Перевод статьи «**Hooking into NDIS and TDI, part 1**»

Перехват NDIS и TDI, часть 1

andreas пишет:
Это первая часть из 2-х статей, описывающих перехват на уровне NDIS и TDI. В первой части мы обсудим перехват на уровне NDIS, а во второй на уровне TDI.

Прежде всего давайте посмотрим на упрощенное изображение сетевого стека в пространстве ядра:

TDI
NDIS уровень протокола
NDIS промежуточный уровень
Miniport
Hardware

Для контроля потока данных на уровне NDIS у нас есть 3 потенциально возможные точки для добавления устройства/драйвера либо перехвата уже существующих. Прежде всего у нас есть уровень мини-порта – драйверы, контролирующие NIC оборудование, но это немного «низковато» для того, что нам надо. Далее у нас есть промежуточный уровень. Этот уровень является наиболее подходящим для наших целей, поскольку он позволит нам контролировать поток данных ко всем NDIS драйверам протоколов. Но тут есть главная помеха: драйвер должен быть с цифровой подписью, чтобы он мог быть добавленным на этот слой.

В зависимости от системы и при каких обстоятельствах мы устанавливаем этот код, нам возможно не удастся легко справится с этой проблемой. Последним у нас идет NDIS протокольного уровня. Добавление драйвера на этот уровень не составит проблем, как к примеру, это делает программа WinPCap. Однако в этом случае мы не сможем контролировать, то что будет видеть пользователь при помощи программ, которые базируются на таких типах драйверов (на пример Ethereal). И все же существует ли способ обхода вопроса цифровой подписи драйвера и в то же время контроля данных на протокольном слое и выше? Да! Мы можем виртуально добавить слой между промежуточным и протокольным уровнем, перехватывая все NDIS протокольные драйвера и их протокольные функции.

После регистрации NDIS протокола он добавляется в связный список. Каждый элемент в этом списке содержит указатель на структуру под названием NDIS_OPEN_BLOCK. Эта структура содержит указатели на все зарегистрированные функциональные указатели протокола. Элементы связного списка представляют собой структуру похожую на следующую:

```
typedef struct _NDIS_LINKED_LIST {
  PNDIS_OPEN_BLOCK pOpenBlock;
  PVOID p;
  REFERENCE ref;
  struct _NDIS_LINKED_LIST *Next;
} NDIS_LINKED_LIST,*PNDIS_LINKED_LIST;
```

Прошло что-то около года с тех пор, как я игрался с этим кодом, поэтому я в принципе не помню название этой структуры. Интересующиеся могут найти его при помощи гугла.

Это также отразится позже в исходном коде, который базируется на абсолютных смещениях вместо определенной выше структуры.

Для того, чтобы иметь возможность перехватить все зарегистрированные NDIS протоколы, нам необходимо найти первый элемент этого связного списка. Фактически это то, что возвращает функция NdisRegisterProtocol в качестве NDIS_HANDLE. Т.е. то что нам необходимо сделать, это зарегистрировать ложный NDIS протокол, сохранить возвращенный указатель и удалить протокол. Это даст нам возможность прохода по списку зарегистрированных NDIS протоколов и обмена существующих указателей на функции на те функции, которыми мы управляем.

Сперва мы регистрируем ложный протокол, для получения указателя. Чтобы быть уверенным в том, что регистрация пройдет успешно, протокол, который мы регистрируем, должен иметь ReceiveHandler:

```
NDIS_STATUS DummyNDISProtocolReceive(
    IN NDIS_HANDLE ProtocolBindingContext,
    IN NDIS_HANDLE MacReceiveContext,
    IN PVOID HeaderBuffer,
    IN UINT HeaderBufferSize,
    IN PVOID LookAheadBuffer,
    IN UINT LookAheadBufferSize,
    IN UINT PacketSize)
{
    return NDIS_STATUS_NOT_ACCEPTED;
}

NDIS_HANDLE RegisterBogusNDISProtocol(void)
{
    NTSTATUS Status = STATUS_SUCCESS;
    NDIS_HANDLE hBogusProtocol = NULL;
    NDIS_PROTOCOL_CHARACTERISTICS BogusProtocol;
    NDIS_STRING ProtocolName;

    NdisZeroMemory(&BogusProtocol,sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
    BogusProtocol.MajorNdisVersion = 0x04;
    BogusProtocol.MinorNdisVersion = 0x0;

    NdisInitUnicodeString(&ProtocolName,L"BogusProtocol");
    BogusProtocol.Name = ProtocolName;
    BogusProtocol.ReceiveHandler = DummyNDISProtocolReceive;

    NdisRegisterProtocol(&Status,&hBogusProtocol,&BogusProtocol,
        sizeof(NDIS_PROTOCOL_CHARACTERISTICS));

    if(Status == STATUS_SUCCESS) return hBogusProtocol;
    else return NULL;
}
```

Как только мы получили указатель, мы можем отменить регистрацию протокола:

```
void DeregisterBogusNDISProtocol(NDIS_HANDLE hBogusProtocol)
{
    NTSTATUS Status;

    NdisDeregisterProtocol(&Status,hBogusProtocol);
```

```
}
```

Проходя по списку и переписывая указатели на функции, нам необходимо сохранять старые указатели, чтобы иметь возможность вызывать исходные функции из нашего кода. Есть как минимум 2 способа сделать это:

1. Создать связный список «перехваченных экземпляров», храня старые указатели для каждого протокола. При вызове наших NDIS-функций этот список должен просматриваться на предмет поиска правильного элемента.
2. Создавать отдельный экземпляр функции для каждого протокола, который мы перехватываем и записывать старый указатель прямо в код функции. Это требует немного больше усилий, но должно быть быстрее в результате во время выполнения, чем поиск по связному списку.

Когда писался этот код, я никогда не задумывался над вторым вариантом, но возможно это то, чем я воспользовался бы сегодня. Так что наслаждайтесь первым вариантом. Он работает нормально и я не замечал каких либо существенных «ударов» по производительности от него.

Для каждого элемента в связном списке зарегистрированных протоколов NDIS, я создам один элемент в своем связном списке и сохраню все необходимые указатели вместе с 2 описателями контекста. Значения описателей будут использованы позже для поиска правильного элемента для текущего протокола. Относительные указатели за тем переписываются таким образом, что указывают на мои версии отсылающих и принимающих функций. Мы также сохраняем указатель на NDIS_OPEN_BLOCK, чтобы проще убирать перехват. Код прохода по списку и перехвата протокола выглядит примерно таким образом:

```
NTSTATUS HookExistingNDISProtocols(void)
{
    UINT *ProtocolPtr;
    NDIS_HANDLE hBogusProtocol = NULL;
    PNDIS_OPEN_BLOCK OpenBlockPtr = NULL;
    PNDIS_PROTOCOL_HOOK pNode;

    hBogusProtocol = RegisterBogusNDISProtocol();
    if(hBogusProtocol == NULL) return STATUS_UNSUCCESSFUL;

    ProtocolPtr = (UINT*)hBogusProtocol;
    ProtocolPtr = (UINT*)((PBYTE)ProtocolPtr + sizeof(REFERENCE) + 8);
    ProtocolPtr = (UINT*)(*ProtocolPtr);

    while(ProtocolPtr != NULL) {
        OpenBlockPtr = (PNDIS_OPEN_BLOCK)(*ProtocolPtr);
        if(OpenBlockPtr != NULL) {
            pNode = NewNDISNode();
            if(pNode != NULL) {
                pNode->ProtocolBindingContext = OpenBlockPtr-
>ProtocolBindingContext;
                pNode->MacBindingContext = OpenBlockPtr->MacBindingHandle;
                pNode->OpenBlockPtr = OpenBlockPtr;
                pNode->RealSendHandler = OpenBlockPtr->SendHandler;
                //How about WanSendHandler?
                pNode->RealPostNt31ReceiveHandler = OpenBlockPtr-
>PostNt31ReceiveHandler;

                InsertNDISNode(pNode);

                OpenBlockPtr->SendHandler = NDISSendHandler;
```

```
                //How about WanSendHandler?
                OpenBlockPtr->PostNt31ReceiveHandler =
NDISPostNt31ReceiveHandler;
            }
        }

        ProtocolPtr = (UINT*)((PBYTE)ProtocolPtr + sizeof(REFERENCE) + 8);
        ProtocolPtr = (UINT*)(*ProtocolPtr);
    }

    DeregisterBogusNDISProtocol(hBogusProtocol);

    return STATUS_SUCCESS;
}
```

Существует гораздо больше интересных функций в NDIS_OPEN_BLOCK для перехвата, но если вы просто хотите контролировать сетевой трафик, то отсылающих и принимающих функций будет достаточно. Другой нюанс о котором стоит упомянуть, это то что NDIS_OPEN_BLOCK меняется от версии к версии ОС. В ОС Windows 2000 он выглядит иначе по сравнению с ОС Windows XP в основном из-за смены имен членов структуры.

Следующее, что следует сделать теперь, это реализовать передающие и принимающие функции, которые выполняют поиск в связном списке, ища указатели на исходные функции для их вызова, если трафик можно пропустить. Если о трафике необходимо сообщить, то это выполняется до вызова исходной функции. Если предполагается игнорировать трафик, то мы можем просто не вызывать исходную функцию и вернуть соответствующий статус.

```
NDIS_STATUS NDISSendHandler(
    IN NDIS_HANDLE MacBindingHandle,
    IN PNDIS_PACKET Packet)
{
    PNDIS_PROTOCOL_HOOK Node

    Node = FindNDISNode(MacBindingHandle,2);
    if(Node == NULL) return NDIS_STATUS_SUCCESS;

    return Node->RealSendHandler(MacBindingHandle,Packet);
}

NDIS_STATUS NDISPostNt31ReceiveHandler(
    IN NDIS_HANDLE ProtocolBindingContext,
    IN NDIS_HANDLE MacReceiveContext,
    IN PVOID HeaderBuffer,
    IN UINT HeaderBufferSize,
    IN PVOID LookAheadBuffer,
    IN UINT LookAheadBufferSize,
    IN UINT PacketSize)
{
    PNDIS_PROTOCOL_HOOK Node;

    Node = FindNDISNode(ProtocolBindingContext,1);
    if(Node == NULL) return NDIS_STATUS_SUCCESS;

    return Node-
>RealPostNt31ReceiveHandler(ProtocolBindingContext,MacReceiveContext,
        HeaderBuffer,HeaderBufferSize,LookAheadBuffer,LookAheadBufferSize,Pac
ketSize);
}
```

Теперь осталось только одно: снятие перехвата. Это делается путем прохода по нашему связному списку и заменой всех указателей на исходные:

```
NTSTATUS ReleaseExistingNDISProtocols(void)
{
    PNDIS_PROTOCOL_HOOK CurrentNode;
    PNDIS_OPEN_BLOCK OpenBlockPtr = NULL;

    CurrentNode = GetFirstNDISNode();
    if(CurrentNode == NULL) return STATUS_UNSUCCESSFUL;

    while(CurrentNode != NULL) {
        OpenBlockPtr = CurrentNode->OpenBlockPtr;
        if(OpenBlockPtr != NULL) {
            OpenBlockPtr->SendHandler = CurrentNode->RealSendHandler;
            OpenBlockPtr->PostNt31ReceiveHandler = CurrentNode-
>RealPostNt31ReceiveHandler;
        }
        CurrentNode = GetNextNDISNode(CurrentNode);
    }

    return STATUS_SUCCESS;
}
```

Что осталось сделать? Код не перехватывает протоколы, которые регистрируются после перехвата NDIS. Выяснение пути для этого оставляется читателю. Код работает? Конечно, я использовал его в win32 версии knockd, названого sesame, который можно найти по адресу [.http://www.toolcrypt.org/.](http://www.toolcrypt.org/)