MDL内存读写是最常用的一种读写模式，通常需要附加到指定进程空间内然后调用内存拷贝得到对端内存中的数据，在调用结束后再将其空间释放掉，通过这种方式实现内存读写操作，此种模式的读写操作也是最推荐使用的相比于CR3切换来说，此方式更稳定并不会受寄存器的影响。

MDL读取内存步骤

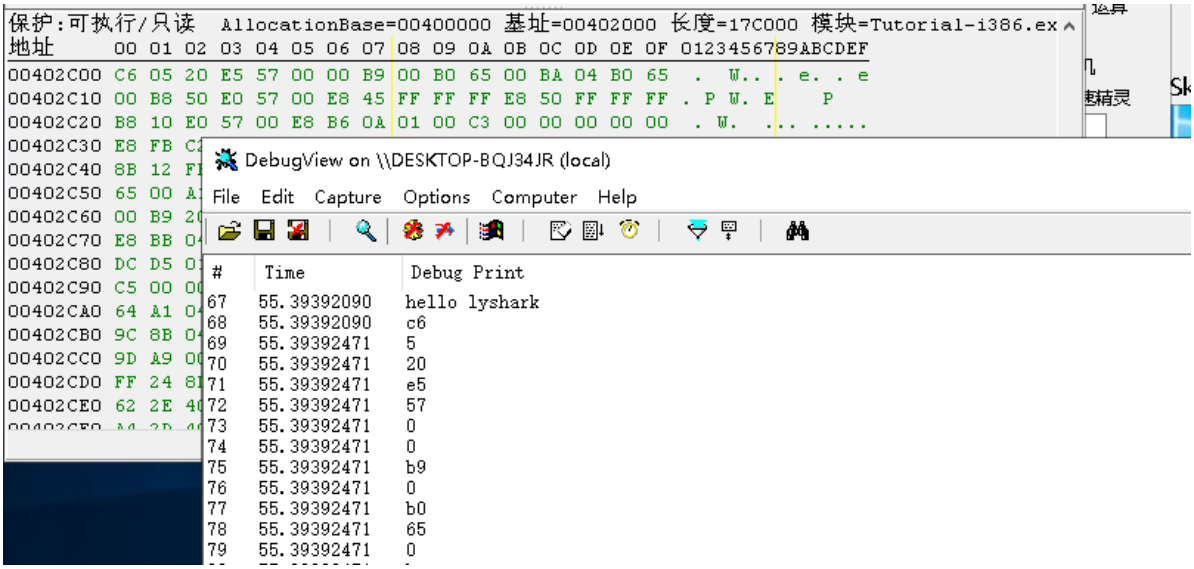
- 1.调用PsLookupProcessByProcessId得到进程Process结构
- 2.调用KeStackAttachProcess附加到对端进程内
- 3.调用ProbeForRead检查内存是否可读写
- 4.拷贝内存空间中的数据到自己的缓冲区内
- 5.调用KeUnstackDetachProcess接触绑定
- 6.调用ObDereferenceObject使对象引用数减1

代码总结起来应该是如下样子，用户传入一个结构体，输出对应长度的字节数据：

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

typedef struct
{
    DWORD pid;                  // 要读写的进程ID
    DWORD64 address;            // 要读写的地址
    DWORD size;                 // 读写长度
    BYTE* data;                 // 要读写的数据
}ReadMemoryStruct;

// MDL读内存
BOOL MDLReadMemory(ReadMemoryStruct* data)
{
    BOOL bRet = TRUE;
    PEPROCESS process = NULL;

    PsLookupProcessByProcessId(data->pid, &process);

    if (process == NULL)
    {
        return FALSE;
    }

    BYTE* GetData;
    __try
    {
        GetData = ExAllocatePool(PagedPool, data->size);
    }
    __except (1)
    {
        return FALSE;
    }

    KAPC_STATE stack = { 0 };
    KeStackAttachProcess(process, &stack);

    __try
```

```
    {
        ProbeForRead(data->address, data->size, 1);
        RtlCopyMemory(GetData, data->address, data->size);
    }
    __except (1)
    {
        bRet = FALSE;
    }

    ObDereferenceObject(process);
    KeUnstackDetachProcess(&stack);
    RtlCopyMemory(data->data, GetData, data->size);
    ExFreePool(GetData);
    return bRet;
}

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

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

    ReadMemoryStruct ptr;

    ptr.pid = 6672;
    ptr.address = 0x402c00;
    ptr.size = 100;

    // 分配空间接收数据
    ptr.data = ExAllocatePool(PagedPool, ptr.size);

    // 读内存
    MDLReadMemory(&ptr);

    // 输出数据
    for (size_t i = 0; i < 100; i++)
    {
        DbgPrint("%x \n", ptr.data[i]);
    }

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

读取内存地址`0x402c00`效果如下所示：



MDL写入内存步骤

- 1.调用PsLookupProcessByProcessId得到进程Process结构
- 2.调用KeStackAttachProcess附加到对端进程内
- 3.调用ProbeForRead检查内存是否可读写
- 4.拷贝内存空间中的数据到自己的缓冲区内
- 5.调用MmMapLockedPages锁定当前内存页面(写入)
- 6.调用RtlCopyMemory内存拷贝完成写入(写入)
- 7.调用IoFreeMdl释放MDL锁(写入)
- 8.调用KeUnstackDetachProcess接触绑定
- 9.调用ObDereferenceObject使对象引用数减1

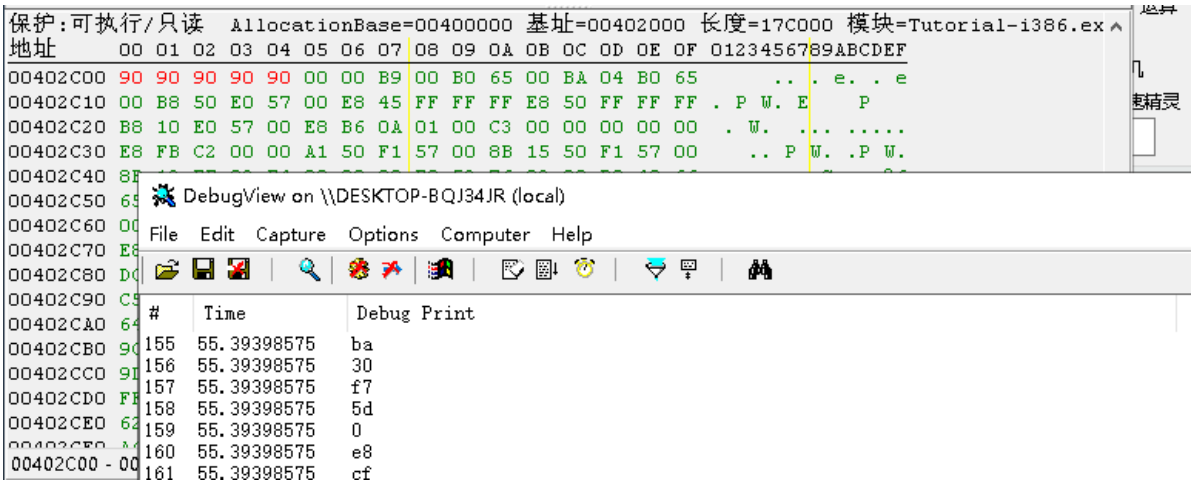
写入时与读取类似，只是多了锁定页面和解锁操作。

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

typedef struct
{
    DWORD pid;                  // 要读写的进程ID
    DWORD64 address;            // 要读写的地址
    DWORD size;                 // 读写长度
    BYTE* data;                 // 要读写的数据
}ReadMemoryStruct;

// MDL写内存
BOOL MDLWriteMemory(ReadMemoryStruct* data)
{
    BOOL bRet = TRUE;
    PEPROCESS process = NULL;

    PsLookupProcessByProcessId(data->pid, &process);
    if (process == NULL)
    {
        return FALSE;
    }

    BYTE* GetData;
    __try
```

```
    {
        GetData = ExAllocatePool(PagedPool, data->size);
    }
    __except (1)
    {
        return FALSE;
    }

    for (int i = 0; i < data->size; i++)
    {
        GetData[i] = data->data[i];
    }

    KAPC_STATE stack = { 0 };
    KeStackAttachProcess(process, &stack);

    PMDL mdl = IoAllocateMdl(data->address, data->size, 0, 0, NULL);
    if (mdl == NULL)
    {
        return FALSE;
    }

    MmBuildMdlForNonPagedPool(mdl);

    BYTE* ChangeData = NULL;

    __try
    {
        ChangeData = MmMapLockedPages(mdl, KernelMode);
        RtlCopyMemory(ChangeData, GetData, data->size);
    }
    __except (1)
    {
        bRet = FALSE;
        goto END;
    }

END:
    IoFreeMdl(mdl);
    ExFreePool(GetData);
    KeUnstackDetachProcess(&stack);
    ObDereferenceObject(process);

    return bRet;
}

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

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

```
    ReadMemoryStruct ptr;

    ptr.pid = 6672;
    ptr.address = 0x402c00;
    ptr.size = 5;

    // 需要写入的数据
    ptr.data = ExAllocatePool(PagedPool, ptr.size);

    // 循环设置
    for (size_t i = 0; i < 5; i++)
    {
        ptr.data[i] = 0x90;
    }

    // 写内存
    MDLWriteMemory(&ptr);

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

写出效果如下:



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