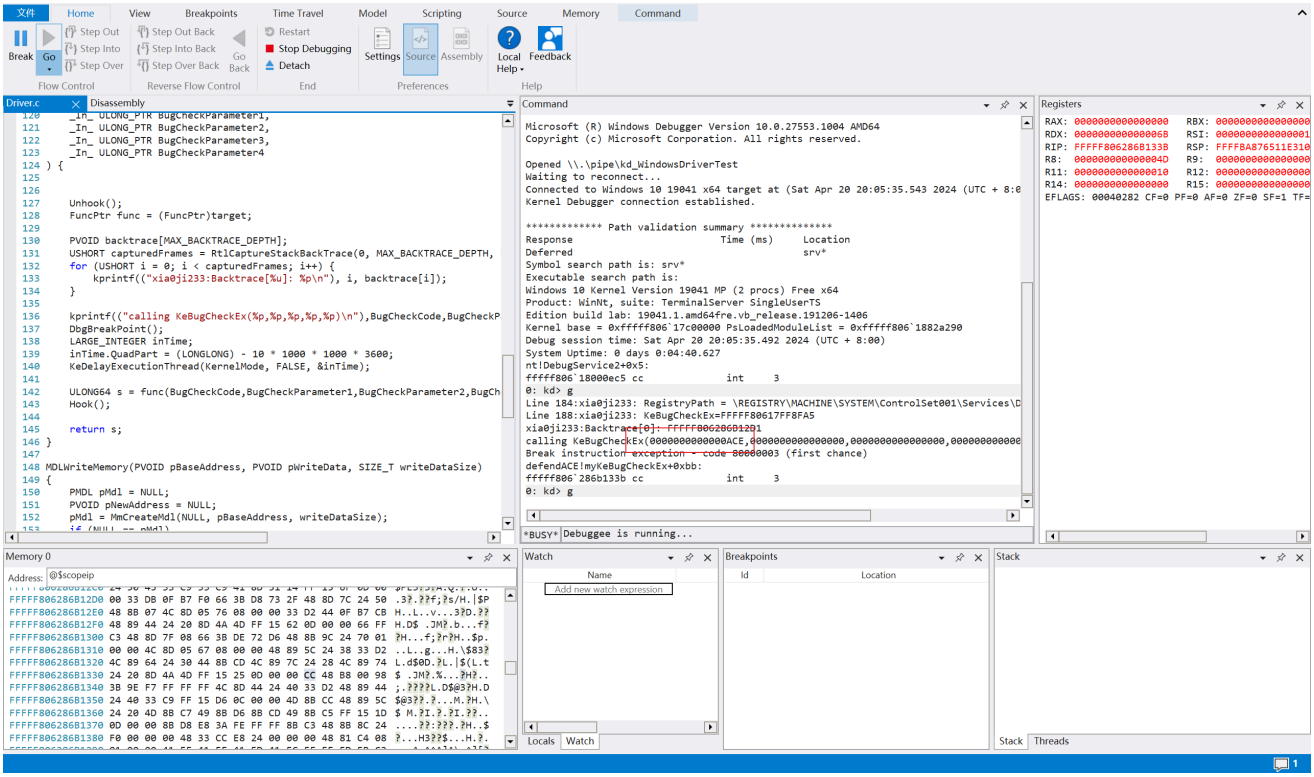
```
#include <ntifs.h>
#include <ntdef.h>
#include <ntstatus.h>
#include <ntddk.h>
#define MAX_BACKTRACE_DEPTH 20
#define SYMBOL L"\\??\\xia0ji2333"
#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)

UINT64 BaseAddr=NULL, DLLSize=0;
#define MAX_BACKTRACE_DEPTH 20

void DeleteDevice(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start delete device\n"), __LINE__);
    if (pDriver->DeviceObject) {
        UNICODE_STRING Sym;
        RtlInitUnicodeString(&Sym, SYMBOL);//CreateFile
        kprintf(("Line %d:xia0ji233: Delete Symbol\n"), __LINE__);
        IoDeleteSymbolicLink(&Sym);
        kprintf(("Line %d:xia0ji233: Delete Device\n"), __LINE__);
        IoDeleteDevice(pDriver->DeviceObject);
    }
    kprintf(("Line %d:xia0ji233: end delete device\n"), __LINE__);
}

HANDLE FileHandler = NULL;

char newcode[] = {
    0x48,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//mov rax,xxx
    0xFF,0xE0//jmp rax
};
char oldcode[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,
};


char* target;
KIRQL WPOFFx64()
{
    KIRQL irql = KeRaiseIrqlToDpcLevel();
    UINT64 cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}

void WPONx64(KIRQL irql)
```

```
{
    UINT64 cr0 = __readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

NTSTATUS Unhook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = oldcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}



NTSTATUS Hook() {
    KIRQL irql = WPOFFx64();
    for (int i = 0; i < sizeof(newcode); i++) {
        target[i] = newcode[i];
    }
    WPONx64(irql);
    return STATUS_SUCCESS;
}


PDRIVER_OBJECT g_Object = NULL;
typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;//驱动的进入点 DriverEntry
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;//驱动的满路径
    UNICODE_STRING BaseDllName;//不带路径的驱动名字
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    union {
        struct {
            ULONG TimeDateStamp;
        };
        struct {
            PVOID LoadedImports;
        };
    };
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

```
typedef NTSTATUS (* FuncPtr) (
    _In_ ULONG BugCheckCode,
    _In_ ULONG_PTR BugCheckParameter1,
    _In_ ULONG_PTR BugCheckParameter2,
    _In_ ULONG_PTR BugCheckParameter3,
    _In_ ULONG_PTR BugCheckParameter4
    );


ULONG  myKeBugCheckEx(_In_ ULONG BugCheckCode,
    _In_ ULONG_PTR BugCheckParameter1,
    _In_ ULONG_PTR BugCheckParameter2,
    _In_ ULONG_PTR BugCheckParameter3,
    _In_ ULONG_PTR BugCheckParameter4
) {


    Unhook();
    FuncPtr func = (FuncPtr)target;

    PVOID backtrace[MAX_BACKTRACE_DEPTH];
    USHORT capturedFrames = RtlCaptureStackBackTrace(0, MAX_BACKTRACE_DEPTH, backtrace, NULL);
    for (USHORT i = 0; i < capturedFrames; i++) {
        kprintf(("xia0ji233:Backtrace[%u]: %p\n"), i, backtrace[i]);
    }

    kprintf(("calling
KeBugCheckEx(%p,%p,%p,%p,%p)\n"),BugCheckCode,BugCheckParameter1,BugCheckParameter2,BugCheckPara
meter3,BugCheckParameter4);
    DbgBreakPoint();
    LARGE_INTEGER inTime;
    inTime.QuadPart = (LONGLONG) - 10 * 1000 * 1000 * 3600;
    KeDelayExecutionThread(KernelMode, FALSE, &inTime);

    ULONG64 s =
func(BugCheckCode,BugCheckParameter1,BugCheckParameter2,BugCheckParameter3,BugCheckParameter4);
    Hook();

    return s;
}

MDLWriteMemory(PVOID pBaseAddress, PVOID pWriteData, SIZE_T writeDataSize)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    pMdl = MmCreateMdl(NULL, pBaseAddress, writeDataSize);
    if (NULL == pMdl)
    {
        return FALSE;
    }
    MmBuildMdlForNonPagedPool(pMdl);
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
```

```
    }
    RtlCopyMemory(pNewAddress, pWriteData, writeDataSize);
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);
    return TRUE;
}
void DriverUnload(PDRIVER_OBJECT pDriver) {
    kprintf(("Line %d:xia0ji233: start unload\n"), __LINE__);
    Unhook();
    DeleteDevice(pDriver);
}


NTSTATUS DriverEntry(
    _In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING RegistryPath
) {
    DriverObject->DriverUnload = DriverUnload;
    kprintf(("Line %d:xia0ji233: RegistryPath = %S\n"), __LINE__, RegistryPath->Buffer);
    UNICODE_STRING unName = { 0 };
    RtlInitUnicodeString(&unName, L"KeBugCheckEx");
    target = ((ULONG64)MmGetSystemRoutineAddress(&unName))+5;
    kprintf(("Line %d:xia0ji233: KeBugCheckEx=%p\n"), __LINE__, target);
    g_Object = DriverObject;
    if (target) {
        for (int i = 0; i < sizeof(oldcode); i++) {
            oldcode[i] = target[i];
        }
        *(UINT64*)(newcode + 2) = myKeBugCheckEx;
        Hook();

    }
    else {
        kprintf(("xia0ji233:hahaha"));
    }
    return 0;
}
```

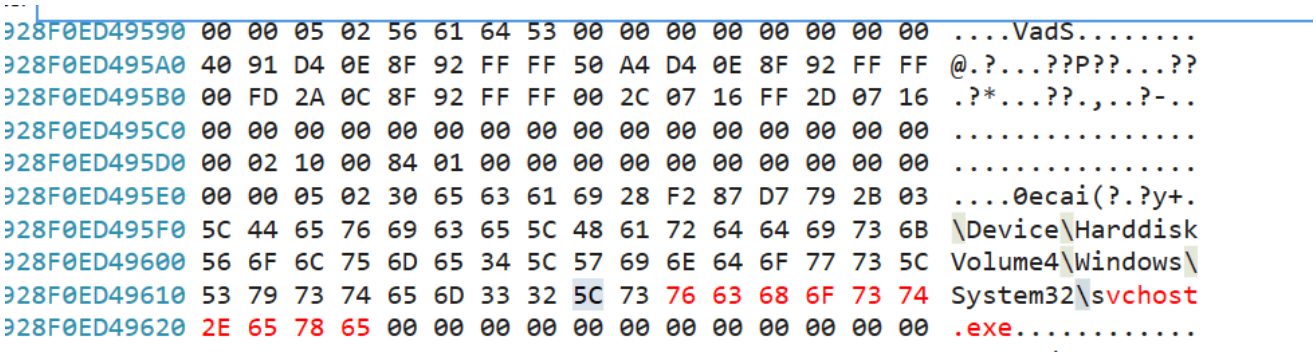
这里去hook KeBugCheckEx，然后直接让它 sleep 一小时，防止它蓝我，我有更多时间可以去分析。



直接拿捏住了蓝屏。

在后面的分析中发现了 GameSec.exe 的内存一直在被读，估计是在搜索进程，然后读取进程的内存，这一部分后面没有分析太出来。

并且在运行的时候发现会读一些 exe 文件的字符串，在 shellcode 开头 + 80 的位置，这里后续没继续分析了。



编写一个search程序，在Load驱动运行后找到内核内存空间中的shellcode，输出shellcode范围内的任意地址

随后想到去dump shellcode，这里注册回调的方式并不能成功拦截住，因此想到直接去 hook，判断线程起始位置是否在模块范围内，或者说在 Loader.sys 范围内，如果有都输出然后调试器看看内存像不像shellcode，无果，于是选择去跟一下，结果跟到一个类似shellcode的地方（很多浮点指令，看起来像垃圾指令的混淆），dump下来用 010 分析。

方法1

通过内存地址的特征可以发现，前八位可以通过 VAD 的方式去获取，后四个位每次加载似乎都是固定的，因此只需要爆破两个字节用一个特征去匹配就行了，这里。

```
#include "vad.h"
#include <ntifs.h>
```

```
#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)
// 定义VAD相对于EProcess头部偏移值
#define eprocess_offset_VadRoot 0x658
#define eprocess_offset_VadCount 0x668

VOID EnumVad(PMMVAD Root, PALL_VADS pBuffer, ULONG nCnt)
{
    if (!Root || !pBuffer || !nCnt)
    {
        return;
    }

    __try
    {
        if (nCnt > pBuffer->nCnt)
        {
            // 得到起始页与结束页
            ULONG64 endptr = (ULONG64)Root->Core.EndingVpnHigh;
            endptr = endptr << 32;

            ULONG64 startptr = (ULONG64)Root->Core.StartingVpnHigh;
            startptr = startptr << 32;

            // 得到根节点
            pBuffer->VadInfos[pBuffer->nCnt].pVad = (ULONG_PTR)Root;

            // 起始页: startingVpn * 0x1000
            pBuffer->VadInfos[pBuffer->nCnt].startVpn = (startptr | Root->Core.StartingVpn) <<
PAGE_SHIFT;

            // 结束页: EndVpn * 0x1000 + 0xfff
            pBuffer->VadInfos[pBuffer->nCnt].endVpn = ((endptr | Root->Core.EndingVpn) <<
PAGE_SHIFT) + 0xfff;

            // VAD标志 928 = Mapped    1049088 = Private   ....
            pBuffer->VadInfos[pBuffer->nCnt].flags = Root->Core.u1.Flags.flag;

            // 验证节点可读性
            if (MmIsAddressValid(Root->Subsection) && MmIsAddressValid(Root->Subsection-
>ControlArea))
            {
                if (MmIsAddressValid((PVOID)((Root->Subsection->ControlArea->FilePointer.Value
>> 4) << 4)))
                {
                    pBuffer->VadInfos[pBuffer->nCnt].pFileObject = ((Root->Subsection-
>ControlArea->FilePointer.Value >> 4) << 4);
                }
            }
            pBuffer->nCnt++;
        }

        if (MmIsAddressValid(Root->Core.VadNode.Left))
        {
            // 递归枚举左子树
            EnumVad((PMMVAD)Root->Core.VadNode.Left, pBuffer, nCnt);
        }
```

```
        if (MmIsAddressValid(Root->Core.VadNode.Right))
        {
            // 递归枚举右子树
            EnumVad((PMMVAD)Root->Core.VadNode.Right, pBuffer, nCnt);
        }
    }
    __except (1)
    {
    }
}

BOOLEAN EnumProcessVad(ULONG Pid, PALL_VADS pBuffer, ULONG nCnt)
{
    PEPROCESS Peprocess = 0;
    PRTL_AVL_TREE Table = NULL;
    PMMVAD Root = NULL;

    // 通过进程PID得到进程EProcess
    if (NT_SUCCESS(PsLookupProcessByProcessId((HANDLE)Pid, &Peprocess)))
    {
        // 与偏移相加得到VAD头节点
        Table = (PRTL_AVL_TREE)((UCHAR*)Peprocess + eprocess_offset_VadRoot);
        if (!MmIsAddressValid(Table) || !eprocess_offset_VadRoot)
        {
            return FALSE;
        }

        __try
        {
            // 取出头节点
            Root = (PMMVAD)Table->Root;

            if (nCnt > pBuffer->nCnt)
            {
                // 得到起始页与结束页
                ULONG64 endptr = (ULONG64)Root->Core.EndingVpnHigh;
                endptr = endptr << 32;

                ULONG64 startptr = (ULONG64)Root->Core.StartingVpnHigh;
                startptr = startptr << 32;

                pBuffer->VadInfos[pBuffer->nCnt].pVad = (ULONG_PTR)Root;

                // 起始页: startingVpn * 0x1000
                pBuffer->VadInfos[pBuffer->nCnt].startVpn = (startptr | Root->Core.StartingVpn)
<< PAGE_SHIFT;

                // 结束页: EndVpn * 0x1000 + 0xfff
                pBuffer->VadInfos[pBuffer->nCnt].endVpn = (endptr | Root->Core.EndingVpn) <<
PAGE_SHIFT;
                pBuffer->VadInfos[pBuffer->nCnt].flags = Root->Core.u1.Flags.flag;

                if (MmIsAddressValid(Root->Subsection) && MmIsAddressValid(Root->Subsection-
>ControlArea))
                {
                    if (MmIsAddressValid((PVOID)((Root->Subsection->ControlArea-
>FilePointer.Value >> 4) << 4)))
                    {
```

```
                    pBuffer->VadInfos[pBuffer->nCnt].pFileObject = ((Root->Subsection->ControlArea->FilePointer.Value >> 4) << 4);
                }
            }
            pBuffer->nCnt++;
        }

        // 枚举左子树
        if (Table->Root->Left)
        {
            EnumVad((MMVAD*)Table->Root->Left, pBuffer, nCnt);
        }

        // 枚举右子树
        if (Table->Root->Right)
        {
            EnumVad((MMVAD*)Table->Root->Right, pBuffer, nCnt);
        }
    }
    __finally
    {
        ObDereferenceObject(Peprocess);
    }
}
else
{
    return FALSE;
}

return TRUE;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    kprintf(("unload\n"));
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    kprintf(("hello xia0ji233\n"));

    typedef struct
    {
        ULONG nPid;
        ULONG nSize;
        PALL_VADS pBuffer;
    }VADProcess;

    __try
    {
        VADProcess vad = { 0 };

        vad.nPid = 4;

        // 默认有1000个线程
        vad.nSize = sizeof(VAD_INFO) * 0x5000 + sizeof(ULONG);

        // 分配临时空间
```

```
        vad.pBuffer = (PALL_VADS)ExAllocatePool(PagedPool, vad.nSize);

        // 根据传入长度得到枚举数量
        ULONG nCount = (vad.nSize - sizeof(ULONG)) / sizeof(VAD_INFO);

        // 枚举VAD
        EnumProcessVad(vad.nPid, vad.pBuffer, nCount);

        uintptr_t addr;

        for (ULONG64 i = 0x0; i <65536; i++)
        {
            addr = vad.pBuffer->VadInfos[0].pVad & 0xffffffff00000000;
            addr = addr + 0x1000;
            addr = addr + (i<<16);
            if (MmIsAddressValid((PVOID)addr) && *((ULONG64 *)addr) == 0x7C8B483024748B48 )
            {
                kprintf(("shellcode found in %p\n"), addr);
                //DbgBreakPoint();
                goto end;
            }

        }
    }
    __except (1)

    {
        kprintf(("Something Error1\n"));
    }
end:
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

vad.h

```
#pragma once
#include <ntifs.h>

typedef struct _MM_GRAPHICS_VAD_FLAGS        // 15 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     ULONG32      Lock : 1;                    // 0 BitPosition
    /*0x000*/     ULONG32      LockContended : 1;           // 1 BitPosition
    /*0x000*/     ULONG32      DeleteInProgress : 1;        // 2 BitPosition
    /*0x000*/     ULONG32      NoChange : 1;                // 3 BitPosition
    /*0x000*/     ULONG32      VadType : 3;                 // 4 BitPosition
    /*0x000*/     ULONG32      Protection : 5;              // 7 BitPosition
    /*0x000*/     ULONG32      PreferredNode : 6;           // 12 BitPosition
    /*0x000*/     ULONG32      PageSize : 2;                // 18 BitPosition
    /*0x000*/     ULONG32      PrivateMemoryAlwaysSet : 1; // 20 BitPosition
    /*0x000*/     ULONG32      WriteWatch : 1;              // 21 BitPosition
    /*0x000*/     ULONG32      FixedLargePageSize : 1;      // 22 BitPosition
    /*0x000*/     ULONG32      ZeroFillPagesOptional : 1;  // 23 BitPosition
    /*0x000*/     ULONG32      GraphicsAlwaysSet : 1;       // 24 BitPosition
    /*0x000*/     ULONG32      GraphicsUseCoherentBus : 1; // 25 BitPosition
    /*0x000*/     ULONG32      GraphicsPageProtection : 3; // 26 BitPosition
}MM_GRAPHICS_VAD_FLAGS, * PMM_GRAPHICS_VAD_FLAGS;
```

```
typedef struct _MM_PRIVATE_VAD_FLAGS         // 15 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     ULONG32      Lock : 1;                   // 0 BitPosition
    /*0x000*/     ULONG32      LockContended : 1;          // 1 BitPosition
    /*0x000*/     ULONG32      DeleteInProgress : 1;       // 2 BitPosition
    /*0x000*/     ULONG32      NoChange : 1;               // 3 BitPosition
    /*0x000*/     ULONG32      VadType : 3;                // 4 BitPosition
    /*0x000*/     ULONG32      Protection : 5;             // 7 BitPosition
    /*0x000*/     ULONG32      PreferredNode : 6;          // 12 BitPosition
    /*0x000*/     ULONG32      PageSize : 2;               // 18 BitPosition
    /*0x000*/     ULONG32      PrivateMemoryAlwaysSet : 1; // 20 BitPosition
    /*0x000*/     ULONG32      WriteWatch : 1;             // 21 BitPosition
    /*0x000*/     ULONG32      FixedLargePageSize : 1;     // 22 BitPosition
    /*0x000*/     ULONG32      ZeroFillPagesOptional : 1;  // 23 BitPosition
    /*0x000*/     ULONG32      Graphics : 1;               // 24 BitPosition
    /*0x000*/     ULONG32      Enclave : 1;                // 25 BitPosition
    /*0x000*/     ULONG32      ShadowStack : 1;            // 26 BitPosition
}MM_PRIVATE_VAD_FLAGS, * PMM_PRIVATE_VAD_FLAGS;


typedef struct _MMVAD_FLAGS            // 9 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     ULONG32      Lock : 1;             // 0 BitPosition
    /*0x000*/     ULONG32      LockContended : 1;    // 1 BitPosition
    /*0x000*/     ULONG32      DeleteInProgress : 1; // 2 BitPosition
    /*0x000*/     ULONG32      NoChange : 1;         // 3 BitPosition
    /*0x000*/     ULONG32      VadType : 3;          // 4 BitPosition
    /*0x000*/     ULONG32      Protection : 5;       // 7 BitPosition
    /*0x000*/     ULONG32      PreferredNode : 6;    // 12 BitPosition
    /*0x000*/     ULONG32      PageSize : 2;         // 18 BitPosition
    /*0x000*/     ULONG32      PrivateMemory : 1;    // 20 BitPosition
}MMVAD_FLAGS, * PMMVAD_FLAGS;

typedef struct _MM_SHARED_VAD_FLAGS            // 11 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     ULONG32      Lock : 1;                     // 0 BitPosition
    /*0x000*/     ULONG32      LockContended : 1;            // 1 BitPosition
    /*0x000*/     ULONG32      DeleteInProgress : 1;         // 2 BitPosition
    /*0x000*/     ULONG32      NoChange : 1;                 // 3 BitPosition
    /*0x000*/     ULONG32      VadType : 3;                  // 4 BitPosition
    /*0x000*/     ULONG32      Protection : 5;               // 7 BitPosition
    /*0x000*/     ULONG32      PreferredNode : 6;            // 12 BitPosition
    /*0x000*/     ULONG32      PageSize : 2;                 // 18 BitPosition
    /*0x000*/     ULONG32      PrivateMemoryAlwaysClear : 1; // 20 BitPosition
    /*0x000*/     ULONG32      PrivateFixup : 1;             // 21 BitPosition
    /*0x000*/     ULONG32      HotPatchAllowed : 1;          // 22 BitPosition
}MM_SHARED_VAD_FLAGS, * PMM_SHARED_VAD_FLAGS;

typedef struct _MMVAD_FLAGS2             // 7 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     ULONG32      FileOffset : 24;        // 0 BitPosition
    /*0x000*/     ULONG32      Large : 1;              // 24 BitPosition
    /*0x000*/     ULONG32      TrimBehind : 1;         // 25 BitPosition
    /*0x000*/     ULONG32      Inherit : 1;            // 26 BitPosition
    /*0x000*/     ULONG32      NoValidationNeeded : 1; // 27 BitPosition
    /*0x000*/     ULONG32      PrivateDemandZero : 1;  // 28 BitPosition
    /*0x000*/     ULONG32      Spare : 3;              // 29 BitPosition
}MMVAD_FLAGS2, * PMMVAD_FLAGS2;
```

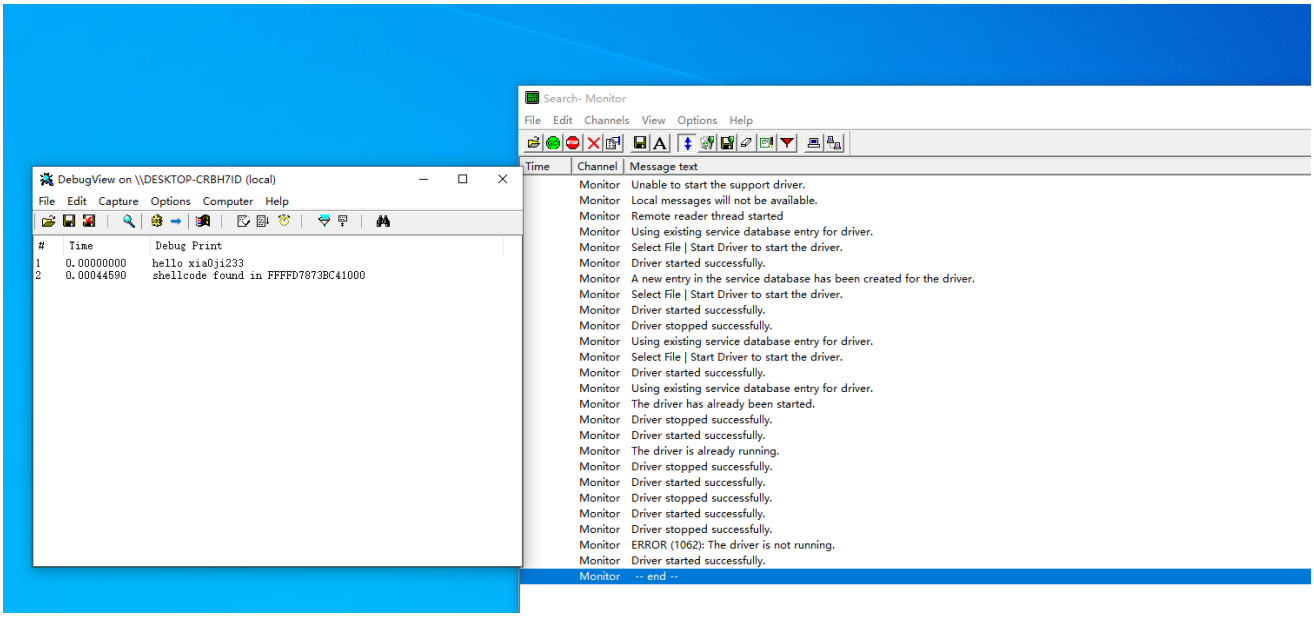
```
typedef struct _MMVAD_SHORT
{
    RTL_BALANCED_NODE VadNode;
    UINT32 StartingVpn;                /*0x18*/
    UINT32 EndingVpn;                  /*0x01C*/
    UCHAR StartingVpnHigh;
    UCHAR EndingVpnHigh;
    UCHAR CommitChargeHigh;
    UCHAR SpareNT64VadUChar;
    INT32 ReferenceCount;
    EX_PUSH_LOCK PushLock;             /*0x028*/
    struct
    {
        union
        {
            ULONG_PTR flag;
            MM_PRIVATE_VAD_FLAGS PrivateVadFlags;                        /*0x030*/
            MMVAD_FLAGS  VadFlags;
            MM_GRAPHICS_VAD_FLAGS GraphicsVadFlags;
            MM_SHARED_VAD_FLAGS   SharedVadFlags;
        }Flags;

    }u1;

    PVOID EventList;                        /*0x038*/

}MMVAD_SHORT, * PMMVAD_SHORT;

typedef struct _MMADDRESS_NODE
{
    ULONG64 u1;
    struct _MMADDRESS_NODE* LeftChild;
    struct _MMADDRESS_NODE* RightChild;
    ULONG64 StartingVpn;
    ULONG64 EndingVpn;
}MMADDRESS_NODE, * PMMADDRESS_NODE;

typedef struct _MMEXTEND_INFO     // 2 elements, 0x10 bytes (sizeof)
{
    /*0x000*/     UINT64       CommittedSize;
    /*0x008*/     ULONG32      ReferenceCount;
    /*0x00C*/     UINT8        _PADDING0_[0x4];
}MMEXTEND_INFO, * PMMEXTEND_INFO;
struct _SEGMENT
{
    struct _CONTROL_AREA* ControlArea;
    ULONG TotalNumberOfPtes;
    ULONG SegmentFlags;
    ULONG64 NumberOfCommittedPages;
    ULONG64 SizeOfSegment;
    union
    {
        struct _MMEXTEND_INFO* ExtendInfo;
        void* BasedAddress;
    }u;
    ULONG64 SegmentLock;
    ULONG64 u1;
```

```
    ULONG64 u2;
    PVOID* PrototypePte;
    ULONGLONG ThePtes[0x1];
};

typedef struct _EX_FAST_REF
{
    union
    {
        PVOID Object;
        ULONG_PTR RefCnt : 3;
        ULONG_PTR Value;
    };
} EX_FAST_REF, * PEX_FAST_REF;

typedef struct _CONTROL_AREA                    // 17 elements, 0x80 bytes (sizeof)
{
    /*0x000*/     struct _SEGMENT* Segment;
    union                                       // 2 elements, 0x10 bytes (sizeof)
    {
        /*0x008*/         struct _LIST_ENTRY ListHead;             // 2 elements, 0x10 bytes
(sizeof)
        /*0x008*/         VOID* AweContext;
    };
    /*0x018*/     UINT64       NumberOfSectionReferences;
    /*0x020*/     UINT64       NumberOfPfnReferences;
    /*0x028*/     UINT64       NumberOfMappedViews;
    /*0x030*/     UINT64       NumberOfUserReferences;
    /*0x038*/     ULONG32 u;                     // 2 elements, 0x4 bytes (sizeof)
    /*0x03C*/     ULONG32 u1;                    // 2 elements, 0x4 bytes (sizeof)
    /*0x040*/     struct _EX_FAST_REF FilePointer;             // 3 elements, 0x8 bytes
(sizeof)
    // 4 elements, 0x8 bytes (sizeof)
}CONTROL_AREA, * PCONTROL_AREA;

typedef struct _SUBSECTION_
{
    struct _CONTROL_AREA* ControlArea;

}SUBSECTION, * PSUBSECTION;

typedef struct _MMVAD
{
    MMVAD_SHORT Core;
    union                 /*0x040*/
    {
        UINT32 LongFlags2;
        //现在用不到省略
        MMVAD_FLAGS2 VadFlags2;

    }u2;
    PSUBSECTION Subsection;              /*0x048*/
    PVOID FirstPrototypePte;        /*0x050*/
    PVOID LastContiguousPte;        /*0x058*/
    LIST_ENTRY ViewLinks;           /*0x060*/
    PEPROCESS VadsProcess;          /*0x070*/
    PVOID u4;                       /*0x078*/
    PVOID FileObject;               /*0x080*/
```

```
}MMVAD, * PMMVAD;

typedef struct _RTL_AVL_TREE         // 1 elements, 0x8 bytes (sizeof)
{
    /*0x000*/     struct _RTL_BALANCED_NODE* Root;
}RTL_AVL_TREE, * PRTL_AVL_TREE;

typedef struct _VAD_INFO_
{
    ULONG_PTR pVad;
    ULONG_PTR startVpn;
    ULONG_PTR endVpn;
    ULONG_PTR pFileObject;
    ULONG_PTR flags;
}VAD_INFO, * PVAD_INFO;

typedef struct _ALL_VADS_
{
    ULONG nCnt;
    VAD_INFO VadInfos[1];
}ALL_VADS, * PALL_VADS;

typedef struct _MMSECTION_FLAGS                         // 27 elements, 0x4 bytes (sizeof)
{
    /*0x000*/     UINT32       BeingDeleted : 1;                     // 0 BitPosition
    /*0x000*/     UINT32       BeingCreated : 1;                     // 1 BitPosition
    /*0x000*/     UINT32       BeingPurged : 1;                      // 2 BitPosition
    /*0x000*/     UINT32       NoModifiedWriting : 1;                // 3 BitPosition
    /*0x000*/     UINT32       FailAllIo : 1;                        // 4 BitPosition
    /*0x000*/     UINT32       Image : 1;                            // 5 BitPosition
    /*0x000*/     UINT32       Based : 1;                            // 6 BitPosition
    /*0x000*/     UINT32       File : 1;                             // 7 BitPosition
    /*0x000*/     UINT32       AttemptingDelete : 1;                 // 8 BitPosition
    /*0x000*/     UINT32       PrefetchCreated : 1;                  // 9 BitPosition
    /*0x000*/     UINT32       PhysicalMemory : 1;                   // 10 BitPosition
    /*0x000*/     UINT32       ImageControlAreaOnRemovableMedia : 1; // 11 BitPosition
    /*0x000*/     UINT32       Reserve : 1;                          // 12 BitPosition
    /*0x000*/     UINT32       Commit : 1;                           // 13 BitPosition
    /*0x000*/     UINT32       NoChange : 1;                         // 14 BitPosition
    /*0x000*/     UINT32       WasPurged : 1;                        // 15 BitPosition
    /*0x000*/     UINT32       UserReference : 1;                    // 16 BitPosition
```

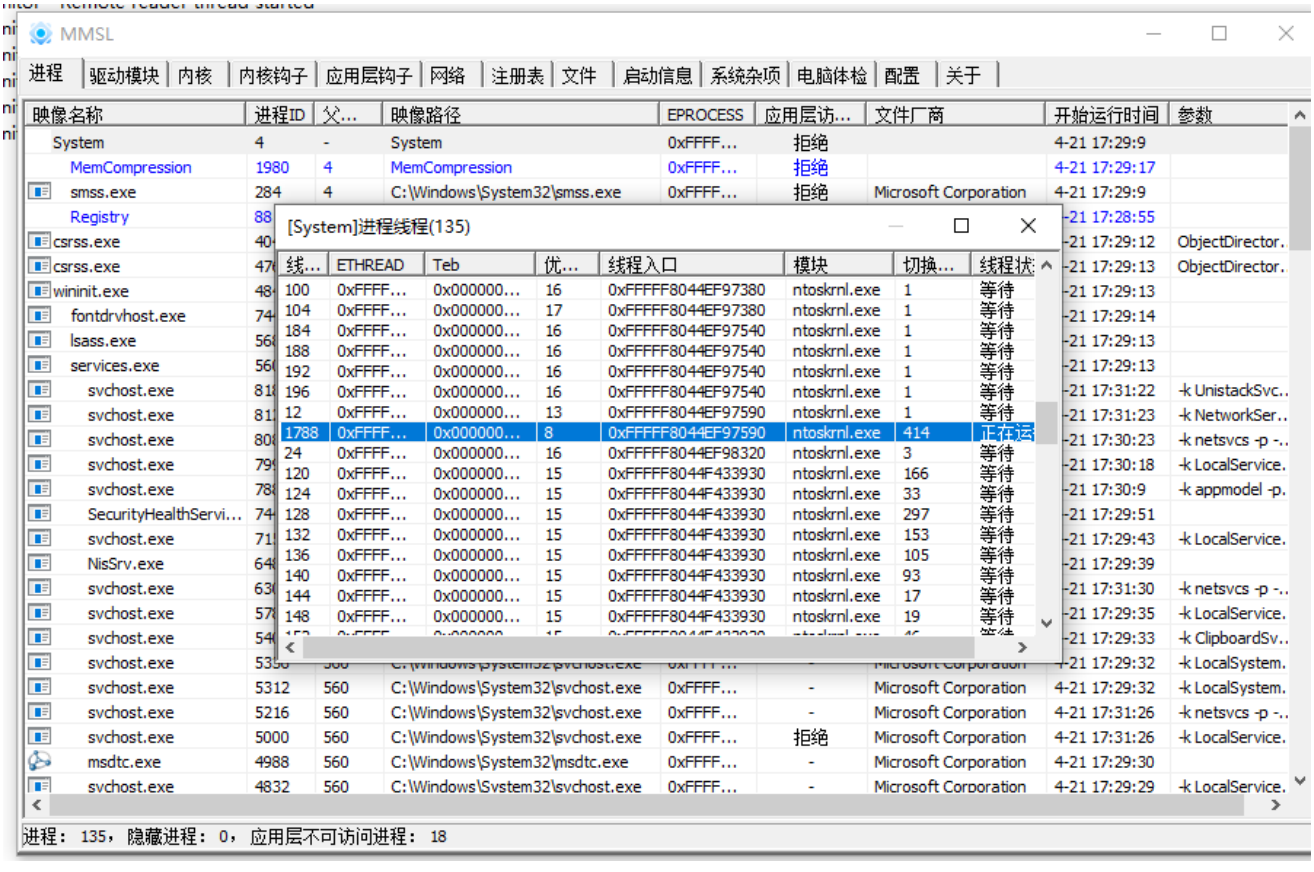
```
    /*0x000*/     UINT32       GlobalMemory : 1;                    // 17 BitPosition
    /*0x000*/     UINT32       DeleteOnClose : 1;                   // 18 BitPosition
    /*0x000*/     UINT32       FilePointerNull : 1;                 // 19 BitPosition
    /*0x000*/     ULONG32      PreferredNode : 6;                   // 20 BitPosition
    /*0x000*/     UINT32       GlobalOnlyPerSession : 1;            // 26 BitPosition
    /*0x000*/     UINT32       UserWritable : 1;                    // 27 BitPosition
    /*0x000*/     UINT32       SystemVaAllocated : 1;               // 28 BitPosition
    /*0x000*/     UINT32       PreferredFsCompressionBoundary : 1;  // 29 BitPosition
    /*0x000*/     UINT32       UsingFileExtents : 1;                // 30 BitPosition
    /*0x000*/     UINT32       PageSize64K : 1;                     // 31 BitPosition
}MMSECTION_FLAGS, * PMMSECTION_FLAGS;

typedef struct _SECTION                          // 9 elements, 0x40 bytes (sizeof)
{
    /*0x000*/     struct _RTL_BALANCED_NODE SectionNode;       // 6 elements, 0x18 bytes
(sizeof)
    /*0x018*/     UINT64       StartingVpn;
    /*0x020*/     UINT64       EndingVpn;
    /*0x028*/     union {
        PCONTROL_AREA   ControlArea;
        PVOID   FileObject;

    }u1;                    // 4 elements, 0x8 bytes (sizeof)
    /*0x030*/     UINT64       SizeOfSection;
    /*0x038*/     union {
        ULONG32 LongFlags;
        MMSECTION_FLAGS Flags;
    }u;                     // 2 elements, 0x4 bytes (sizeof)
    struct                                       // 3 elements, 0x4 bytes (sizeof)
    {
        /*0x03C*/         ULONG32      InitialPageProtection : 12; // 0 BitPosition
        /*0x03C*/         ULONG32      SessionId : 19;             // 12 BitPosition
        /*0x03C*/         ULONG32      NoValidationNeeded : 1;     // 31 BitPosition

    };
}SECTION, * PSECTION;
```

先加载 loader 再加载 search，成功输出 shellcode 的地址，特征码匹配前八个字节，在自己的环境只输出了一个地址，如果输出多个地址可以考虑加长特征码。



方法2

通过获取到线程结构得到它的线程上下文，输出 RIP 的值应该也行，通过PCHUNTER看到shellcode运行的线程

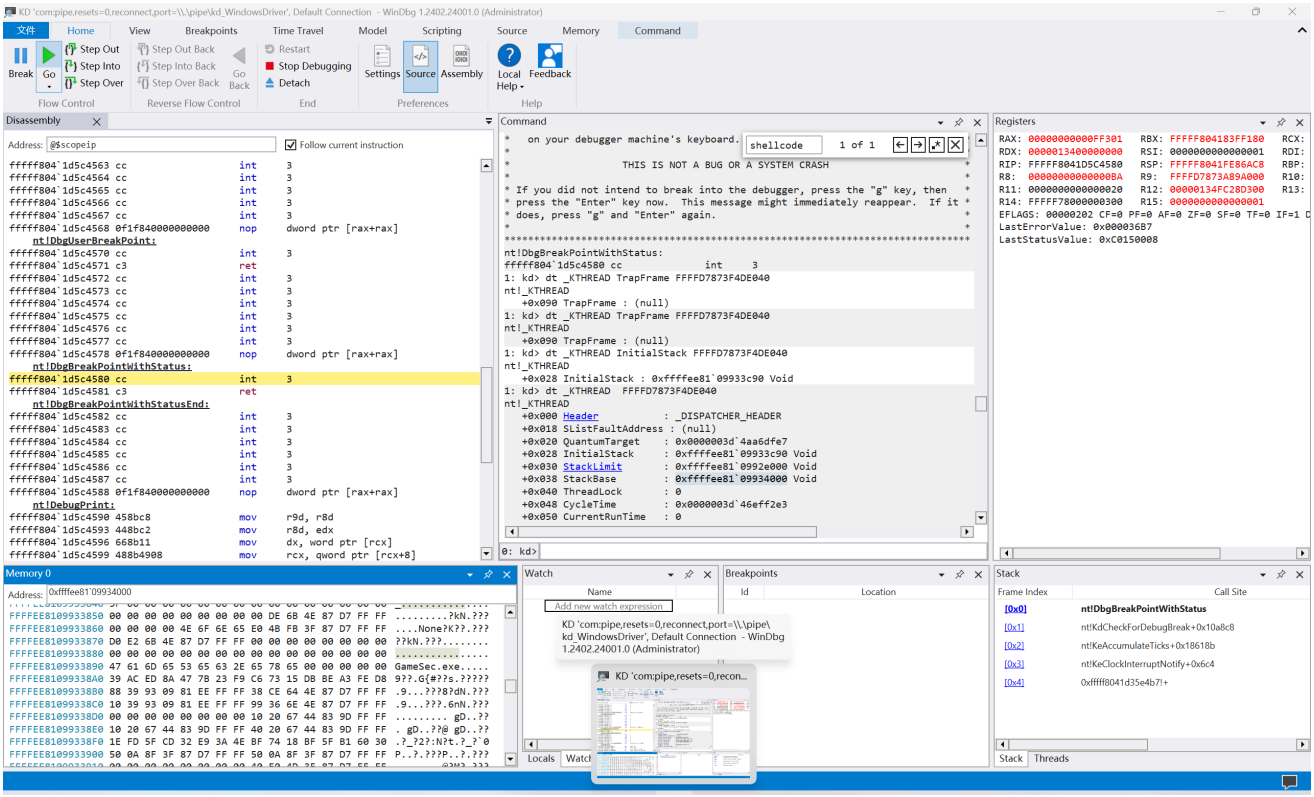


通过分析可知 shellcode 执行的线程具有如下特点：

与 TID=12 的线程入口相同

保持运行状态

据此可以筛选得到这个线程，通过线程结构体可以找到它的栈

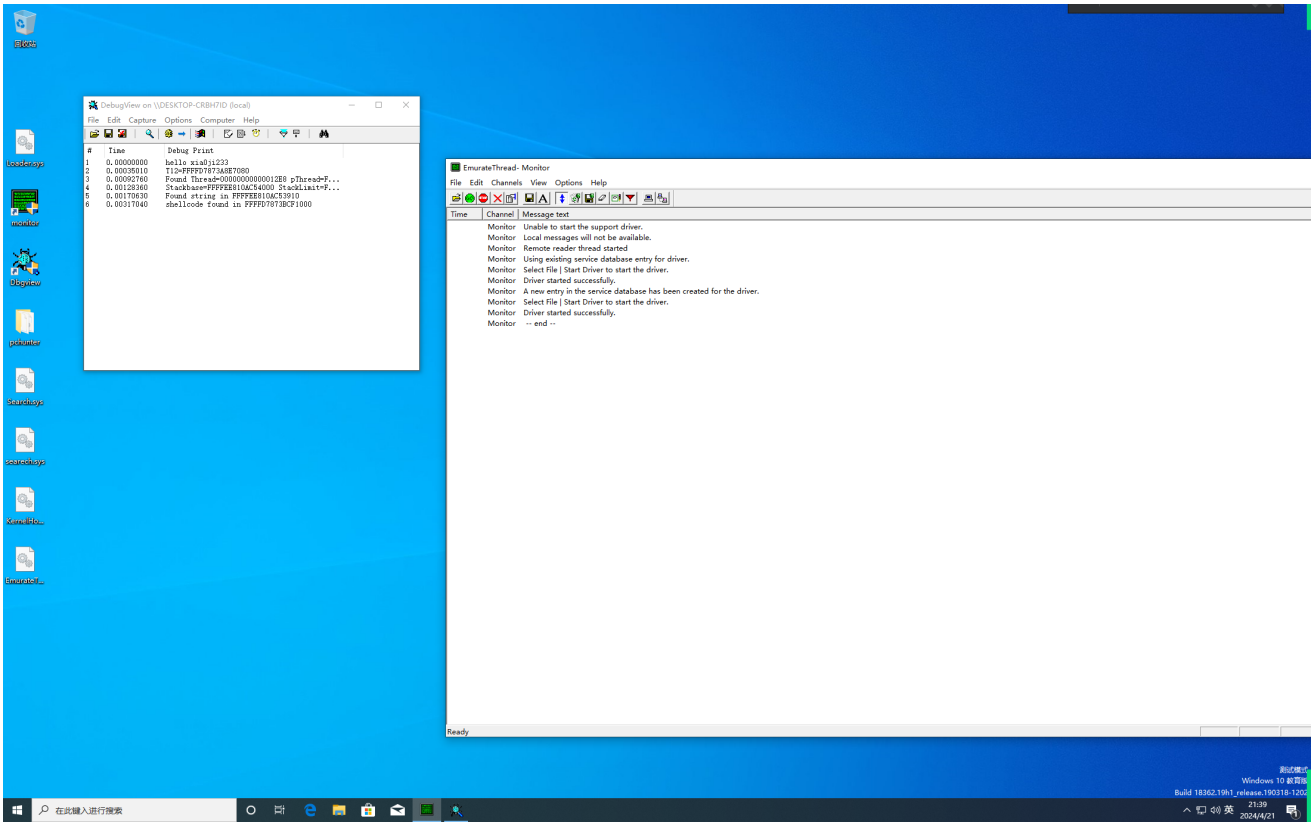


多次运行发现栈中存在 GameSec.exe 这个字符串。并且在它 - 0x28 的位置有一个地址，那个地址前八位和shellcode一致，所以同样可以爆破+特征码匹配。

```
#include<ntifs.h>
#include <ntddk.h>
#include <ntstrsafe.h>

#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)
#define MAX_BACKTRACE_DEPTH 20
ULONG64 num = 0;
NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(ULONG, PVOID, ULONG, PULONG);

VOID DRIVERUNLOAD(_In_ struct _DRIVER_OBJECT* DriverObject)
{
    kprintf(("unload\n"));

}


NTSTATUS EnumerateKernelThreads() {


    PETHREAD T12 = NULL;
    PETHREAD T;
    PsLookupThreadByThreadId(12, &T12);
    kprintf(("T12=%p\n"), T12);
    ULONG64 Start = *(ULONG64 *)((ULONG64)T12 + 0x620);
    HANDLE TargetThread = 0;
    for (int i = 16; i < 0x20000; i+=4) {
        T = NULL;
        PsLookupThreadByThreadId(i, &T);
        if (T) {
```

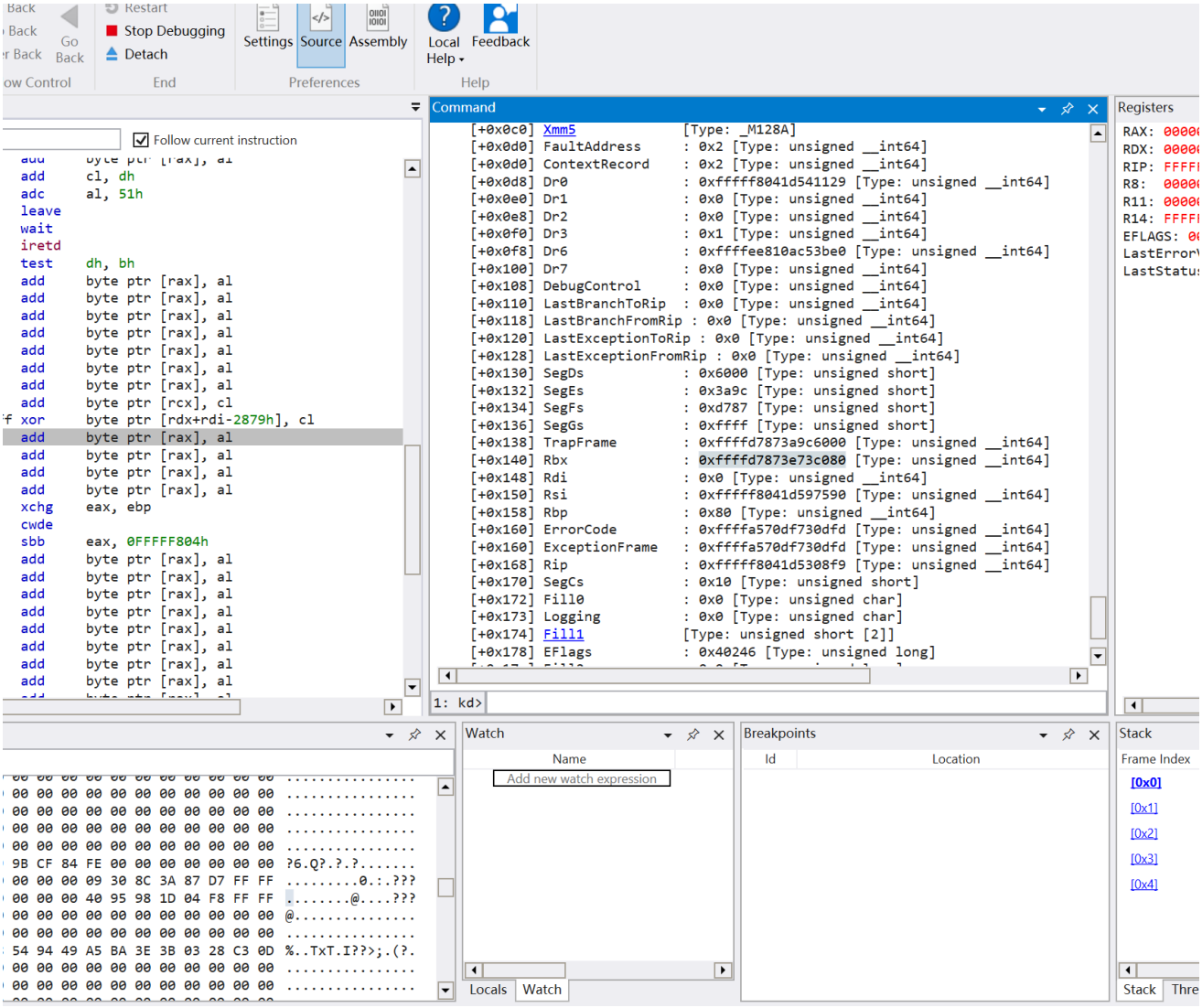
```
            ULONG64 Startaddr = *(ULONG64 *)((ULONG64)T + 0x620);
            if (Startaddr == Start) {
                kprintf(("Found Thread=%p pThread=%p\n"), i,T);

                TargetThread = i;
                break;
            }

        }
    }


    if (T != NULL) {
        ULONG64 StackBase = 0;
        ULONG64 StackLimit = 0;
        StackBase= *(ULONG64 *)((ULONG64)T + 0x38);
        StackLimit=*(ULONG64 *)((ULONG64)T + 0x30);
        kprintf(("Stackbase=%p StackLimit=%p\n"), StackBase, StackLimit);
        for (ULONG64 addr = StackBase-0x10 ; addr > StackLimit; addr -= 8) {
            if (!strcmp(addr, "GameSec.exe")) {
                kprintf(("Found string in %p\n"), addr);
                uintptr_t address;
                for (ULONG64 i = 0x0; i <65536; i++)
                {
                    address = (*(ULONG64*)(addr-0x28)) & 0xffffffff00000000;
                    address = address + 0x1000;
                    address = address + (i<<16);
                    if (MmIsAddressValid((PVOID)address) && *((ULONG64 *)address) ==
0x7C8B483024748B48 )
                    {
                        kprintf(("shellcode found in %p\n"), address);
                        //DbgBreakPoint();
                        break;
                    }

                }
                break;
            }

        }


    }


    return STATUS_SUCCESS;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING pReg)
{

    kprintf(("hello xia0ji233\n"));

    EnumerateKernelThreads();
    pDriver->DriverUnload = DRIVERUNLOAD;
    return STATUS_SUCCESS;
}
```

先运行 Loader.sys，再运行EmurateThread.sys，可以成功输出shellcode的地址。



方法3

随后我发现线程结构体中的 TrapFrame 有点东西，通过一段时间的运行之后，发现它的一些寄存器中会带上点东西

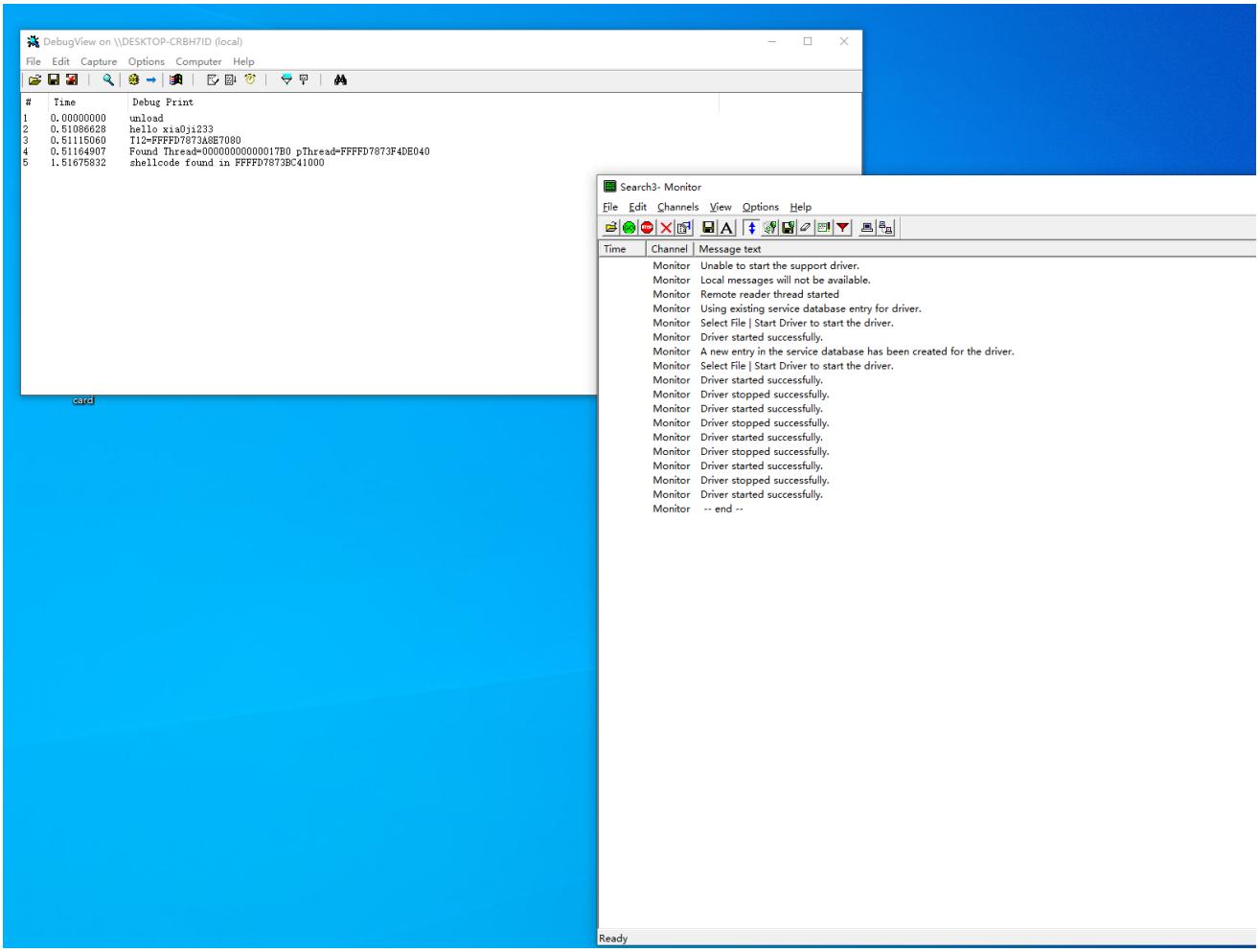


这里我选 Rdx，取前8位，暴力搜索4位（65536，可接受范围内）匹配。

```
#include<ntifs.h>
#include <ntddk.h>
#include <ntstrsafe.h>

#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)
#define MAX_BACKTRACE_DEPTH 20
ULONG64 num = 0;
NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(ULONG, PVOID, ULONG, PULONG);

VOID DRIVERUNLOAD(_In_ struct _DRIVER_OBJECT* DriverObject)
{
    kprintf(("unload\n"));

}


NTSTATUS EnumerateKernelThreads() {


    PETHREAD T12 = NULL;
    PETHREAD T;
```

```
    PsLookupThreadByThreadId(12, &T12);
    kprintf(("T12=%p\n"), T12);
    ULONG64 Start = *(ULONG64 *)((ULONG64)T12 + 0x620);
    HANDLE TargetThread = 0;
    for (int i = 16; i < 0x20000; i+=4) {
        T = NULL;
        PsLookupThreadByThreadId(i, &T);
        if (T) {
            ULONG64 Startaddr = *(ULONG64 *)((ULONG64)T + 0x620);
            if (Startaddr == Start) {
                kprintf(("Found Thread=%p pThread=%p\n"), i,T);

                TargetThread = i;
                break;
            }

        }
    }


    if (T != NULL) {
        KTRAP_FRAME* TrapFrame = NULL;
        while (!TrapFrame) {
            TrapFrame=(KTRAP_FRAME *)*(ULONG64 *)((ULONG64)T + 0x90);
            LARGE_INTEGER inTime;
            inTime.QuadPart = 1000 * -10000;
            KeDelayExecutionThread(KernelMode, FALSE, &inTime);
        }

        ULONG64 address;
        for (ULONG64 i = 0x0; i <65536; i++)
        {
            address = TrapFrame->Rdx & 0xffffffff00000000;
            address = address + 0x1000;
            address = address + (i<<16);
            if (MmIsAddressValid((PVOID)address) && *((ULONG64 *)address) == 0x7C8B483024748B48
)
            {
                kprintf(("shellcode found in %p\n"), address);
                //DbgBreakPoint();
                break;
            }

        }

    }


    return STATUS_SUCCESS;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING pReg)
{

    kprintf(("hello xia0ji233\n"));

    EnumerateKernelThreads();
    pDriver->DriverUnload = DRIVERUNLOAD;
```

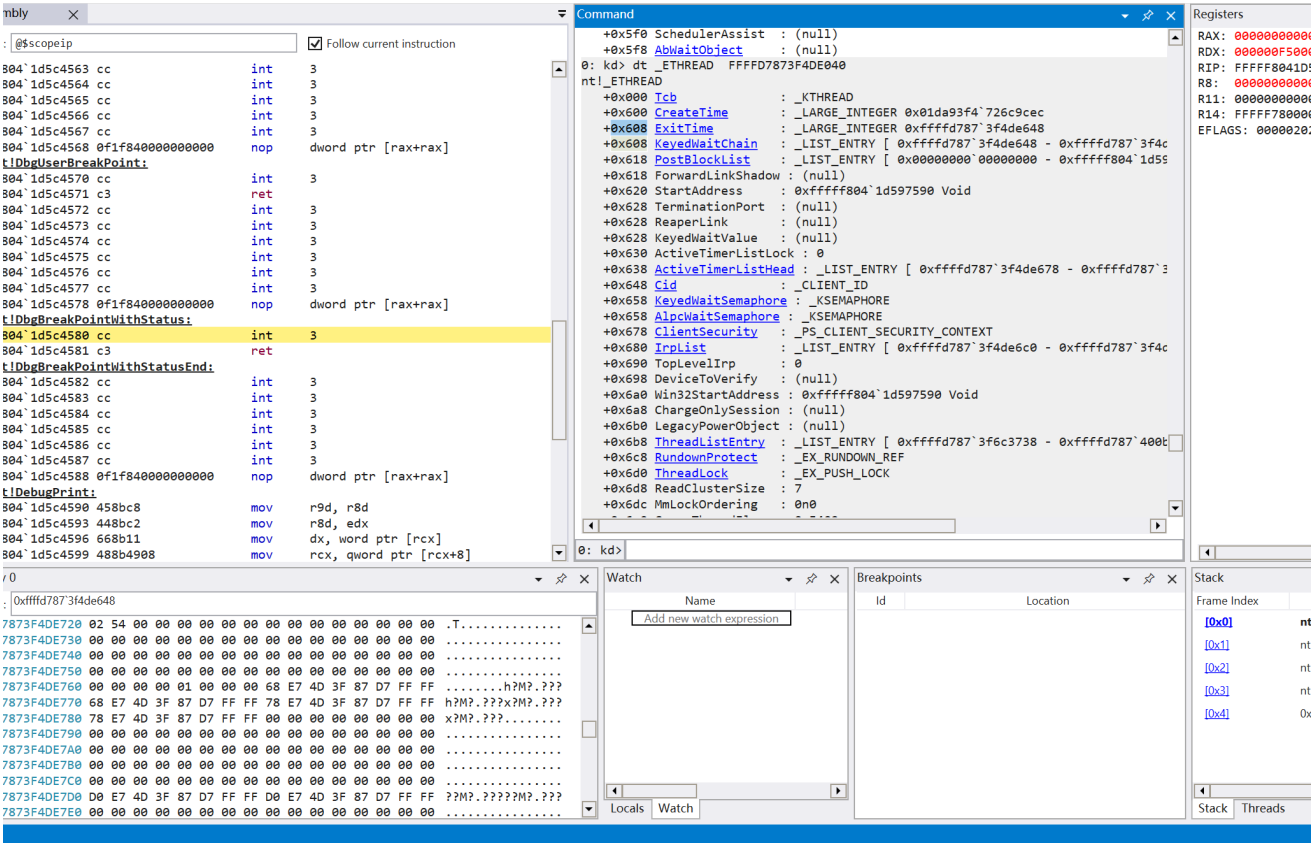
```
    return STATUS_SUCCESS;
}
```



先运行 Loader.sys 再运行 Search3.sys，可以直接得到 shellcode 的地址。

方法4

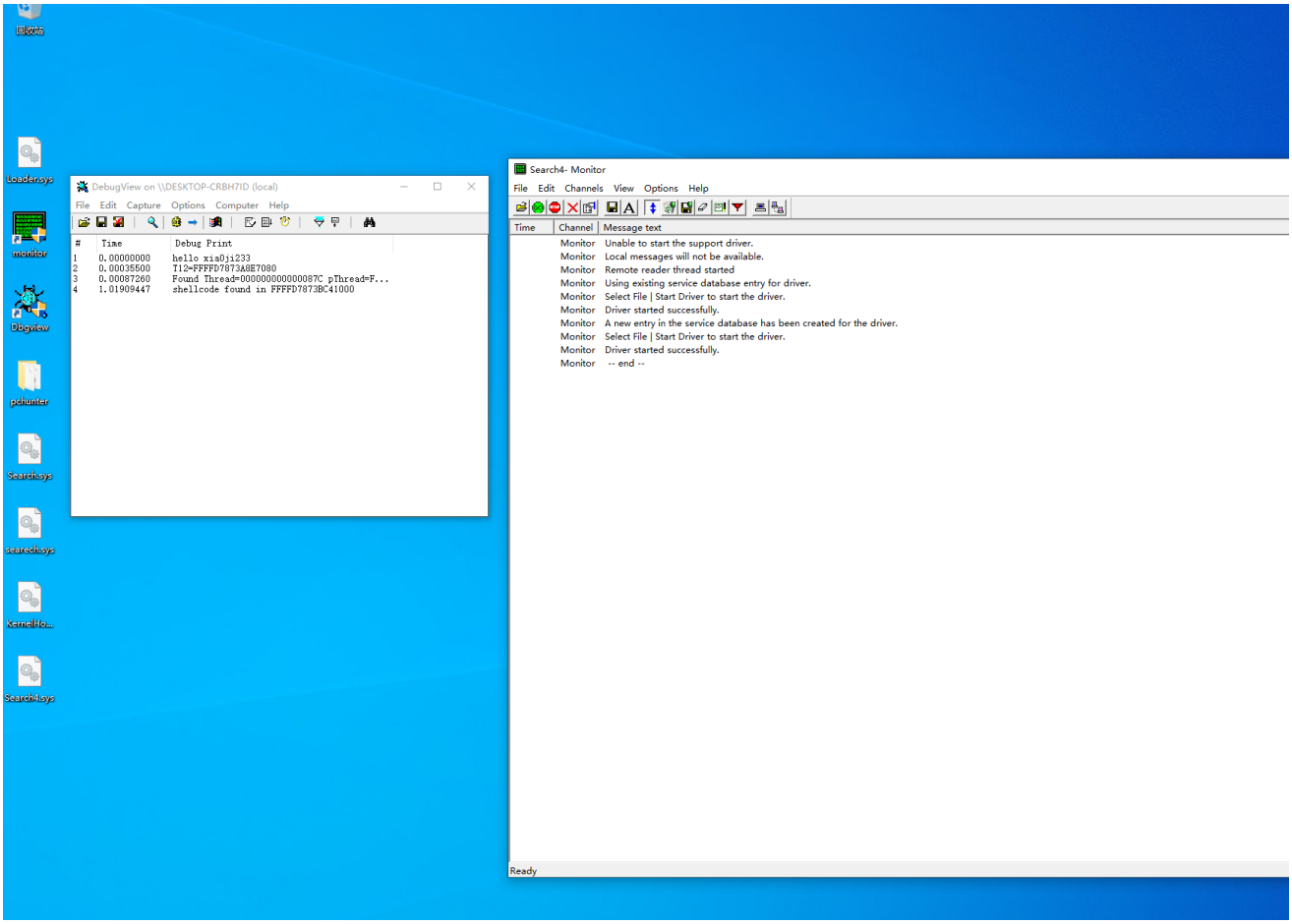
观察到 ETHTREAD 结构体中有个指针指向了距离shellcode比较近的位置



获取这个指针的前八位，然后爆破，匹配特征码。

```
#include<ntifs.h>
#include <ntddk.h>
#include <ntstrsafe.h>

#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)
#define MAX_BACKTRACE_DEPTH 20
ULONG64 num = 0;
NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(ULONG, PVOID, ULONG, PULONG);

VOID DRIVERUNLOAD(_In_ struct _DRIVER_OBJECT* DriverObject)
{
    kprintf(("unload\n"));

}


NTSTATUS EnumerateKernelThreads() {


    PETHREAD T12 = NULL;
    PETHREAD T;
    PsLookupThreadByThreadId(12, &T12);
    kprintf(("T12=%p\n"), T12);
    ULONG64 Start = *(ULONG64 *)((ULONG64)T12 + 0x620);
    HANDLE TargetThread = 0;
    for (int i = 16; i < 0x20000; i+=4) {
        T = NULL;
        PsLookupThreadByThreadId(i, &T);
        if (T) {
```

```
            ULONG64 Startaddr = *(ULONG64 *)((ULONG64)T + 0x620);
            if (Startaddr == Start) {
                kprintf(("Found Thread=%p pThread=%p\n"), i,T);

                TargetThread = i;
                break;
            }

        }
    }


    if (T != NULL) {
        ULONG64 ExitTime = NULL;
        while (!ExitTime) {
            ExitTime=*(ULONG64 *)((ULONG64)T + 0x608);
            LARGE_INTEGER inTime;
            inTime.QuadPart = 1000 * -10000;
            KeDelayExecutionThread(KernelMode, FALSE, &inTime);
        }

        ULONG64 address;
        for (ULONG64 i = 0x0; i <65536; i++)
        {
            address = ExitTime & 0xffffffff00000000;
            address = address + 0x1000;
            address = address + (i<<16);
            if (MmIsAddressValid((PVOID)address) && *((ULONG64 *)address) == 0x7C8B483024748B48
)
            {
                kprintf(("shellcode found in %p\n"), address);
                //DbgBreakPoint();
                break;
            }

        }

    }


    return STATUS_SUCCESS;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING pReg)
{

    kprintf(("hello xia0ji233\n"));

    EnumerateKernelThreads();
    pDriver->DriverUnload = DRIVERUNLOAD;
    return STATUS_SUCCESS;
}
```



先运行Loader.sys，再运行 Search4.sys，即可获得shellcode的地址。

方法5

挂起线程，此时会将线程上下文保存在栈顶中，再去遍历一遍栈，获得RIP指针，这里判断只需要拿 RSP 即可，当[addr+0x180]-0x400==addr（+0x180是RSP相对于上下文结构体的偏移，0x400是context上下文大小）时，取出 RIP 即可。

```
#include<ntifs.h>
#include <ntddk.h>
#include <ntstrsafe.h>
#include <ntimage.h>

#define kprintf(format, ...) DbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, format,
##__VA_ARGS__)
#define MAX_BACKTRACE_DEPTH 20
ULONG64 num = 0;
NTSTATUS EnumerateKernelThreads();

typedef NTSTATUS(NTAPI* PPsSuspendThread)(IN PETHREAD Thread, OUT PULONG PreviousSuspendCount
OPTIONAL);

PPsSuspendThread g_PsSuspendThread = NULL;
typedef struct _KLDR_DATA_TABLE_ENTRY
{
    LIST_ENTRY InLoadOrderLinks;
    PVOID ExceptionTable;
    ULONG ExceptionTableSize;
    PVOID GpValue;
    ULONG UnKnow;
    PVOID DllBase;
```

```
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT __Unused5;
    PVOID SectionPointer;
    ULONG CheckSum;
    PVOID LoadedImports;
    PVOID PatchInformation;
} KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;



VOID DRIVERUNLOAD(_In_ struct _DRIVER_OBJECT* DriverObject)
{
    kprintf(("unload\n"));

}


PVOID SearchOPcode(PDRIVER_OBJECT pObj, PWCHAR DriverName, PCHAR sectionName, PUCHAR opCode, int
len, int offset)
{
    PVOID dllBase = NULL;
    UNICODE_STRING uniDriverName;
    PKLDR_DATA_TABLE_ENTRY firstentry;

    // 获取驱动入口
    PKLDR_DATA_TABLE_ENTRY entry = (PKLDR_DATA_TABLE_ENTRY)pObj->DriverSection;

    firstentry = entry;
    RtlInitUnicodeString(&uniDriverName, DriverName);

    // 开始遍历
    while ((PKLDR_DATA_TABLE_ENTRY)entry->InLoadOrderLinks.Flink != firstentry)
    {
        // 如果找到了所需模块则将其基地址返回
        if (entry->FullDllName.Buffer != 0 && entry->BaseDllName.Buffer != 0)
        {
            if (RtlCompareUnicodeString(&uniDriverName, &(entry->BaseDllName), FALSE) == 0)
            {
                dllBase = entry->DllBase;
                break;
            }
        }
        entry = (PKLDR_DATA_TABLE_ENTRY)entry->InLoadOrderLinks.Flink;
    }

    if (dllBase)
    {
        __try
        {
            // 载入模块基地址
            PIMAGE_DOS_HEADER ImageDosHeader = (PIMAGE_DOS_HEADER)dllBase;
            if (ImageDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
            {
```

```
            return NULL;
        }
        // 得到模块NT头
        PIMAGE_NT_HEADERS64 pImageNtHeaders64 = (PIMAGE_NT_HEADERS64)((PUCHAR)dllBase +
ImageDosHeader->e_lfanew);

        // 获取节表头
        PIMAGE_SECTION_HEADER pSectionHeader = (PIMAGE_SECTION_HEADER)
((PUCHAR)pImageNtHeaders64 + sizeof(pImageNtHeaders64->Signature) + sizeof(pImageNtHeaders64-
>FileHeader) + pImageNtHeaders64->FileHeader.SizeOfOptionalHeader);

        PUCHAR endAddress = 0;
        PUCHAR starAddress = 0;

        // 寻找符合条件的节
        for (int i = 0; i < pImageNtHeaders64->FileHeader.NumberOfSections; i++)
        {
            // 寻找符合条件的表名
            if (memcmp(sectionName, pSectionHeader->Name, strlen(sectionName) + 1) == 0)
            {
                // 取出开始和结束地址
                starAddress = pSectionHeader->VirtualAddress + (PUCHAR)dllBase;
                endAddress = pSectionHeader->VirtualAddress + (PUCHAR)dllBase +
pSectionHeader->SizeOfRawData;
                break;
            }
            // 遍历下一个节
            pSectionHeader++;
        }
        if (endAddress && starAddress)
        {
            // 找到会开始寻找特征
            for (; starAddress < endAddress - len - 1; starAddress++)
            {
                // 验证访问权限
                if (MmIsAddressValid(starAddress))
                {
                    int i = 0;
                    for (; i < len; i++)
                    {
                        // 判断是否为通配符'*'
                        if (opCode[i] == 0x2a)
                            continue;

                        // 找到了一个字节则跳出
                        if (opCode[i] != starAddress[i])
                            break;
                    }
                    // 找到次数完全匹配则返回地址
                    if (i == len)
                    {
                        return starAddress + offset;
                    }
                }
            }
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {}
```

```
    }

    return NULL;
}


NTSTATUS EnumerateKernelThreads() {


    PETHREAD T12 = NULL;
    PETHREAD T;
    PsLookupThreadByThreadId(12, &T12);
    kprintf(("T12=%p\n"), T12);
    ULONG64 Start = *(ULONG64 *)((ULONG64)T12 + 0x620);
    HANDLE TargetThread = 0;
    for (int i = 16; i < 0x20000; i+=4) {
        T = NULL;
        PsLookupThreadByThreadId(i, &T);
        if (T) {
            ULONG64 Startaddr = *(ULONG64 *)((ULONG64)T + 0x620);
            if (Startaddr == Start) {
                kprintf(("Found Thread=%p pThread=%p\n"), i,T);

                TargetThread = i;
                break;
            }

        }
    }
    if (T) {
        ULONG64 StackBase = 0;
        ULONG64 StackLimit = 0;
        StackBase= *(ULONG64 *)((ULONG64)T + 0x38);
        StackLimit=*(ULONG64 *)((ULONG64)T + 0x30);
        kprintf(("Stackbase=%p StackLimit=%p\n"), StackBase, StackLimit);
        PVOID threadObj = NULL;
        NTSTATUS state = ObReferenceObjectByHandle(TargetThread, 0x1FFFFF, *PsThreadType,
KernelMode, &threadObj, NULL);
        g_PsSuspendThread(threadObj, NULL);
        for (ULONG64 addr = StackBase - 0x10 - sizeof(KTRAP_FRAME); addr > StackLimit; addr -=
8) {
            KTRAP_FRAME* ptr = addr;
            if (ptr->Rsp == addr) {
                kprintf(("shellcode found in %p\n"), ptr->Rip);
                break;
            }
        }

    }
    return STATUS_SUCCESS;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING pReg)
{

    kprintf(("hello xia0ji233\n"));
    UCHAR SuspendOpCode[] = { 0x48, 0x8b, 0xf9, 0x83, 0x64, 0x24, 0x20, 0x00, 0x65, 0x48, 0x8b,
0x34, 0x25, 0x88, 0x01 };
```

```
    g_PsSuspendThread = (PPsSuspendThread)SearchOPcode(pDriver, L"ntoskrnl.exe", "PAGE",
SuspendOpCode, sizeof(SuspendOpCode), -24);

    EnumerateKernelThreads();
    pDriver->DriverUnload = DRIVERUNLOAD;
    return STATUS_SUCCESS;
}
```

先运行 Loader.sys，在运行search5.sys，即可输出shellcode。