首先CR3是什么，CR3是一个寄存器，该寄存器内保存有页目录表物理地址(PDBR地址)，其实CR3内部存放的就是页目录表的内存基地址，运用CR3切换可实现对特定进程内存地址的强制读写操作，此类读写属于有痕读写，多数驱动保护都会将这个地址改为无效，此时CR3读写就失效了，当然如果能找到CR3的正确地址，此方式也是靠谱的一种读写机制。

在读写进程之前需要先找到进程的`PEPROCESS`结构，查找结构的方法也很简单，依次遍历进程并对比进程名称即可得到。

```
#include <ntifs.h>
#include <windef.h>
#include <intrin.h>

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS
*Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);

// 定义全局EProcess结构
PEPROCESS Global_Peprocess = NULL;

// 根据进程名获得EPROCESS结构
NTSTATUS GetProcessObjectByName(char *name)
{
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    SIZE_T i;

    __try
    {
        for (i = 100; i<20000; i += 4)
        {
            NTSTATUS st;
            PEPROCESS ep;
            st = PsLookupProcessByProcessId((HANDLE)i, &ep);
            if (NT_SUCCESS(st))
            {
                char *pn = PsGetProcessImageFileName(ep);
                if (_stricmp(pn, name) == 0)
                {
                    Global_Peprocess = ep;
                }
            }
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        return Status;
    }
    return Status;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}
```

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

    NTSTATUS nt = GetProcessObjectByName("Tutorial-i386.exe");

    if (NT_SUCCESS(nt))
    {
        DbgPrint("[+] eprocess = %x \n", Global_Peprocess);
    }

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

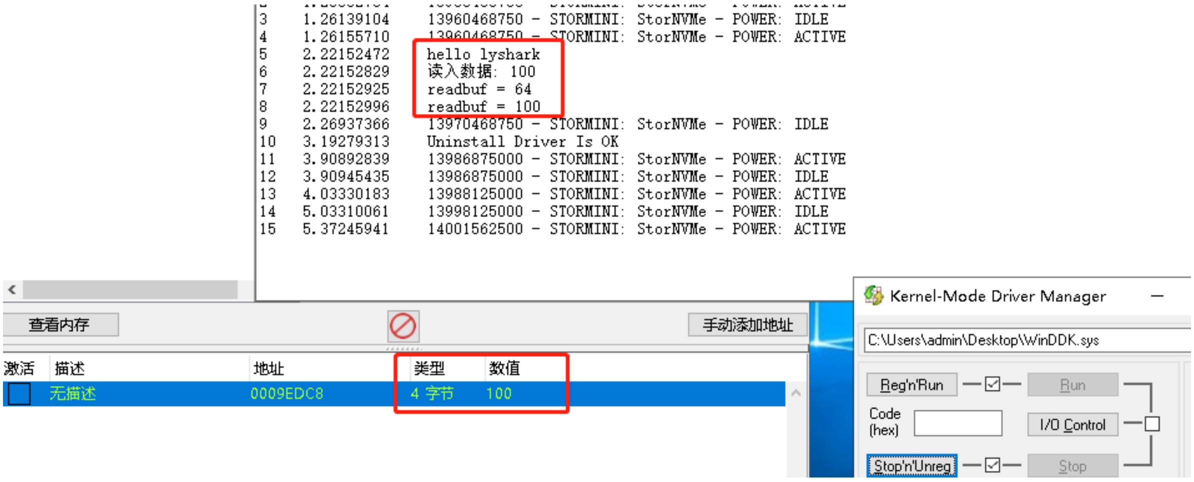
以打开`Tutorial-i386.exe`为例，打开后即可返回他的`Proces`，当然也可以直接传入进程PID同样可以得到进程`Process`结构地址。

```
// 根据PID打开进程
PEPROCESS Peprocess = NULL;
DWORD PID = 6672;
NTSTATUS nt = PsLookupProcessByProcessId((HANDLE)PID, &Peprocess);
```

通过CR3读取内存实现代码如下，我们读取`Tutorial-i386.exe`里面的`0x0009EDC8`这段内存，读出长度是4字节，代码如下。

```
#include <ntifs.h>
#include <windef.h>
#include <intrin.h>

#define DIRECTORY_TABLE_BASE 0x028

#pragma  intrinsic(_disable)
#pragma  intrinsic(_enable)

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);

// 关闭写保护
KIRQL Open()
{
    KIRQL irql = KeRaiseIrqlToDpcLevel();
    UINT64 cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}

// 开启写保护
void Close(KIRQL irql)
{
    UINT64 cr0 = __readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

// 检查内存
ULONG64 CheckAddressVal(PVOID p)
{
    if (MmIsAddressValid(p) == FALSE)
        return 0;
    return *(PULONG64)p;
}

// CR3 寄存器读内存
BOOLEAN CR3_ReadProcessMemory(IN PEPROCESS Process, IN PVOID Address, IN UINT32
Length, OUT PVOID Buffer)
{
    ULONG64 pDTB = 0, OldCr3 = 0, vAddr = 0;
    pDTB = CheckAddressVal((UCHAR*)Process + DIRECTORY_TABLE_BASE);
    if (pDTB == 0)
    {
        return FALSE;
    }

    _disable();
    OldCr3 = __readcr3();
    __writecr3(pDTB);
    _enable();

    if (MmIsAddressValid(Address))
    {
        RtlCopyMemory(Buffer, Address, Length);
        DbgPrint("读入数据: %ld", *(PDWORD)Buffer);
        return TRUE;
    }

    _disable();
    __writecr3(OldCr3);
    _enable();
    return FALSE;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}

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

    // 根据PID打开进程
    PEPROCESS Peprocess = NULL;
    DWORD PID = 6672;
    NTSTATUS nt = PsLookupProcessByProcessId((HANDLE)PID, &Peprocess);

    DWORD buffer = 0;

    BOOLEAN bl = CR3_ReadProcessMemory(Peprocess, (PVOID)0x0009EDC8, 4,
&buffer);

    DbgPrint("readbuf = %x \n", buffer);
    DbgPrint("readbuf = %d \n", buffer);

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

读出后输出效果如下：



```
3     1.26139104     13960468750 - STORMINI: StorNVMe - POWER: IDLE
4     1.26155710     13960468750 - STORMINI: StorNVMe - POWER: ACTIVE
5     2.22152472     hello lyshark
6     2.22152829     读入数据: 100
7     2.22152925     readbuf = 64
8     2.22152996     readbuf = 100
9     2.26937366     13970468750 - STORMINI: StorNVMe - POWER: IDLE
10    3.19279313     Uninstall Driver Is OK
11    3.90892839     13986875000 - STORMINI: StorNVMe - POWER: ACTIVE
12    3.90945435     13986875000 - STORMINI: StorNVMe - POWER: IDLE
13    4.03330183     13988125000 - STORMINI: StorNVMe - POWER: ACTIVE
14    5.03310061     13998125000 - STORMINI: StorNVMe - POWER: IDLE
15    5.37245941     14001562500 - STORMINI: StorNVMe - POWER: ACTIVE
```

激活	描述	地址	类型	数值
□	无描述	0009EDC8	4 字节	100

C:\Users\admin\Desktop\WinDDK.sys

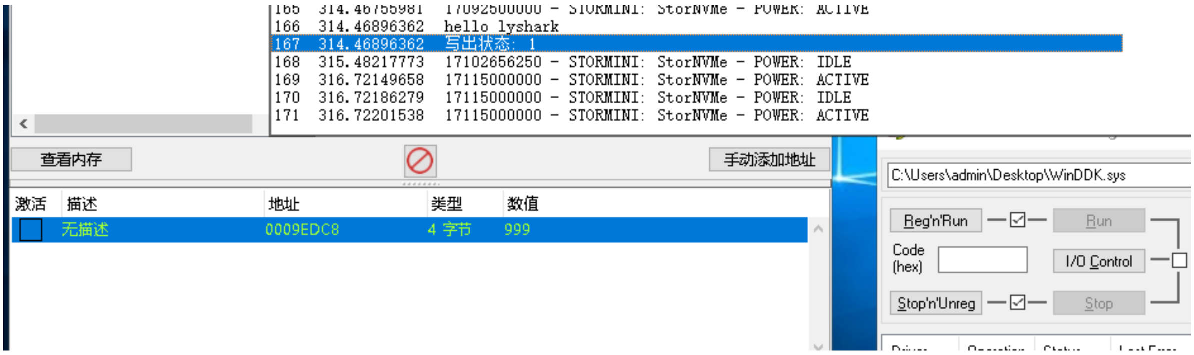
写出内存与读取基本一致，代码如下。

```
#include <ntifs.h>
#include <windef.h>
#include <intrin.h>

#define DIRECTORY_TABLE_BASE 0x028

#pragma  intrinsic(_disable)
#pragma  intrinsic(_enable)

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS
*Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);

// 关闭写保护
KIRQL Open()
{
    KIRQL irql = KeRaiseIrqlToDpcLevel();
    UINT64 cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}
```

```
// 开启写保护
void Close(KIRQL irql)
{
    UINT64 cr0 = __readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

// 检查内存
ULONG64 CheckAddressVal(PVOID p)
{
    if (MmIsAddressValid(p) == FALSE)
        return 0;
    return *(PULONG64)p;
}

// CR3 寄存器写内存
BOOLEAN CR3_WriteProcessMemory(IN PEPROCESS Process, IN PVOID Address, IN UINT32
Length, IN PVOID Buffer)
{
    ULONG64 pDTB = 0, OldCr3 = 0, vAddr = 0;

    // 检查内存
    pDTB = CheckAddressVal((UCHAR*)Process + DIRECTORY_TABLE_BASE);
    if (pDTB == 0)
    {
        return FALSE;
    }

    _disable();

    // 读取CR3
    OldCr3 = __readcr3();

    // 写CR3
    __writecr3(pDTB);
    _enable();

    // 验证并拷贝内存
    if (MmIsAddressValid(Address))
    {
        RtlCopyMemory(Address, Buffer, Length);
        return TRUE;
    }
    _disable();

    // 恢复CR3
    __writecr3(OldCr3);
    _enable();
    return FALSE;
}
```

```
VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}

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

    // 根据PID打开进程
    PEPROCESS Peprocess = NULL;
    DWORD PID = 6672;
    NTSTATUS nt = PsLookupProcessByProcessId((HANDLE)PID, &Peprocess);

    DWORD buffer = 999;

    BOOLEAN bl = CR3_WriteProcessMemory(Peprocess, (PVOID)0x0009EDC8, 4,
&buffer);
    DbgPrint("写出状态: %d \n", bl);

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

写出后效果如下：



```
165   314.46755981   17092500000 - STORMINI: StorNVMe - POWER: ACTIVE
166   314.46896362   hello lyshark
167   314.46896362   写出状态: 1
168   315.48217773   17102656250 - STORMINI: StorNVMe - POWER: IDLE
169   316.72149658   17115000000 - STORMINI: StorNVMe - POWER: ACTIVE
170   316.72186279   17115000000 - STORMINI: StorNVMe - POWER: IDLE
171   316.72201538   17115000000 - STORMINI: StorNVMe - POWER: ACTIVE
```

激活	描述	地址	类型	数值
□	无描述	0009EDC8	4 字节	999

C:\Users\admin\Desktop\WinDDK.sys

至于进程将CR3改掉了读取不到该寄存器该如何处理，这里我找到了一段参考代码，可以实现寻找CR3地址这个功能。

```
#include <ntddk.h>
#include <ntstrsafe.h>
#include <windef.h>
#include <intrin.h>

#pragma pack(push, 1)

typedef struct _IDTR // IDT基址
{
    USHORT limit;    // 范围 占8位
    ULONG64 base;    // 基地址 占32位 _IDT_ENTRY类型指针
}IDTR, *PIDTR;

typedef union _IDT_ENTRY
{
    struct kidt
    {
        USHORT OffsetLow;
        USHORT Selector;
        USHORT IstIndex : 3;
        USHORT Reserved0 : 5;
        USHORT Type : 5;
        USHORT Dpl : 2;
        USHORT Present : 1;
        USHORT OffsetMiddle;
        ULONG OffsetHigh;
        ULONG Reserved1;
    }idt;
    UINT64 Alignment;
} IDT_ENTRY, *PIDT_ENTRY;

#pragma pack(pop)

// 输出调试内容
void DebugPrint(const char* fmt, ...)
{
    UNREFERENCED_PARAMETER(fmt);
    va_list ap;
    va_start(ap, fmt);
    vDbgPrintEx(DPFLTR_IHVDRIVER_ID, DPFLTR_ERROR_LEVEL, fmt, ap);
    va_end(ap);
    return;
}

// 获取IDT表地址
ULONG64 GetIdtAddr(ULONG64 pIdtBaseAddr, UCHAR pIndex)
{
    PIDT_ENTRY Pidt_info = (PIDT_ENTRY)(pIdtBaseAddr);
    Pidt_info += pIndex;
    ULONG64 vCurrentAddr = 0;
    ULONG64 vCurrentHighAddr = 0;
    vCurrentAddr = Pidt_info->idt.OffsetMiddle;
    vCurrentAddr = vCurrentAddr << 16;
    vCurrentAddr += Pidt_info->idt.OffsetLow;

    vCurrentHighAddr = Pidt_info->idt.OffsetHigh;
    vCurrentHighAddr = vCurrentHighAddr << 32;
    vCurrentAddr += vCurrentHighAddr;
    return vCurrentAddr;
}

VOID UnLoadDriver()
{

}

NTSTATUS DriverEntry(_In_ PDRIVER_OBJECT pPDriverObj, _In_ PUNICODE_STRING
pRegistryPath)
{
    UNREFERENCED_PARAMETER(pRegistryPath);
    pPDriverObj->DriverUnload = (PDRIVER_UNLOAD)UnLoadDriver;
    /**
    TP版KiPageFault
    fffff880`09f54000 50              push    rax
    // 这里实际上是真实处理函数的地址  需要 & 0xFFFFFFFFFFF00000
    fffff880`09f54001 48b87830ce0980f8ffff mov rax,0FFFFF88009CE3078h
    fffff880`09f5400b 4883ec08        sub     rsp,8
    fffff880`09f5400f 48890424        mov     qword ptr [rsp],rax
    fffff880`09f54013 48311424        xor     qword ptr [rsp],rdx
    fffff880`09f54017 e810000000      call    fffff880`09f5402c
    fffff880`09f5401c 896eff          mov     dword ptr [rsi-1],ebp
    fffff880`09f5401f 230500000089    and     eax,dword ptr [fffff87f`92f54025]
    **/
    //得到TP KiPageFault地址
    // _IDTR vContent;
    // __sidt(&vContent);
    ULONG64 vTpKiPageFault = GetIdtAddr(vContent.base, 0xE);

    //得到TP 动态内存起始值
    ULONG64 vTpMemory = *(PULONG64)(vTpKiPageFault + 0x3) & 0xFFFFFFFFFFF00000;

    //得到TP KiPageFault真实处理函数
    ULONG64 vTpKiPageFaultFuncAddr = vTpMemory + 0x4CE7C;

    if (MmIsAddressValid((PVOID)vTpKiPageFaultFuncAddr))
    {//真实处理函数有效

        //得到TP数据对象基地址
        ULONG64 vTpDataObjectBase = *(PULONG)(vTpMemory + 0x1738B) + vTpMemory + 0x1738F;

        if (MmIsAddressValid((PVOID)vTpDataObjectBase))
        {//基地址有效

            //得到TP 用来保存真实CR3 保存当前所属进程ID 的对象
            ULONG64 vTpDataObject = *(PULONG64)vTpDataObjectBase;

            DebugPrint("数据对象:0x%016llx, 真实CR3:0x%016llx, 所属进程ID:%d\n", vTpDataObject, *(PULONG64)(vTpDataObject + 0x70), *(PULONG)(vTpDataObject + 0x18));
        }
        else
            DebugPrint("vTpDataObjectBase无法读取:0x%016llx\n", vTpDataObjectBase);
    }
    else
        DebugPrint("vTpKiPageFaultFuncAddr无法读取:0x%016llx\n", vTpKiPageFaultFuncAddr);

    return STATUS_SUCCESS;
}
```

> 本书作者： 王瑞 (LyShark)
>
> 作者邮箱： me@lyshark.com
>
> 作者博客： https://lyshark.cnblogs.com
>
> 团队首页： www.lyshark.com