# 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;

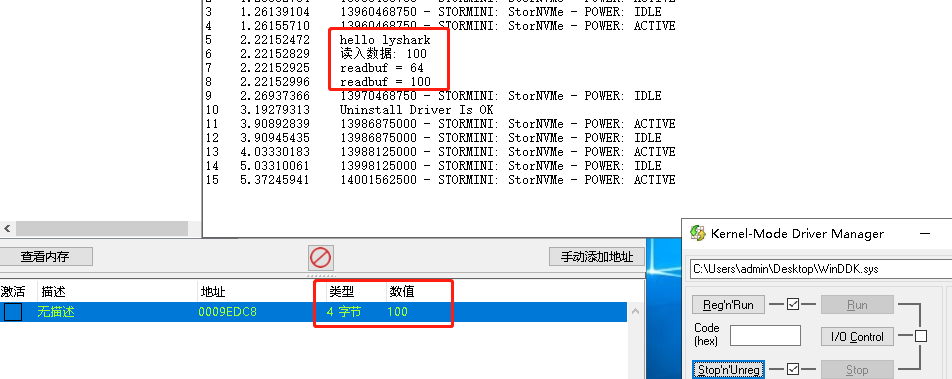
BOOLEAN bl = CR3\_ReadProcessMemory(Peprocess, (PVOID)0x0009EDC8, 4, &buffer);

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：



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;

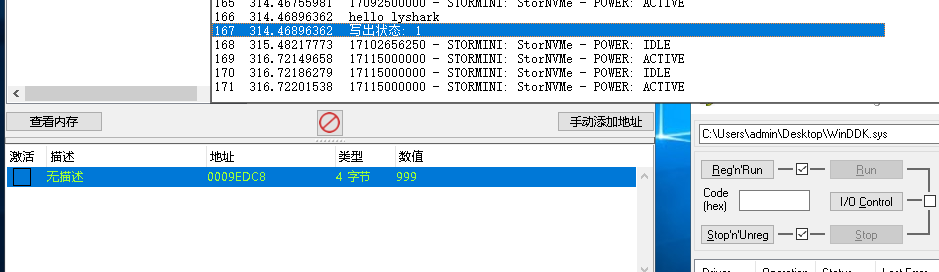
BOOLEAN bl = CR3\_WriteProcessMemory(Peprocess, (PVOID)0x0009EDC8, 4, &buffer);

DbgPrint("写出状态: %d \n", bl);

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 写出后效果如下：



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： m [e@lyshark.com](mailto:e@lyshark.com)

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com