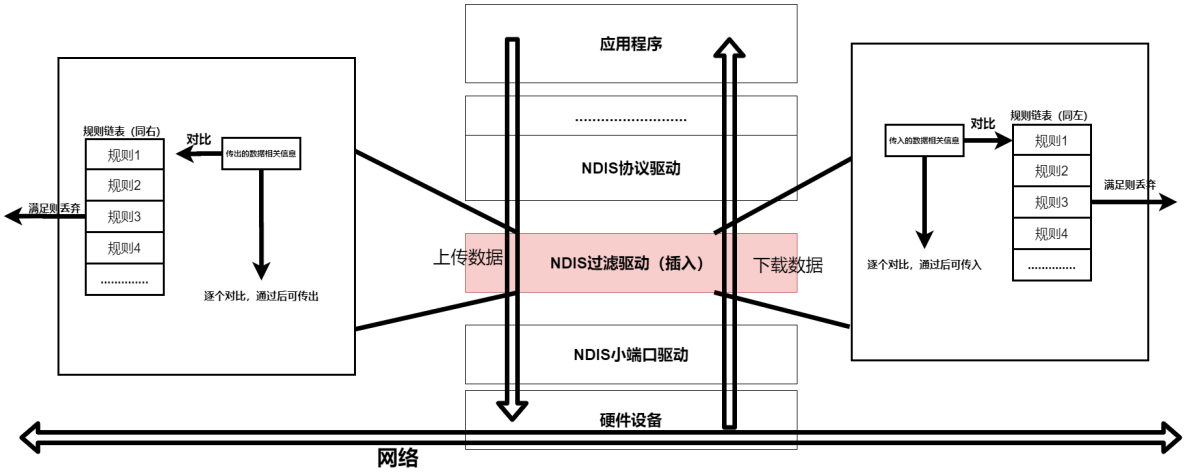
# NDIS Driver

## 测试编译环境以及结构：

| Windows版本 | WDK版本 |
| --- | --- |
| Win 10 x64 1909 | 10.0.16299.15 |

**基本实现原理：**

红色部分即为我们编译的驱动程序




对比处的逻辑详情可查看源代码

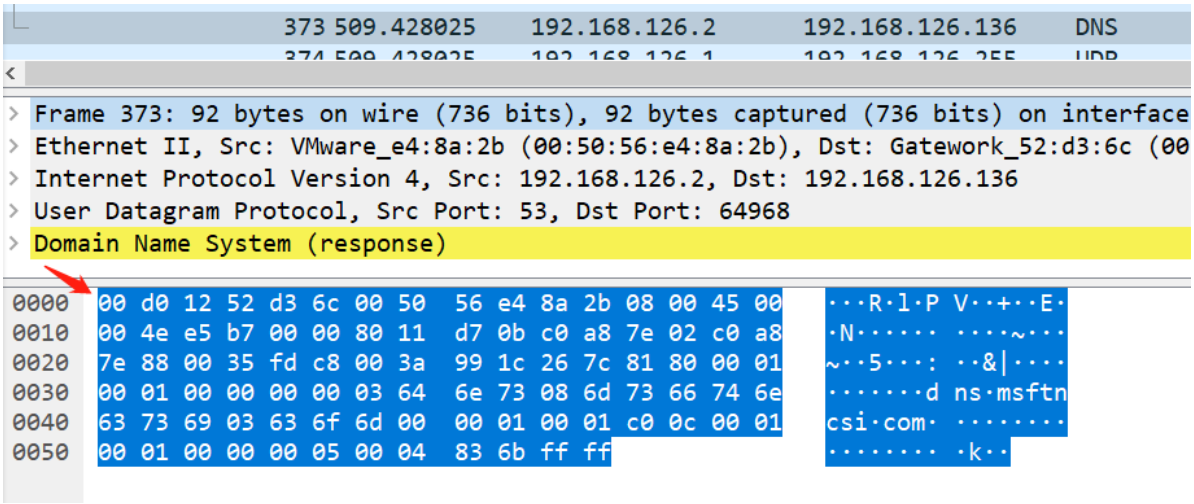
### 相关结构与函数说明

**数据帧获取函数：**

```
//通过NET_BUFFER获取一个网络封包数据,此PPacket需要释放，（组包函数）
PPacket IC_Get_Packer(PNET_BUFFER nb);
```

PPacket是数据帧的空间首地址。

一个数据帧如图所示（利用wireshark展示）




红色箭头所指处地址即为PPacket的值。

**数据帧相关信息结构：**

```
typedef struct packet_information {
    //此结构可随着功能的增加自定义
    UCHAR V4_Or_V6;//是过滤ipv4还是ipv6  0:ipv4  1:ipv6
    union src_ip
    {
        PUCHAR ipv4;
        PUCHAR ipv6;
    }SRC_IP;//源ip
    union dst_ip
    {
        PUCHAR ipv4;
        PUCHAR ipv6;
    }DST_IP;//目的ip
    USHORT src_port; //源端口  传输层为 UDP、TCP时才为有效值，默认为0
    USHORT dst_port;//目的端口  传输层为 UDP、TCP时才为有效值，默认为0
}Packet_Information, * PPacket_Information;
```

此结构通过下列一系列函数辅助生成。该结构运用于与规则进行对比。

这个结构就是上图“基本实现原理”中的

**数据链路层结构以及获取函数：**

数据链路层仅支持 Ethernet II

```
//路局链路层
typedef struct  ether_header {
    UCHAR   ether_dhost[ETHER_ADDR_LEN];//目的mac
    UCHAR   ether_shost[ETHER_ADDR_LEN];//源mac
    USHORT  ether_type;//网络层协议类型
}TP_ETHERNET, *PTP_ETHERNET;
```

获取函数：

```
//返回数据链路层部分数据
PTP_ETHERNET IC_Get_Data_Link_Layer(PPacket packet)
```

**网络层结构以及获取函数**

网络层仅支持 IPv4 和 IPv6，可以自行增加

```
//网络层类型
typedef struct ipv4 {
    PUCHAR Src_add;//源IP
    PUCHAR Dst_add;//目的IP
    PUCHAR Protocol;//上层协议
    ULONG Length;//ip头长度

}IP_V4,* PIP_V4;

typedef struct ipv6 {
    PUCHAR Src_add;//源IP
    PUCHAR Dst_add;//目的IP
    PUCHAR Protocol;//上层协议
    ULONG Length;//ip头长度
}IP_V6, *PIP_V6;

typedef union network_layer
{
    //这里可以增加类型
    IP_V4 ipv4;
    IP_V6 ipv6;

}Network_Layer;
```

网络层获取函数：

```
//返回网络层部分数据
Network_Layer IC_Get_Network_Layer(PPacket packet, PTP_ETHERNET data_link,
PPacket_Information LSPI)
```

**传输层结构以及获取函数：**

传输层仅支持 TCP和UDP，可以自行增加

```
//传输层类型
//UDP 结构 其中 二字节以上需要 大端序转小端序才是正确值
typedef struct udp_header
{
    USHORT srcport;   // 源端口
    USHORT dstport;   // 目的端口
    USHORT total_len; // 包括UDP报头及UDP数据的长度(单位:字节)
    USHORT chksum;    // 校验和
}TP_UDP, * PTP_UDP;
```

```
//TCP 结构 其中 二字节以上需要 大端序转小端序才是正确值
typedef struct tp_tcp {
    USHORT src_port;    //源端口号
    USHORT dst_port;    //目的端口号
    unsigned int   seq_no;      //序列号
    unsigned int   ack_no;      //确认号

    UCHAR reserved_1 : 4; //保留6位中的4位首部长度
    UCHAR thl : 4;    //tcp头部长度
    UCHAR flag : 6;  //6位标志
    UCHAR reseverd_2 : 2; //保留6位中的2位

    USHORT wnd_size;   //16位窗口大小
    USHORT chk_sum;    //16位TCP检验和
    USHORT urgt_p;     //16为紧急指针
}TP_TCP, * PTP_TCP;

//可添加类型结构
typedef union transport_layer
{
    //这里可以增加类型
    PTP_TCP tcp;
    PTP_UDP udp;

}Transport_Layer;
```

获取函数:

```
//返回传输层部分数据
Transport_Layer IC_Get_Transport_Layer(PPacket packet, PTP_ETHERNET data_link,
Network_Layer network_data, PPacket_Information LSPI)
```

**过滤规则结构:**

```
typedef struct filter_condition {
    //此结构可随着功能的增加自定义，和驱动的相同结构一起改
    UCHAR S_Or_R;//在发送时过滤还是收到时过滤 0：发送 ，1：收到
    UCHAR S_Or_D;//是过滤源还是目的 0：源，1：目的
    UCHAR V4_Or_V6;//是过滤ipv4还是ipv6  0:ipv4  1:ipv6
    union
    {
        UCHAR ipv4[4];
        UCHAR ipv6[16];
    }IP;
    USHORT port;
    int flag;//标识号，删除时要用
    struct filter_condition* prior;
    struct filter_condition* next;
}Filter_Condition, * PFilter_Condition;
```

是一个双向链表。计算机收到数据帧后会与每一个规则进行对比，若满足，则拦截。

此规则通过RING3传入。

**恶意会话相关：**

```
//记录所有会话，仅记录对方的ip
typedef struct record {
    UCHAR v4_or_v6;
    union
    {
        UCHAR ipv4[4];
        UCHAR ipv6[16];
    }IP;
    struct record* next;
}Record,*PRecord;

typedef struct record_list {
    PRecord head;
    PRecord last;
    int num;
}Rocord_List,*PRocord_List;
//benign rocord list
Rocord_List B_Rrd_List = { 0 };//记录温和会话链表（满100个清空）
//malware rocord list
Rocord_List M_Rrd_List = { 0 };//记录恶意会话链表（卸载驱动或重启时清空）
```

会话记录默认开启，每 发送、接收一个数据帧，都会记录。

假设收到的数据帧中的源ip地址不是我们之前发送数据帧的目的地址，则记为恶意会话。（就是别人对本机主动发起的会话）

### 与RING3的交互

该驱动生成的符号链接名为 "NDISLWF"

应用层可通过以下示例与该驱动进行连接：

```
#define DEVICE_NAME L"\\\\.\\NDISLWF" //符号链接名
HANDLE mydevice = CreateFile(DEVICE_NAME, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_SYSTEM, NULL);
```

RING3程序可通过DeviceIoControl函数配合以下宏操作该驱动。

```
#define NEW_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x999,METHOD_BUFFERED,FILE_READ_ACCESS)//增加条件
#define DEL_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x99a,METHOD_BUFFERED,FILE_READ_ACCESS)//删除条件
#define START_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x99b,METHOD_BUFFERED,FILE_READ_ACCESS)//开启
#define STOP_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x99c,METHOD_BUFFERED,FILE_READ_ACCESS)//停止
#define DEL_ALL_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x99e,METHOD_BUFFERED,FILE_READ_ACCESS)//删除所有规则
#define GET_B_FC_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x99f,METHOD_NEITHER,FILE_ANY_ACCESS)//获取恶意会话
#define GET_B_NUM_CODE
CTL_CODE(FILE_DEVICE_UNKNOWN,0x9a0,METHOD_NEITHER,FILE_ANY_ACCESS)//获取恶意会话数
目
```

例如：

```
//开启过滤功能
DWORD lenth = 0;
DeviceIoControl(mydevice, START_FC_CODE, NULL, 0, NULL, 0, &lenth, NULL);
```

```
//添加过滤规则
DWORD lenth = 0;
Filter_Condition Task={0};
.......修改Task内的参数.......
DeviceIoControl(mydevice, NEW_FC_CODE, NULL, 0, &Task, sizeof(Task), &lenth,
NULL);
```

此处的Filter_Condition是应用程序中的结构，和此驱动中的Filter_Condition几乎相同。详细请参考RING3处的说明文档。

具体实现过程请查看源代码。

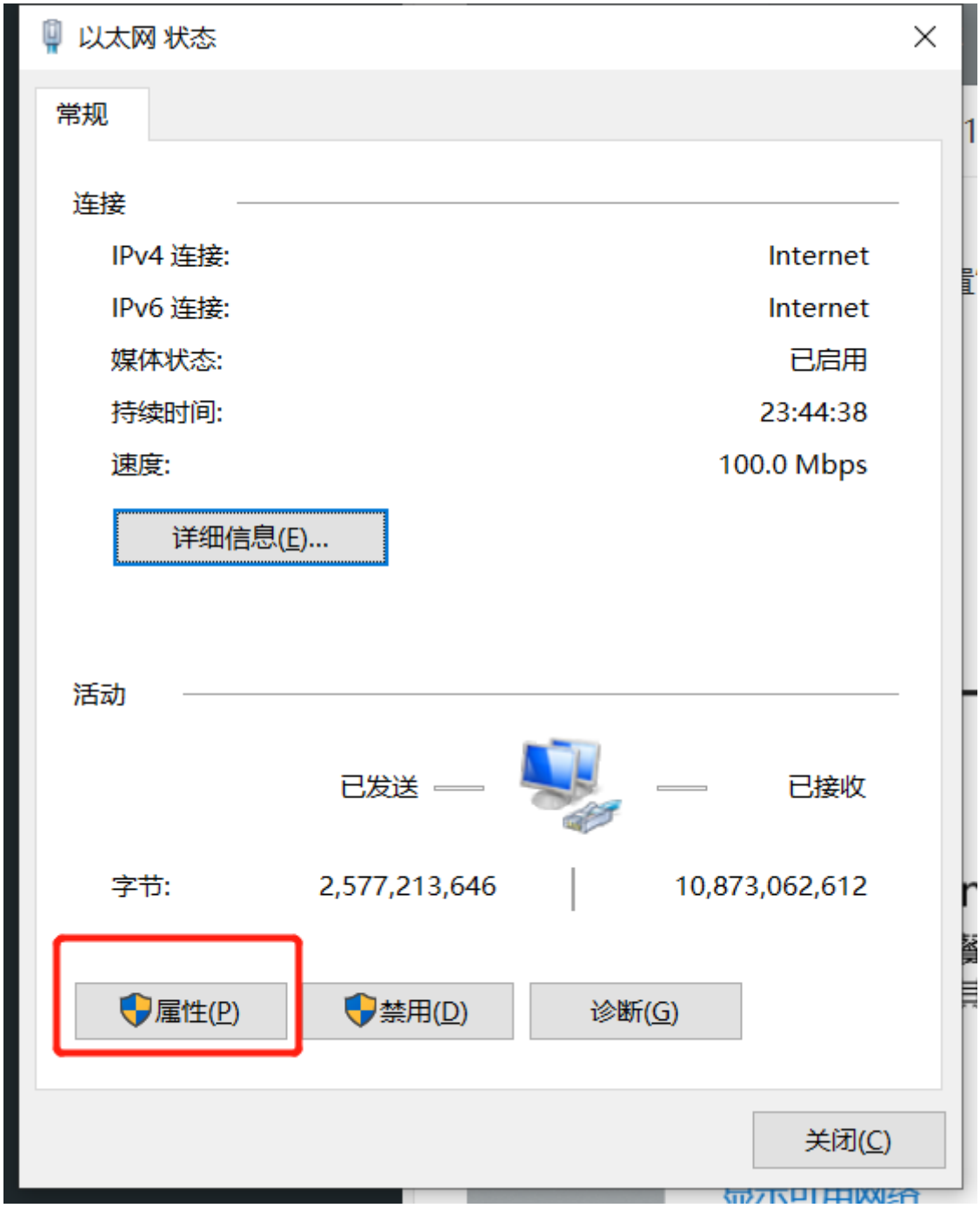
## 安装步骤：

### 1.打开“打开“网络和Internet”设置”

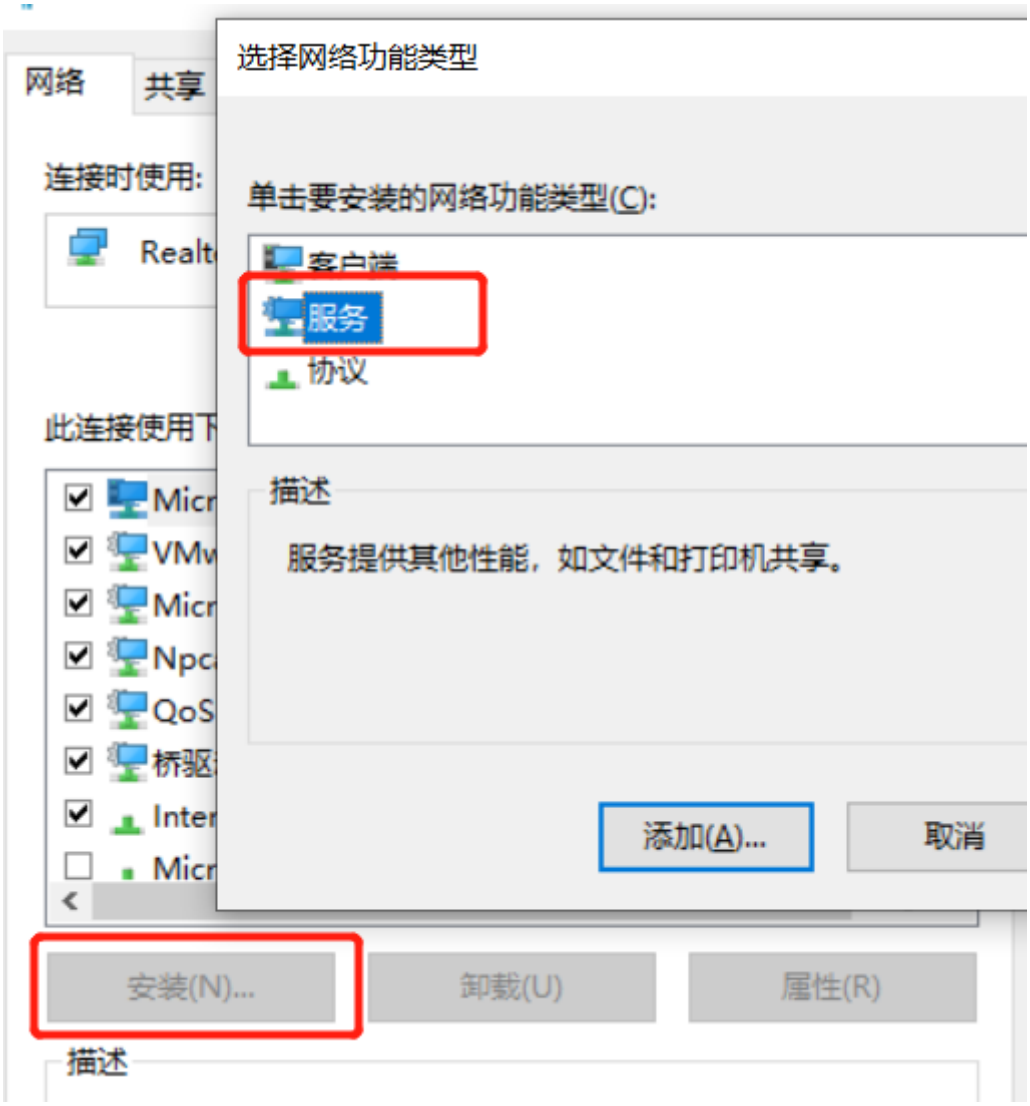

选择“更改适配器选项”

**2.双击当前正在使用的网络**

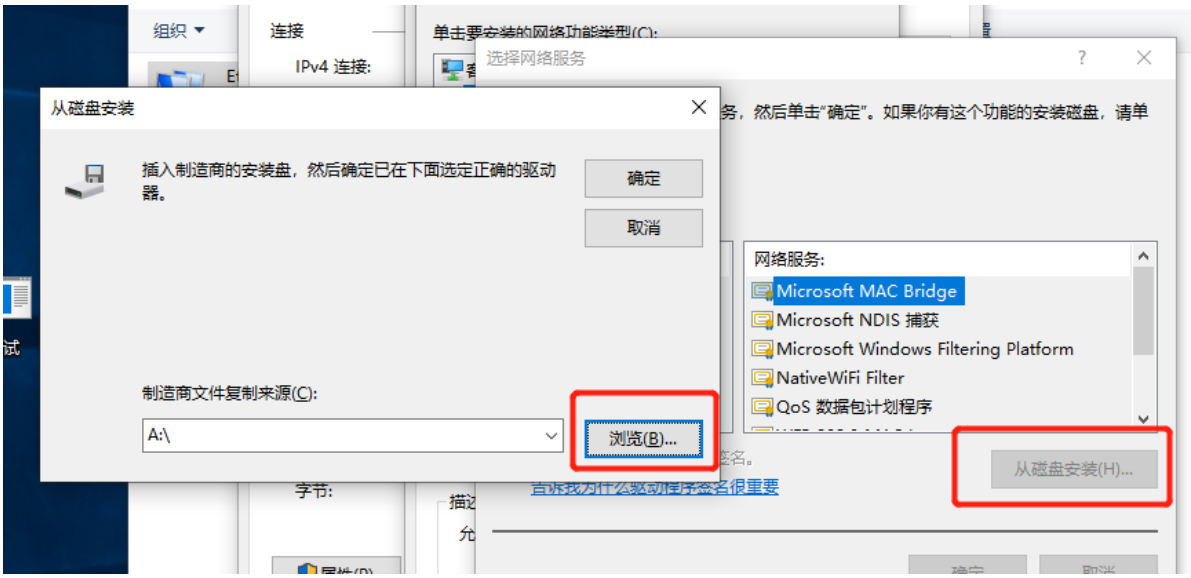
**3.点击属性**


以太网 状态
常规
连接
IPv4 连接:
Internet
IPv6 连接:
Internet
媒体状态:
已启用
持续时间:
23:44:38
速度:
100.0 Mbps
详细信息(E)...
活动
已发送
已接收
字节:
2,577,213,646
10,873,062,612
属性(P)
禁用(D)
诊断(G)
关闭(C)

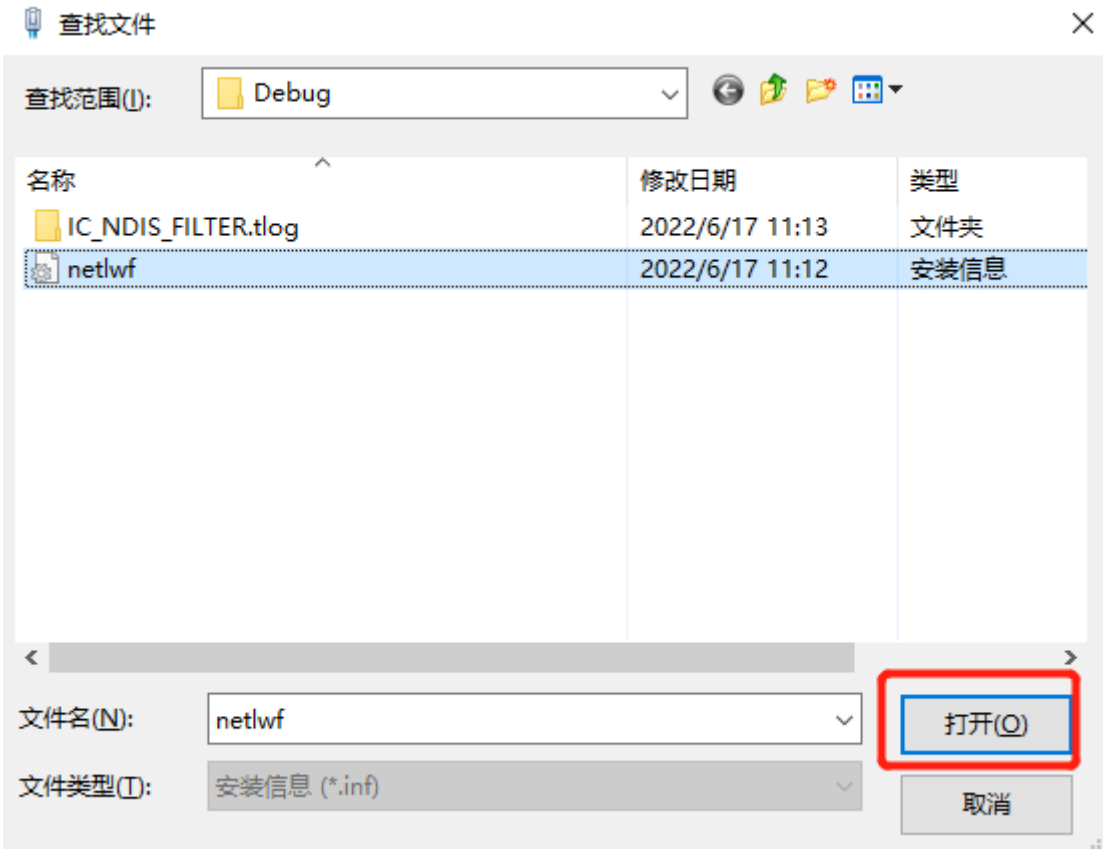
4.点击 安装 服务



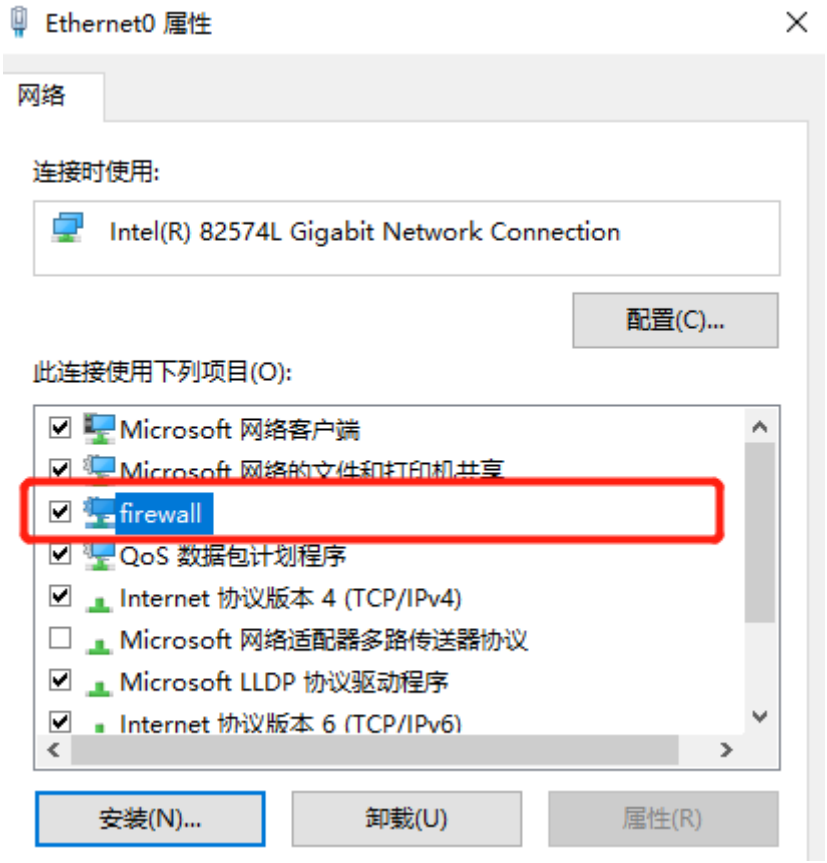

5.点击添加，选择从磁盘中添加




6.选择生成的文件夹中的“netlwf.inf” 单击打开，然后一路确认




**7.安装成功示例**




项目中出现“firewall”

**8.注意事项：**

如果你没有对编译出来的驱动程序进行签名的话，需要在启动项中选择 “禁止驱动签名强制性”，才可正常安装。