

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

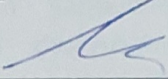
ПРОГРАММА РЕАЛИЗАЦИИ
ЗАЩИЩЁННОГО СОЕДИНЕНИЯ

Студент  30.05.24 Тихонов Андрей Алексеевич
(подпись, дата)

Руководитель  30.05.24 Гурин Евгений Иванович
(подпись, дата)

Нормоконтролёр  30.05.24 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 24.05.24 г. № 11)

Заведующий кафедрой  М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
04.06.2024 № 1)

Секретарь ГЭК  Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 19 » 04 20 24 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Тихонов Андрей Алексеевич гр.20ВВП2 факультета вычислительной техники

2. Тема работы: «Программа реализации защищённого соединения»

Тема утверждена приказом ПГУ № 405/о от " 19 " 04 24 г.

3. Руководитель бакалаврской работы д.т.н., профессор кафедры ВТ ПГУ Гурин Е.И.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать драйвер, выполняющий анализ и шифрование, проходящего через сетевой адаптер, трафика. Драйвер должен выполнять шифрование трафика только на конкретном сетевом адаптере, на который был установлен. Шифрование должно производиться над сетевыми пакетами, передающимися по протоколу IPv4 и содержащими TCP или UDP заголовки транспортного уровня. Определять, подлежит ли шифрованию сетевой пакет или нет драйвер должен при помощи анализа IP адреса отправителя или получателя, в зависимости от направления передачи. Настройка драйвера должна осуществляться при помощи приложения оснастки с графическим интерфейсом. Приложение оснастки должно позволять настраивать IP адрес и криптографический ключ шифрования для нескольких соединений. Так же приложение оснастки должно позволять пользователю включать и выключать функцию шифрования трафика.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки сетевых драйверов. Также необходимо ознакомиться с инструментами для разработки сетевого драйвера, такие как Windows WDK и SDK, а также со стеком сетевых технологий NDIS и особенностями работы различных сетевых протоколов.

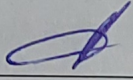
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2024-30.04.2024	
2.	Анализ требований	10	01.05.2024-02.05.2024	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2024-07.05.2024	
4.	Описание алгоритмов	40	08.05.2024-11.05.2024	
5.	Разработка программного обеспечения	150	12.05.2024-29.05.2024	
6.	Оформление пояснительной записки	24	30.05.2024-31.05.2024	

Дата выдачи задания "19 " 04 24 г.

Руководитель бакалаврской работы  19.04.24 Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент  19.04.24 Тихонов А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  31.05.2024 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 74 л., 47 рис., 1 табл., 8 источн. и 3 прил.

РЕАЛИЗАЦИЯ ЗАЩИЩЁННОГО СОЕДИНЕНИЯ ПРИ ПОМОЩИ ДРАЙВЕРА, ШИФРОВАНИЕ СТЕВОВОГО ТРАФИКА, ДРАЙВЕР ОПЕРАЦИОННОЙ СИСТЕМЫ WINDOWS, ДРАЙВЕР ФИЛЬТР NDIS

Цель работы – разработать сетевой драйвер, выполняющий шифрование исходящих и приходящих на сетевой адаптер сетевых пакетов, и оснастку, позволяющую выполнять настройку драйвера.

Разработанный драйвер должен выполнять анализ и шифрование сетевого трафика конкретного сетевого адаптера, на который он был установлен. Шифрование должно производиться только для тех сетевых пакетов, которые подходят под настройки драйвера. Драйвер должен поддерживать шифрование таких протоколов как TCP и UDP и выполнять аутентификацию при помощи анализа IPv4 заголовка.

Результатом разработки является проект, удовлетворяющий поставленной задаче. А именно, разработан, реализован и протестирован драйвер для шифрования сетевого трафика, а также оснастка для настройки и управления разработанным драйвером. Разработанный драйвер написан на языке программирования C, с использованием наборов инструментов WDK и SDK, а также с использованием библиотеки NDIS. Оснастка драйвера разработана на языке программирования C# с использованием .NET. В качестве среды разработки обоих проектов использовалась IDE Visual Studio 2019, а для отладки драйвера в процессе разработки программа WinDBG.

					ПГУ 1.090301.37.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Программа реализации защищённого соединения	Лит.	Лист	Листов	
Разраб.		Тихонов А.А.		30.05.24			2	74	
Провер.		Гурин Е.И.		30.05.24					
Н. Контр.		Бычков А. С.		30.05.24					
Утверд.		Митрохин М.А.		30.05.24	Пояснительная записка	ФВТ, каф. ВТ			
						гр. 20ВВП2			

Содержание

Перечень сокращений и обозначений	3
Введение	4
1 Анализ темы	5
1.1 Актуальность темы	6
1.2 Анализ аналогов	7
1.3 Постановка задачи	11
2 Выбор средств и инструментов разработки	13
2.1 IDE Visual Studio 2019	13
2.2 Язык программирования C	14
2.3 WDK	14
2.4 Windows SDK	15
2.5 NDIS	15
2.6 Язык программирования C#	16
2.7 .NET	17
3 Архитектура драйвера	18
4 Реализация программного обеспечения	26
4.1 Регистрация драйвера	26
4.2 Работа с реестром	28
4.3 Получение сетевых листов и выделение заголовков из пакетов	30
4.4 Получение полезной нагрузки для шифрования	34
4.5 Разбор алгоритма шифрования	39
4.6 Обработка IOCTL запросов	42
5 Установка и тестирование	46
5.1 Настройка реестра и оснастки	46
5.2 Установка драйвера	49
5.3 Тестирование	54
5.3.1 Тестирование без шифрования	55
5.3.2 Тестирование с шифрованием	58
5.3.3 Тестирование с неправильно настроенным шифрованием	63
5.3.4 Тестирование скорости передачи данных	67
6 Безопасность жизнедеятельности	70
Заключение	71
Список использованных источников	72

Приложение А	73
Приложение А.1 – Диаграмма развёртывания	74
Приложение А.2 – Диаграмма последовательностей	75
Приложение А.3 – Диаграмма деятельности 1	76
Приложение А.4 – Диаграмма деятельности 2	77
Приложение Б	78
Приложение В	111

					ПГУ 1.090301.37.001 ПЗ	Лист
						2
Изм.	Лист	№ докум.	Подпи	Дата		

Перечень сокращений и обозначений

HTTP (англ. Hyper Text Transfer Protocol) – протокол передачи гипертекста

IDE (англ. Integrated Development Environment) – интегрированная среда разработки

MDL (англ. Memory Descriptor List) – структура отображения памяти

NDIS (англ. Network Driver Interface Specification) – спецификация сетевого интерфейса для драйверов

SDK (англ. Software Development Kit) – набор инструментов и компонентов для разработки

UML (англ. Unified Modeling Language) – унифицированный язык моделирования

WDK (англ. Windows Driver Kit) – программный набор инструментов и библиотек для разработки драйверов

					ПГУ 1.090301.37.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпи	Дата		

Введение

В современном мире, персональных или конфиденциальных данных, находящихся в сети интернет или внутренних базах данных различных компаний, учреждений и отдельных пользователей с каждым годом становится всё больше. Так же в ходе использования этих данных и во время их передачи по сети они могут быть украдены и использованы третьими лицами.

Кроме персональных и конфиденциальных данных по сети передаются большое количество других различных сетевых пакетов, которые используются различными приложениями или браузерами. Такие пакеты могут быть так же перехвачены и изменены злоумышленниками, что может нарушить работу используемых пользователем приложений, сайтов и оборудования, а также обеспечить злоумышленникам полный доступ к данным.

Так как стандартная защита передаваемых по сети данных, такая как защищённые протоколы по типу HTTPS, FTP и другие, зачастую является недостаточной, разработка программной защиты передаваемых сетевых пакетов всегда будет актуальной темой.

					ПГУ 1.090301.37.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпи	Дата		

1 Анализ темы

Задача защиты передачи данных в сети всегда стояла остро как для простых пользователей, так и различных компаний, государственных учреждений и даже военной отрасли. С каждым годом количество различных видов информации, в том числе большой объём конфиденциальной, которую необходимо хранить или передавать в электронном виде растёт и будет продолжать расти, что поддерживает тему защиты передачи информации всегда актуальной.

Информация, передаваемая в открытом (не зашифрованном) виде, может быть перехвачена многими способами, вследствие чего данные, в том числе конфиденциальные, могут быть скопированы или изменены так как необходимо злоумышленнику, например, для имплементации вредоносных скриптов в сетевые запросы. Одним из способов защитить информацию во время передачи это передавать её в закрытом (зашифрованном) виде. При этом шифровать можно не только данные, но и, например, изменять или добавлять сетевые протоколы путём инкапсуляции или добавлять дополнительные условия проверки приходящих данных. Даже если данные в таком виде будут перехвачены, для извлечения из них информации злоумышленнику необходимо сначала расшифровать их. Сложность дешифрации зависит от выбранного алгоритма шифрации и дополнительных защитных условий, добавленных разработчиками.

Стоит так же отметить, защита передачи данных является непростой задачей не только со стороны алгоритмизации, но и оптимизации. Мало придумать сложный алгоритм шифрования и построения безопасных соединений, важно помнить и о скорости передачи данных, так, как например слишком сложные алгоритмы шифрования могут сильно нагружать систему и уменьшать скорость передачи данных в десятки и сотни раз. Вследствие чего можно сделать вывод, что разработка программных средств защиты передачи информации является непростой задачей.

					ПГУ 1.090301.37.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпи	Дата		

1.1 Актуальность темы

В современном мире, как бы нам не казалось, что с развитием информационных технологий и увеличением объёма персональных и конфиденциальных данных так же будет уменьшаться количество ситуаций утечек данных. Однако, как показывает статистика их не только не стало значительно меньше, но и значительно вырос объём украденных данных.

Так, из отчёта «Лаборатории Касперского» за 2023 год из зафиксированных случаев утечек краж данных только значимых баз данных российских компаний объём опубликованной информации возрос на 33%, не говоря уже о более мелких представителях среднего и малого бизнеса, а так же государственных организаций [1]. Данная статистика продемонстрирована на рисунке 1.



Рисунок 1 – Статистика утечек данных российских компаний за 2022 и 2023

По статистике данных по отраслям в России можно заметить, что больше всего от утечек страдают:

– ритейл. Самая пострадавшая отрасль за 2022 – 2023 год с 37 крупными утечками данных;

- интернет сервисы. 33 крупных взлома за последние два года;
- финансы. 16 взломов за последние два года, причём 14 из которых приходятся на 23 год;
- карьера и образование. 25 крупных утечек данных за два года;
- IT. 14 взломов, наибольшая часть из которых приходится на 2023 год.

Так же взломам с последующими утечками данных были подверженные и государственные отрасли. Например, за последние два года было замечено 16 крупных взломов в различных госорганах и 10 в отрасли здравоохранения, а также в 2022 году было обнаружено 18 утечек данных в сервисах доставки еды и ресторанах.

От многих из приведённых примеров утечек данных как можно было заметить пострадали не только компании и их конфиденциальные данные, но и личные данные и информация пользователей, например медицинская информация пациентов из утечек баз данных больниц и других медицинских учреждений или адреса места жительства и телефоны при взломах сервисов ресторанов и служб доставки были опубликованы и проданы в сети интернет. Все вышеприведённые примеры подтверждают важность развития информационной безопасности и актуальность выбранной в рамках бакалаврской работы темы.

Так же стоит отметить, что для того, чтобы обезопасить передачу данных и избежать утечек, необходимо передавать данные по защищённому каналу. Примером такого соединения может послужить соединение точка-точка, при котором при передачи любых данных между двумя компьютерами будет производиться шифрование на отдельных для каждого соединения ключах. Именно такой алгоритм построения зашифрованных каналов будет использовать драйвер, разработанный в данной дипломной работе

1.2 Анализ аналогов

На данный момент открытых популярных программных решений шифрования трафика на сетевом уровне модели OSI практически не существует. Большинство аналогов программного обеспечения, решающего задачу

					ПГУ 1.090301.37.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпи	Дата		

шифрования передаваемой информации являются одним из компонентов программ и работает на другом уровне сетевой модели OSI, примером таких программ может послужить мессенджер «Telegram» в котором реализовано шифрование передаваемых сообщений, или же является одной из частей комплекса программного обеспечения которое обладает грифом секретности и разрабатывается на заказ для конкретных компаний или государственных учреждений.

Одним из ближайших аналогов являются программы VPN. VPN – это технологии создания виртуальной частной сети, которая объединяет одно или несколько устройств посредством туннелирования (рисунок 2), то есть построения отдельного защищённого канала связи [2]. При этом VPN сеть обычно строится с использованием одного или нескольких компьютеров-серверов, которые и управляют процессом шифрования данных в туннеле и сокрытии IP адреса клиента. Стоит отметить, что у разных VPN провайдеров может различаться степень шифрования передаваемых данных.

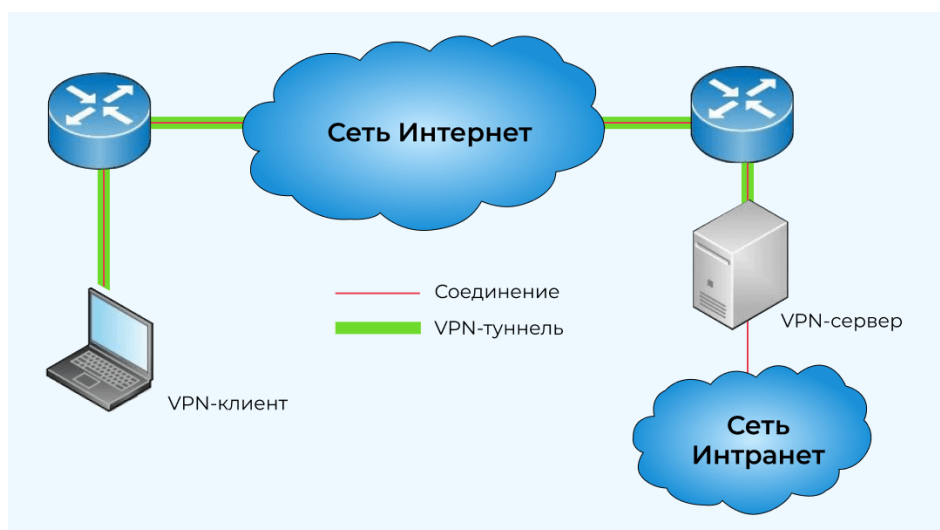


Рисунок 2 – Пример построение туннельного соединения с VPN сервером

Конкретная часть комплекса VPN технологий, которую можно называть ближайшим аналогом программы, разработанной в ходе выполнения данной бакалаврской работы, является VPN протокол. VPN протоколы как правило

выполняют две основные функции проверку подлинности (авторизацию) и шифрование.

В основном выделяют следующие VPN протоколы:

– PPTP. Point-to-Point Tunneling Protocol один из первых VPN-протоколов. Из-за примитивного шифрования является очень быстрым, но не безопасным. На данный момент считается устаревшим;

– SSTP. Secure Socket Tunneling Protocol использует SSL-протокол, инкапсулирует пакеты данных по HTTPS и поддерживает шифрование AES-256. Это протокол работает быстрее других, но и использует больше ресурсов вычислительной системы;

– L2TP/IPsec Layer 2 Tunneling Protocol и IPsec. Эти обычно используются в паре так как протокол L2TP, созданный как замена устаревшему PPTP не поддерживает шифрование данных, поэтому эту роль выполняет IPsec. По сравнению с другими протоколами, у L2TP/IPsec средняя скорость передачи данных, однако эта связка требует большого количества ресурсов вычислительной системы;

– OpenVPN. Open-Source Virtual Private Network в шифровании опирается на библиотеку OpenSSL и поддерживает разные алгоритмы шифрования такие как например AES и Blowfish. По скорости этот протокол занимает промежуточное место между PPTP и связки L2TP/IPsec.

В большинстве VPN протоколов используются следующие алгоритмы шифрования:

– шифр AES. Один из самых безопасных шифров. Доступен с длиной ключа 128 и 256 бит;

– Blowfish. Данный шифр раньше он был шифром по умолчанию, используемым в большинстве VPN-подключений, но теперь его в значительной степени заменил AES-256;

– ChaCha20. Набирающий шифр популярность как единственный совместимый с протоколом WireGuard. Как и AES, ChaCha20 использует ключ длиной 256 бит, что считается очень безопасным;

					ПГУ 1.090301.37.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпи	Дата		

– Camellia. Шифр, очень похожий на AES с точки зрения безопасности и скорости. Считается, что даже при использовании опции меньшей длины ключа (128 бит) невозможно взломать с помощью атаки грубой силы, учитывая современные технологии.

Часть аутентификации и шифрования данных в вышеприведённых можно считать за аналоги разработанного в данной работе программного обеспечения. Из всех VPN протоколов можно выделить две основных характеристики – скорость работы или влияние на производительность системы и степень безопасности алгоритма шифрования.

Так же в качестве аналога можно привести в пример различные программы построения SSH соединения. SSH сокращение от Secure Shell используется для обеспечения безопасного удалённого доступа к различным операционным системам. В основном данный протокол используется для удалённого управления сервером по принципу точка-многоточка, т.е к одному SSH компьютеру (серверу) могут подключаться несколько компьютеров (клиентов).

Для настройки SSH соединения необходимо на одном компьютере, к которому предполагается подключение запустить программу SSH сервера (например OpenSSH или freeSSHd) и настроить тип подключения, аутентификации и шифрования. На компьютере для управление передачей файлов, управлением системой или просмотром баз данных необходимо запустить SSH клиент (например SSHWindows, OpenSSH, Xshell) и подключиться по логину или паролю, настроенному на сервере, после чего между клиентом и сервером будет построено зашифрованное соединение.

Как можно увидеть, построение SSH соединения несёт в себе много лишнего функционала и дополнительных настроек, таких как отдельного создания клиента и сервера для подключения и возможность управления не только передачей данных, но и другими частями системы сервера. Так же стоит отметить, что программы построения SSH соединения никак не защищены от троянов и вирусов, расположенных на конечных компьютерах и в качестве

основы для шифрования, использует только TCP технологии так как работает поверх TCP протокола, т.е использование любых других протоколов будет производиться без шифрации.

При этом SSH сервисы предлагают на выбор использования разных типов шифрования, таких как:

- симметричное. При таком кодировании один и тот же ключ шифрования используется и для шифрации и дешифрации передаваемых данных;
- ассиметричное. При таком кодировании используется пара ключей, открытый и закрытый. Открытый ключ используется для шифрования данных и может передаваться в открытом виде, а закрытый для расшифровки.

1.3 Постановка задачи

В данной ВКР решается задача реализация создания защищённого соединения путём шифрования пакетов данных. Для этого необходимо реализовать механизм защиты данных, передаваемых по сети, при помощи драйвера, внедряемого в сетевой стек, а также оснастку для управления разработанным драйвером.

Стоит отметить, что для выполнения поставленной задачи драйвер должен работать на сетевом уровне модели OSI [3] (рисунок 3) т.к. именно на этом уровне у драйвера будет возможность просматривать и управлять всеми сетевыми пакетами, приходящими на сетевой адаптер.

Для работы со всеми пакетами данных необходимо разработать механизмы получения всей полезной нагрузки (данных, содержащихся в сетевом пакете). Так же необходимо разработать алгоритм шифрования полезной нагрузки, который будет и отвечать требованиям безопасности и не слишком сильно замедлять систему.

Так как аутентификация в драйвере будет происходить по Ip адресу пакета, а шифрование по ключу в оснастке драйвера необходимо добавить возможность управлять данными параметрами, т.е. добавлять изменять или удалять пары этих данных. Так же необходимо реализовать возможность создания и настройки

					ПГУ 1.090301.37.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпи	Дата		

одновременно нескольких защищённых соединений с разными компьютерами путём добавления в оснастки новых записей с Ip адресом и ключом шифрования.

Модель				
Уровень (layer)	Тип данных (PDU)	Функции	Примеры	Оборудование
Host layers	7. Прикладной (application)	Данные	HTTP, FTP, POP3, SMTP, WebSocket	Хосты (клиенты сети), Межсетевой экран
	6. Представления (presentation)		ASCII, EBCDIC, SSL, gzip	
	5. Сеансовый (session)		RPC, PAP, L2TP, gRPC	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность TCP, UDP, SCTP, Порты	
Media layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация IPv4, IPv6, IPsec, AppleTalk, ICMP	Маршрутизатор, Сетевой шлюз, Межсетевой экран
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.	Сетевой мост, Коммутатор, точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор, Повторитель (сетевое оборудование)

Рисунок 3 – Описание уровней сетевой модели OSI

Шифрование, в разработанном драйвере должно работать на симметричных ключах, которые не должны передаваться по сети, а настраиваться локально для каждого соединения для повышения безопасности. Так же стоит отметить, что шифрование в драйвере будет производиться в потоковом режиме, а не блочном.

Установка драйвера должна производиться на конкретный выбранный адаптер, для внедрения его в сетевой стек одного конкретного физического канала связи если это необходимо. Например, если в системе присутствует два сетевых адаптера, и мы хотим передавать данные и стоять защищённые соединения только с одного из них, это можно будет сделать.

2 Выбор средств и инструментов разработки

Для разработки драйвера и графического интерфейса из поставленной задачи необходимо использовать наборы инструментов и средств разработки, как и приложений пользовательского режима (user mode), так и средств разработки программ и драйверов режима ядра (kernel mode)[4].

Средства и инструменты разработки, выбранные для разработки драйвера режима ядра:

- WDK;
- Windows SDK;
- NDIS;
- IDE Visual Studio 2019;
- язык программирования С.

Выбранные инструменты разработки для создания пользовательского интерфейса:

- .NET Framework;
- IDE Visual Studio 2019;
- язык программирования C#.

2.1 IDE Visual Studio 2019

Visual Studio – это интегрированная среда разработки, разработанная компанией Microsoft. Данная среда разработки используется для редактирования кода, создания и отладки приложений различных платформ, сборки кода, а также публикации проектов. Visual studio позволяет разрабатывать все типы приложений, от драйверов и служб до веб приложений и поддерживает разработку на таких языках программирования как: C++, C#, Python, JavaScript, TypeScript и другие.

Так же важно отметить возможность установки дополнительных компонентов или библиотек, которые расширяют функционал и возможности программиста. Так, например, при помощи дополнительных компонентов можно

добавить возможности использования других языков программирования, разработки облачных, мобильных или веб приложений, средства работы с базами данных, создания расширений для приложений Microsoft включая и саму Visual Studio. В данной дипломной работе так же использовалась эта возможность среды выбранной разработки для установки дополнительных библиотек, средств сборки и компонентов для работы с драйверами сетевого стека, использующими NDIS [5].

2.2 Язык программирования C

C - компилируемый язык программирования общего назначения, сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования. Был разработан в 1972 году Деннисом Ритчи с идеей создать такой язык программирования, который был приближен к ассемблеру по эффективности и быстродействию, но в то же время имел понятный и простой для человека синтаксис. Преимущество языка C в том что с его помощью можно написать программы, у которых будет обширный доступ к памяти и аппаратным устройствам компьютера на очень низком уровне, именно поэтому это язык программирования используется для написания драйвера в данной выпускной работе.

2.3 WDK

WDK (Windows Driver Kit) – это программный набор инструментов и библиотек, выпущенный компанией Microsoft который позволяет разрабатывать различные драйвера для операционной системы Windows [6]. В его состав входят документация, средства сборки и отладки, инструменты для разработки драйверов. Так же WDK имеет своё расширение в Visual Studio для его имплементации в данное IDE. Важно отметить, что для получения полного набора инструментов вместе с WDK так же необходимо установить и использовать SDK соответствующей версии.

					ПГУ 1.090301.37.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпи	Дата		

WDK содержит инструменты и библиотеки для разработки различных типов драйверов, от драйверов устройств, которые например позволят использовать в разработанное вами устройство PCI или USB в операционной системе Windows, до драйверов которые будут встраиваться в стек уже работающих драйверов. В данной работе как раз используется последний тип драйвера, который встраивается в стек драйверов сетевого адаптера.

2.4 Windows SDK

SDK (Software Development Kit) – это набор инструментов и компонентов для разработки различных приложений, как пользовательского режима, так и режима ядра. Этот пакет инструментов предоставляет набор API-интерфейсов которые можно применять для разработки различных приложений под Windows. Стоит отметить, что Windows SDK не заменяет существующие стандартные SDK и типы приложений Windows, а лишь дополняет уже существующие инструменты и модели приложений своими API.

В данной работе SDK используется совместно с WDK в качестве пакетного инструментария, для создания драйвера. Для этого оба этих пакета имплементируются и используются в среде разработки Visual Studio 19.

2.5 NDIS

NDIS (Network Driver Interface Specification) – это спецификация сетевого интерфейса для драйверов, которая определяет и устанавливает правила того, как сетевые драйверы должны взаимодействовать с сетевым стеком операционной системы Windows. Кроме того, NDIS так же предоставляет библиотеку для работы с сетевым стеком в драйверах. С помощью NDIS появляется возможность разрабатывать драйверы сетевых устройств, например сетевых карт, модемов и коммутаторах на разных уровнях. С его помощью можно как создавать собственные виртуальные адаптеры, фильтры трафика или даже собственные сетевые протоколы.

Основные возможности, которые предоставляет NDIS:

					ПГУ 1.090301.37.001 ПЗ	Лист
						15
Изм.	Лист	№ докум.	Подпи	Дата		

– управлять буферами данных, проходящими через сетевое устройство. В связи с чем появляется возможность фильтровать, изменять, добавлять или удалять сетевые данные (например пакеты или сетевые фреймы);

– позволяет получать уведомления и обрабатывать сетевые события. устройства, такие как отправка или получение пакетов, изменения состояния устройства и другие, что позволяет создавать различные типы драйверов сетевых устройств;

– имплементация в стек драйверов. Благодаря этой особенности NDIS может встраиваться в стек уже существующих драйверов на разных уровнях и управлять потоком данных на этих уровнях. К примеру, с его помощью драйвер фильтра NDIS может задержать, отменить, добавить пакеты и передать вверх или вниз по стеку драйверов, что бы например конечное веб или приложение пользовательского режима получила данные в необходимом системе виде или не получила его совсем.

Все вышеописанные возможности доказывают, что NDIS является очень мощным инструментом, использующимся для создания сетевых драйверов или драйверов сетевых устройств.

В данной работе библиотека NDIS используется для обеспечения эффективного взаимодействия между драйвером и ядром операционной систем. Он позволяет разработанному драйверу внедряться в стек драйверов NDIS и осуществлять перехват пакетов, их фильтрацию и их шифрацию или дешифрацию, в зависимости от события.

2.6 Язык программирования C#

C# — это типизированный, объектно-ориентированный, простой и в то же время мощный язык программирования, который позволяет разработчикам создавать многофункциональные приложения. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET (программной платформы от компании Microsoft, предназначенной для

создания обычных программ и веб-приложений). С# является мощным и гибким языком программирования, который позволяет разработчикам создавать широкий спектр приложений, в данной выпускной работе данный язык программирования использовался для создания графического интерфейса управления драйвером.

2.7 .NET

.NET – это кроссплатформенная среда разработки, позволяющая создавать приложения для различных операционных систем, включая Windows Linux macOS. При этом .NET включает в себя различные библиотеки, инструменты и фреймворки, позволяющие разрабатывать различные типы приложений, от игр до веб сервисов. Так же стоит отметить, что .NET позволяет легко интегрировать в себя новые библиотеки, и фреймворки и технологии. В этом ключе стоит так же упомянуть технологию Windows Forms, которая используется для создания графического интерфейса.

В данной работе на базе .NET будет разработано приложение с графическим интерфейсом, позволяющее управлять драйвером. При разработке графического интерфейса приложения использовалась технология Windows Forms.

3 Архитектура драйвера

Драйвер, разрабатываемый в данной выпускной работе, встраивается в стек сетевых драйверов NDIS как драйвер фильтр. Стек сетевых драйверов NDIS является частью операционной системы и представляет собой набор драйверов, которые реализуют функции NDIS по реализации взаимодействия между сетевым оборудованием и операционной системой. Т.е. данный стек драйверов отвечает за по сети, управление сетевыми соединениями и обработку сетевых пакетов. Стоит так же отметить, что у каждого сетевого адаптера в системе, не важно виртуальный он или физический существует свой собственный стек сетевых драйверов NDIS. NDIS предоставляет несколько типов сетевых драйверов [7]:

- драйвера минипорт-адаптера. Это один из типов драйверов NDIS который выполняет взаимодействие с конкретным физическим устройством (сетевая карта, модем и т.д.). Минипорт драйвера обеспечивают низкоуровневый доступ к аппаратным ресурсам, что позволяет операционной системе как-либо взаимодействовать с устройством. В стеке драйверов NDIS данный драйвер является самым низкоуровневым;

- промежуточные драйвера NDIS. Это один из двух типов промежуточных драйверов, которые располагаются над драйверами минипорта, но всегда находятся перед драйверами протоколов. Главной особенностью данного типа драйверов является то, что они могут управлять сразу несколькими сетевыми интерфейсами. В основном промежуточные драйвера NDIS используются для обеспечения корректной работы сетевых интерфейсов, например обеспечения распределения нагрузки между несколькими сетевыми адаптерами в системе, но они так же могут использоваться для простой обработки сетевых пакетов;

- драйвера фильтры. Это второй тип промежуточных драйверов NDIS, который располагается между драйверами минипорта и сетевого протокола. Данный тип драйверов выполняет функцию предварительной обработки пакетов в стеке, например данный драйвер может использоваться для фильтрации пакетов по определённым критериям, описанным разработчиками, изменение данных в проходящих по стеку пакетах или добавление новых или удаления пакетов. Кроме

мониторинга и изменения проходящих данных данный тип драйверов может использоваться и для распределения нагрузки в системе, например путём запуска параллельной обработки проходящих через него данных. Но стоит что драйвера фильтров, в отличии от промежуточных драйверов могут управлять только одним сетевым устройством. Кроме функций работы событий обработки пакетов, этот тип драйвера так же может реагировать и на изменения состояния работы сетевого адаптера, к которому он относится, например если он выключается, уходит в сон или на перезагрузку, что ещё расширяет спектр применений и возможности реализации функционала данного типа драйвера:

– драйвера протоколов NDIS. Протоколы является драйверами самого верхнего уровня в стеке драйверов NDIS. Этот тип драйвера отвечает за взаимодействие с системой, для доставки данных до конечных приложений или от конечных приложений вниз по стеку. Так же эти драйверы используются для реализации конкретных сетевых протоколов, например TCP/IP или UDP. Хотя сетевых протоколов существует огромное множество, в стеке драйверов NDIS (рисунок 4) стоит самым верхним и в конечном счёты выбирается только один после прохождения по всей цепочке драйверов [8].

Стоит так же отметить, что в стеке драйверов, может содержаться любое количество промежуточных драйверов и драйверов фильтров, что позволяет каждому отдельному драйверу выполнять свою отдельную функцию. Однако не следует создавать слишком высокий стек драйверов, так как это может сильно замедлить производительность системы. Далее на рисунке представлен стек драйверов с неопределённым числом промежуточных драйверов и драйверов фильтров. Далее на рисунке 5 представлена UML диаграмма развёртывания разработанной системы для построения защищённого соединения.

Далее на рисунке 6 представлена UML диаграмма последовательности использования оснастки драйвера пользователем



Рисунок 4 – Пример стека NDIS драйверов



Рисунок 5 – UML диаграмма развёртывания

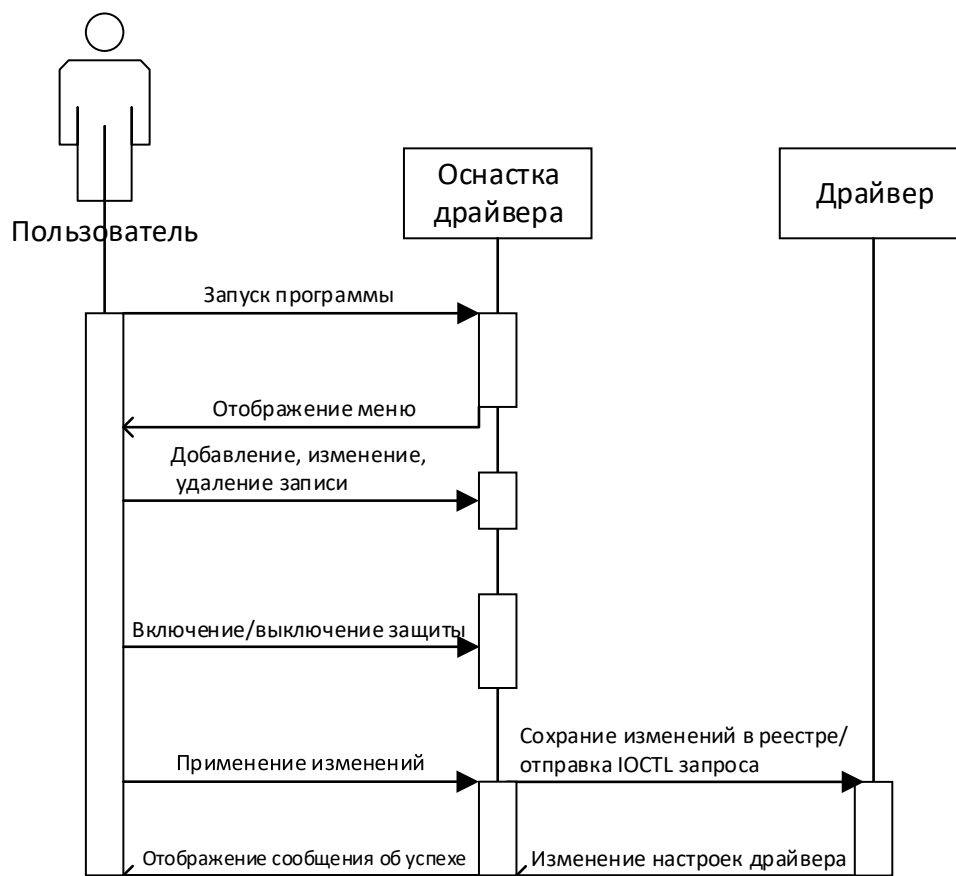


Рисунок 4 – UML диаграмма последовательности

Обработка пакетов в разработанном драйвере происходит в соответствии с протоколами NDIS. Так как разработанное ПО является драйвером фильтром, он должен зарегистрировать в системе определённые функции, на которые будет ссылаться NDIS при передаче пакетов данных по стеку драйверов. Следующие функции, отвечающие непосредственно за обработку пакетов данных, являются самыми важными:

- SendNetBufferLists. Функция, в которую NDIS передаёт сетевые листы, содержащие пакеты, передаваемые по стеку вниз. В этой функции происходит обработка пакетов при отправке пакетов по сети;

- SendNetBufferListsComplete. Функция, в которую NDIS передаёт сетевые листы, при завершении отправки пакетов. В данной функции драйвер освобождает ресурсы, выделяемые при обработке сетевых пакетов на отправке;

- ReceiveNetBufferLists. Функция, в которую NDIS передаёт сетевые листы, содержащие пакеты, передаваемые по стеку вверх. В этой функции происходит

обработка пакетов при получении пакетов из сети;

- ReturnNetBufferLists. Функция, в которую NDIS передаёт сетевые листы, при завершении получение сетевых пакетов. В данной функции драйвер освобождает ресурсы, выделяемые при обработке сетевых пакетов при получении.

При отправке или получении пакетов драйвер получает из NDIS сетевые пакеты в виде сетевых листов (NET_BUFFER_LIST) в соответствующую функцию. В сетевых листах содержится информация о отдельных пакетах в виде сетевых буферов (NET_BUFFER). Задача драйвера при обработке трафика в обоих направлениях разбить сетевой лист на сетевые буферы и обработать по отдельности каждый сетевой буфер.

Стоит отметить, что драйвер на данный момент поддерживает шифрование пакетов передаваемый по протоколу IPv4 и транспортным протоколам UDP или TCP. Данный факт даёт возможность развития данного драйвера, путём добавления большего количества протоколов, например при помощи инкапсулирования протоколов.

Схема обработки при получении или отправки сетевых листов выглядит следующим образом:

- разбиение сетевого листа (NET_BUFFER_LIST) на сетевые буферы (NET_BUFFER), для дальнейшей обработки полученных данных в виде отдельных сетевых пакетов;

- получение Ethernet заголовка из пакета, для проверки следующего сетевого заголовка. Если следующий заголовок IPv4 обработка пакета продолжается;

- извлечение из IPv4 ip-адреса отправителя или получателя, в соответствии с направлением передачи данных;

- поиск по IP адресу записи с ключом шифрования для дальнейшей обработки пакета, если запись не найдена, пакет будет передан в открытом виде;

- проверка заголовка транспортного уровня, на наличие TCP или UDP заголовка;

– шифрование или расшифровка данных в пакете, следующих после транспортного UDP или TCP заголовка, более подробно процесс шифрования и работа с данными в NET_BUFFER будет рассмотрена в следующей главе;

– после завершения процесса шифрования производится переход к завершению обработки сетевого пакета и переход к обработке следующего в сетевом листе, пока все пакеты в листе не будут обработаны.

Разрабатываемый в данной работе драйвер при передаче данных в безопасном режиме существует защищённое соединение, задаваемое при помощи настроек оснастки, для каждого взятого клиента. Такой тип соединения устройств в сети называется точка-точка. В компьютерных сетях такой тип соединений обычно используется для передачи данных между двумя конкретными устройствами. На следующей схеме (рисунок 7) можно увидеть какая топология сети получится при построении защищённого соединения с несколькими устройствами.

Защищённое соединение

Открытое соединение

ПК1	192.168.0.1
ПК2	192.168.0.2
ПК3	192.168.0.3
ПК4	192.168.0.4
ПК5	192.168.0.5
ПК6	192.168.0.6

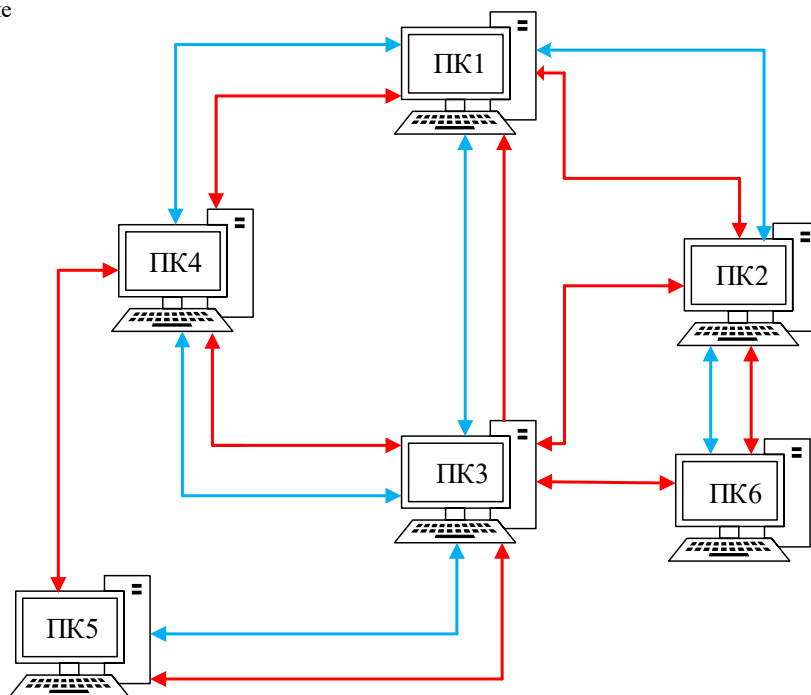


Рисунок 5 – Пример защищённой топологии сети

Из данной схемы можно понять, что между каждой парой компьютеров, у которых настроена функция защиты сети разработанного драйвера, строится

защищённое соединение. У компьютеров в данном примере топологии настройки драйвера отображены в таблице 1.

Таблица 1 – Настройки защищённых соединений

Имя компьютера	IP-адреса	Ключи
ПК1	192.168.0.3	Key13
	192.168.0.2	Key12
	192.168.0.4	Key14
ПК2	192.168.0.6	Key26
	192.168.0.1	Key12
ПК3	192.168.0.1	Key13
	192.168.0.4	Key34
	192.168.0.5	Key35
ПК4	192.168.0.1	Key12
	192.168.0.3	Key34
	192.168.0.5	WrongKey1
ПК5	192.168.0.3	Key35
	192.168.0.4	WrongKey2
ПК6	192.168.0.2	Key26

Из настроек можно увидеть, что защищённое соединение будет установлено только при правильных настройках IP-адреса и ключа на обоих клиентах. IP-адрес используется для идентификации компьютера и обнаружения пакета клиента, пакеты которого необходимо шифровать или расшифровывать. Стоит отметить, что ключ должен быть одинаковым на обоих клиентах, иначе передача зашифрованных данных между клиентами будет невозможна. Данную ситуацию можно увидеть в соединении между ПК4 и ПК5, так как в таблице настроек драйвера для данных компьютеров находятся отличающие ключи шифрования.

Так же стоит отметить, что, например, если между компьютерами ПК3 и ПК2 защищённое соединение не будет автоматически построено через

промежуточную точку ПК1. При данных настройках защиты сети на компьютерах, между ПК3 и ПК2 данные можно будет передавать только в открытом виде.

					ПГУ 1.090301.37.001 ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпи	Дата		

4 Реализация программного обеспечения

В процессе разработки драйвера фильтра необходимо реализовать множество функций, которые используются для обеспечения корректного взаимодействия с операционной системой Windows и встраивания драйвера в стек. К таким функциям относится, например: привязка драйвера к устройству, регистрация драйвера, события изменения состояния, обработка PnP событий, обработка Oid запросов и т.д. Поскольку вышеописанный функционал не имеют отношения к теме фильтрации и шифрованию трафика в данной главе они не будут рассмотрены.

Однако в данной главе будут рассмотрены основные компоненты драйвера и оснастки, а именно:

- регистрация драйвера;
- работа с реестром;
- получение сетевых листов и выделение заголовков из пакетов;
- получение полезной нагрузки для шифрования;
- разбор алгоритма шифрования;
- обработка IOCTL запросов.

4.1 Регистрация драйвера

Регистрация драйвера в виде фильтра происходит в функции DriverEntry, которая является функцией входа в драйвер (аналогом функции main). При регистрации драйвера фильтра основная задача состоит в заполнении структуры характеристик драйвера NDIS (NDIS_FILTER_DRIVER_CHARACTERISTICS). В данной структуре располагается информация самом драйвере, например, имя драйвера, версии NDIS с которым работает драйвер и тип драйвера.

Так же в данной структуре находятся указатели на функции обработчики событий, которые необходимы для корректной работы драйвера и вызываются NDIS или системой по мере необходимости. Далее представлен с участок кода с

					ПГУ 1.090301.37.001 ПЗ	Лист
						26
Изм.	Лист	№ докум.	Подпи	Дата		

установкой всех обработчиков и вызовом функции регистрации драйвера и его привязки к устройству.

```
FChars.SetOptionsHandler = FilterRegisterOptions;
FChars.AttachHandler = FilterAttach;
FChars.DetachHandler = FilterDetach;
FChars.RestartHandler = FilterRestart;
FChars.PauseHandler = FilterPause;
FChars.SetFilterModuleOptionsHandler = FilterSetModuleOptions;
FChars.OidRequestHandler = FilterOidRequest;
FChars.OidRequestCompleteHandler = FilterOidRequestComplete;
FChars.CancelOidRequestHandler = FilterCancelOidRequest;

FChars.SendNetBufferListsHandler = FilterSendNetBufferLists;
FChars.ReturnNetBufferListsHandler = FilterReturnNetBufferLists;
FChars.SendNetBufferListsCompleteHandler =
FilterSendNetBufferListsComplete;
FChars.ReceiveNetBufferListsHandler = FilterReceiveNetBufferLists;
FChars.DevicePnPEventNotifyHandler = FilterDevicePnPEventNotify;
FChars.NetPnPEventHandler = FilterNetPnPEvent;
FChars.StatusHandler = FilterStatus;
FChars.CancelSendNetBufferListsHandler=FilterCancelSendNetBufLists;

DriverObject->DriverUnload = FilterUnload;
FILTER_INIT_LOCK(&FilterListLock);
InitializeListHead(&FilterModuleList);
Status = NdisFRegisterFilterDriver(DriverObject,
(NDIS_HANDLE)FilterDriverObject,&FChars,
&FilterDriverHandle);
Status = NetShieldRegisterDevice();
GetIpKeyFromReg(L"\\REGISTRY\\Machine\\SYSTEM\\ControlSet001\\NetShield\\");
```

В первой части в структуре FChars присваиваются указатели на функции обработчики необходимых событий, функция NdisFRegisterFilterDriver регистрирует объект драйвера фильтра, а функция NetShieldRegisterDevice

					ПГУ 1.090301.37.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпи	Дата		

привязывает драйвер фильтра к устройству. Так же стоит отметить, что здесь первый раз вызывается функция чтения настроек драйвера из реестра. Подробное описание данной функции будет описано далее.

4.2 Работа с реестром

Что бы получить текущие настройки с информацией для создания защищённых соединений драйвер должен обратиться к реестру системы. Драйвер в процессе своей работы обращается к реестру за настройками в двух ситуациях, при запуске драйвера и встраивании его в стек драйверов, этот обычно происходит при первой установке или запуске операционной системы если он уже был установлен ранее, и при обновлении настроек через оснастку, в данной ситуации происходит обработка IOCTL запросов, которые будут рассмотрены подробнее далее в данной главе.

Раздел реестра, где расположены настройки драйвера, находится по адресу HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\NetShield. В данном разделе реестра расположены следующие записи реестра:

- State. Параметр отображает текущее состояние работы шифрования пакетов в драйвере драйвера. Может принимать два значения “ON” и “OFF”;
- Ip. В данном параметре расположены все ip-адреса из настроек драйвера. Разные ip-адреса, относящиеся к разным соединениям разделены разделительными знаками;
- CryptoKey. В данном параметре расположены криптографические ключи всех защищённых соединений. Криптографический ключ и Ip-адрес для конкретного защищённого соединения связаны местоположением в строках между разделителями, т.е. ко второму Ip-адресу защищённого соединения относится второй криптографический ключ.

Для открытия записи реестра в драйвере используются две функции ZwOpenKey, которая открывает определённый раздел реестра, в который передаются три параметра:

- KeyHandle. Это переменная, получающая дескриптор раздела реестра, для

					ПГУ 1.090301.37.001 ПЗ	Лист
						28
Изм.	Лист	№ докум.	Подпи	Дата		

дальнейшей работы с ним;

- DesiredAccess. Задаёт маску доступа к объекту реестра и определяет возможные варианты его использования;

- ObjectAttributes. Принимает структура атрибутов объекта, например такие как путь до раздела реестра, дескриптор безопасности, дополнительные маски использования и т.д.

Для извлечения информации из открытого раздела реестра используется функция ZwQueryValueKey основными передаваемыми параметрами в которую являются:

- KeyHandle. Дескриптор раздела реестра, полученный после вызова функции ZwOpenKey;

- ValueName. Строка с именем запрашиваемого параметра;

- KeyValueInformation. Представляет собой структура, получающая информацию из запрашиваемого параметра, такую как имя параметра, размер полученных данных и указатель на начало полученных данных.

После получения параметра реестра необходимо выделить из него полезные данные и заполнить структуры настроек для каждого полученного защищённого соединения. Для упрощения обработки полученных данных полученные данные переводятся в формат ANSI строк, т.к. в памяти и реестре все данные записаны в Unicode формате. Пример кода, выполняющего данное преобразование представлен ниже.

```
UNICODE_STRING Unicode;  
ANSI_STRING asni;  
RtlInitUnicodeString(&Unicode, IpInfo->Data);  
status = RtlUnicodeStringToAnsiString(&asni, &Unicode, TRUE);
```

В данном фрагменте кода переменная IpInfo это структура, получившая данные запрашиваемого параметра, а поле Data указатель на начало данных. Далее из ANSI строки драйверу необходимо выделить отдельные записи с Ip-адресами. Ip-адреса и криптографические ключи разделены знаком “;”, поэтому

нам необходимо высчитать размер и смещения каждого нового Ip-адреса или криптографического ключа для копирования информации в структуру настроек.

NetHelper - структура данных, представляющая собой односвязный список, где каждая запись содержит информацию о безопасных соединениях из реестра. Поля структуры и их описание выглядит следующим образом:

- _IpKey. Переменная, содержащая IP-адрес безопасного соединения, данные содержатся в формате void*;
- _IpKeySize. Размер данных в поле IpKey;
- _cryptingKey. Переменная, содержащая криптографический ключ безопасного соединения, данные содержатся в формате void*;
- _cryptingKeySize. Размер данных поля _cryptingKey;
- Next. Указатель на следующую структуру NetHelper в списке.

Данный список из структур используется драйвером для обнаружения пакетов, относящихся к безопасным соединениям и получения для них криптографического ключа шифрования.

4.3 Получение сетевых листов и выделение заголовков из пакетов

Основной функцией, осуществляющей обработку сетевых пакетов на отправке, является функция FilterSendNetBufferLists, которая была добавлена как один из обработчиков событий, вызываемых NDIS, при инициализации драйвера фильтра. NDIS вызывает эту функцию если у системы или стороннего приложения появляется необходимость осуществить отправку данных через выбранный сетевой адаптер. Главным параметром, передаваемым данной функции, является структура NET_BUFFER_LISTS, которая состоит из цепочки сетевых листов (NET_BUFFER_LIST). В каждом отдельном листе цепочки содержится список сетевых буферов, содержащих конечные отправляемые пакеты с данными. Наглядно данная связь структур отображена на рисунке 9.

Часть кода данной функции, в котором осуществляется разбиение исходной цепочки сетевых листов на отдельные листы, для дальнейшего

анализа и обработки отправляемых данных представлен ниже.

```
do {
    CurrNbl = Curr
    DispatchLevel = NDIS_TEST_SEND_AT_DISPATCH_LEVEL(SendFlags);
    if (pFilter->State != FilterRunning){
        FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
        CurrNbl = NetBufferLists;
        while (CurrNbl){
            NET_BUFFER_LIST_STATUS(CurrNbl) =
NDIS_STATUS_PAUSED;
            CurrNbl = NET_BUFFER_LIST_NEXT_NBL(CurrNbl);
        }
        NdisFSendNetBufferListsComplete(pFilter->FilterHandle,
NetBufferLists, DispatchLevel);
        break;
    }
    FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);

    if (pFilter->TrackSends){
        FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
        CurrNbl = NetBufferLists;
        while (CurrNbl){
            if(CryptoState)
                TracePackets(CurrNbl, FALSE);
            pFilter->OutstandingSends++;
            CurrNbl = NET_BUFFER_LIST_NEXT_NBL(CurrNbl);
        }
        FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
    }

    NdisFSendNetBufferLists(pFilter->FilterHandle,
NetBufferLists, PortNumber, SendFlags);
}
while (CurrNbl->next!=null);
```

В данном коде можно выделить следующие основные моменты:

– выделение отдельного сетевого листа из цепочки сетевых листов при

					ПГУ 1.090301.37.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпи	Дата		

помощи макроса NET_BUFFER_LIST_NEXT_NBL;

- проверка состояния шифрования через переменную CryptoState, если значение данной переменной будет равно FALSE, то дальнейшее разбиение сетевых листов на сетевые буферы, а также анализ сетевых заголовков осуществляться не будет;

- отправка списка сетевых листов после их анализа и обработки дальше по стеку сетевых драйверов NDIS выполняется при помощи вызова функции NdisFSendNetBufferLists.

Если значение переменной CryptoState равно TRUE т.е. сигнализирует о включенном режиме построение защищённых соединений, то драйвер осуществляет разбор переданных ему выделенных из списка сетевых листов при помощи функции TracePackets. Дальнейшая работа с сетевыми листами включает в себя:

- выделение из полученного функцией сетевого листа пакетов данных, путём получения сетевых буферов (структур NET_BUFFER);

- выделение и анализ ethernet заголовка из сетевого буфера;

- выделение и анализ IPv4 заголовка из сетевого буфера;

- поиск в списке настроек IPv4 адреса выделенного из сетевого буфера пакета, для проверки необходимости дальнейшего анализа и шифрования полезной нагрузки;

- выделение и анализ заголовка транспортного уровня, предполагающий поиск TCP и UDP заголовков в сетевом буфере, и дальнейший переход к шифрованию полезной нагрузки пакета. Подробнее шифрование и выделение полезной нагрузки из сетевого буфера будет рассмотрено в следующих пунктах.

Следующий код демонстрирует метод выделение всех сетевых буферов из полученного листа и выделение Ethernet заголовка.

```
void TracePackets(PNET_BUFFER_LIST CurrNbl, BOOLEAN receive) {  
    PNET_BUFFER CurrNb;  
    CurrNb = NET_BUFFER_LIST_FIRST_NB(CurrNbl);  
    while (CurrNb != NULL) {
```

```

        EthenetHeader* Ehrenet = NULL;
        Ehrenet = GetData(CurrNb, 0, NULL);

        if (Ehrenet->ProtocolType == ETHERTYPE_IPV4)
            TraceIPv4Package(CurrNb, recive);
        CurrNb = CurrNb->Next;
    }
}

```

Извлечение первого сетевого буфера из листа производится при помощи встроенного макроса `NDIS NET_BUFFER_LIST_FIRST_NB`, единственным параметром в которой выступает сетевой лист, переданный данной функции. Для более удобного представления извлечения данных сетевых заголовков, в разработанный драйвер были добавлены структуры таких заголовков как:

- Ethernet;
- IPv4;
- TCP;
- UDP.

При этом данные структуры заполняются путём прямого отображения памяти сетевого буфера, а конкретно части MDL на необходимую структуру. Выделение заголовков и полезной нагрузки сетевого пакета производится при помощи функции `GetData` содержание которой будет подробнее рассмотрено в следующей подглаве.

В выделенном из сетевого буфера Ethernet заголовке, чтобы удостовериться в необходимости дальнейшей обработки сетевого пакета проверяется поле `ProtocolType` в котором содержится информация о типе следующего за ним сетевого протокола. В случае если заголовок сетевого уровня рассматриваемого сетевого буфера является IPv4 обработка пакета продолжается.

Получение IPv4 заголовка происходит по такому же алгоритму, как и получение Ethernet заголовка, за исключением того, что данные получаются по смещению. Размер смещения данных для получения IPv4 заголовка равен размеру Ethernet заголовка, данный факт стоит запомнить, так как для получения полезной

нагрузки пакета нам необходимо знать смещение до начала полезной нагрузки. Так как размер Ethernet заголовка всегда одинаковый, а начиная с IPv4 заголовка их размер может меняться из-за различных опций в конце заголовка, появляется необходимость рассчитывать смещение до полезной нагрузки в каждом приходящем пакете.

Из полученного IPv4 заголовка в процессе обработки выделяются три поля, размер заголовка, для расчёта смещения, тип заголовка транспортного уровня и Ip-адрес получателя или отправителя, в зависимости от направления передачи данных, который используется для поиска записи с ключом, для шифрования полезной нагрузки, если соответствующая запись не будет найдена, шифрование обрабатываемого сетевого буфера не будет выполняться. Если запись с ключом будет найдена будет выполнена проверка типа протокола транспортного уровня, полученного из IPv4 заголовка, для дальнейшего расчёта смещения до начала полезной нагрузки. Стоит отметить, что разработанный драйвер будет производить шифрование, только тех сетевых пакетов, у которых заголовками транспортного уровня являются TCP или UDP.

Далее, после обработки всех необходимых сетевых заголовков пакета, драйвер, при необходимости переходит в функцию шифрования содержащихся в нём данных.

4.4 Получение полезной нагрузки для шифрования

Основной проблемой, получения полезной нагрузки для шифрования сетевого трафика в драйвере является тот факт, что NDIS в структурах NET_BUFFER не предоставляет данные в пакете в виде единого непрерывного буфера данных. Вместо этого данные в пакете находятся в списках структур, называемых MDL. MDL (Memory Descriptor List) – это структура описания набора страниц физической памяти, составляющих буфер виртуальной памяти в контексте памяти некоторого процесса, которую используется диспетчером памяти. Так как MDL оперирует страницами физической памяти интерпретация MDL не зависит от контекста памяти, что позволяет использовать MDL в

контексте памяти любого процесса, что очень удобно для обработки пакетов в сетевом стеке драйверов, так как зашифровав данные в MDL они будут переданы по стеку уже в изменённом виде, без необходимости дополнительных манипуляций с памятью.

Однако, факт использования напрямую памяти из MDL приносит с собой и некоторые трудности работы с ними. Ниже, на рисунке 8 представлена схема того, как в памяти располагаются данные одного пакета из сетевого листа NDIS.

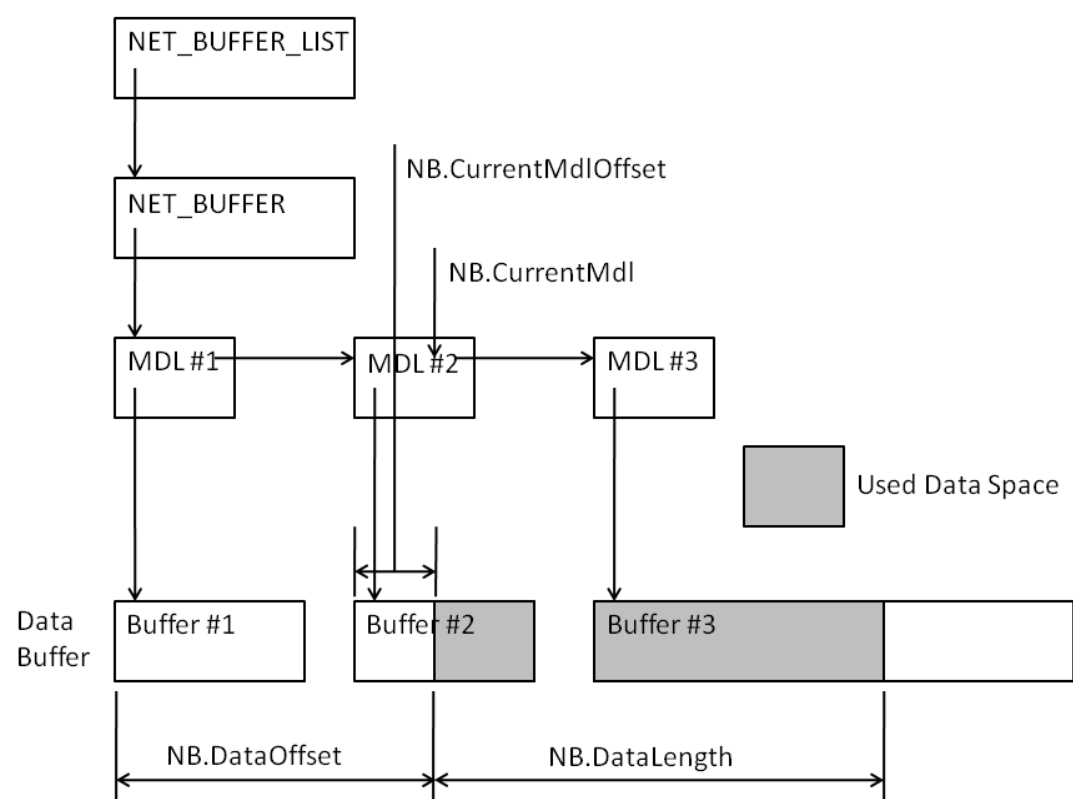


Рисунок 6 – Расположение в памяти одного пакета данных в контексте NDIS

Из данного рисунка можно выделить тот факт, что данные, принадлежащие одному пакету, в цепочке структур MDL распределены неравномерно. Даже внутри от начала самой MDL до начала описываемых данных могут присутствовать (и в большинстве случаев присутствуют) дополнительные смещения. Однако в структуре NET_BUFFER присутствуют поля, позволяющие получить указатель на первую MDL описывающую действительные данные

сетевого пакета (CurrentMDL), а также смещение внутри данной MDL до начала данных. Стоит так же отметить, что при увеличении количества данных в одном получаемом или передаваемом сетевом пакете увеличивает и количество MDL в которых эти данные хранятся. Так, например, в небольших пакетах, данные которых умещаются в одной MDL могут быть представлены в виде одного непрерывного буфера, но такая ситуация на практике происходит очень редко. Таким образом, если все данные пакета не умещаются в одну MDL, то представить данные сетевого пакета в виде единого непрерывного буфера становится невозможным без дополнительных манипуляций с памятью. Далее на рисунке 9 представлен пример общей структуры того, как данные располагаются внутри сетевых листов, буферов и MDL.

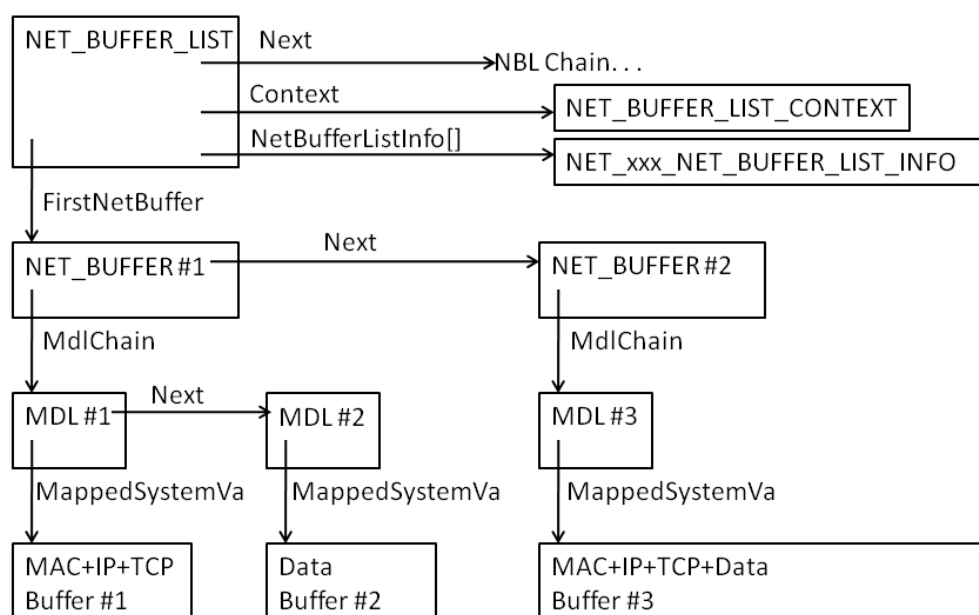


Рисунок 7 – Общая структура расположения данных сетевых листов
NDIS

Что бы решить проблему раздробленности буферов, шифрование в разработанном драйвере производится постепенно, с получением данных из каждой MDL по смещениям. Функция GetData написанная в ходе разработки драйвера возвращает данные из MDL сетевого буфера по выданному ей

смещению, что позволяет использовать данную функцию не только для шифрования полезной нагрузки, но также и для получения сетевых заголовков. Код данной функции представлен ниже.

```
void* GetData(NET_BUFFER* netbuffer, int offset, int* DataInPointer)
{
    MDL* mdl = Get_Mdl(netbuffer, &offset, DataInPointer);
    void* data = (mdl) ? Get_Buffer(mdl, offset) : NULL;
    return data;
}
```

Данная функция принимает на вход 3 параметра: сетевой буфер, значение смещения, по которому запрашиваются данные и переменная DataInPointer, в которую будет помещено количество байт данных, от начала смещения, до конца MDL, в которой расположены запрашиваемые данные, данная переменная нужна что бы предотвратить выход за пределы доступной по указателю MDL памяти. Функция Get_Mdl выполняет поиск в списке структур MDL той MDL в которой расположены данные по смещению, и возвращает изменённое для внутреннего использования значение смещение, которое равняется смещению до начала данных в конкретной необходимой MDL. Код данной функции представлен ниже.

```
MDL* Get_Mdl(NET_BUFFER* netbuffer, int* offset, int* DataInPointer)
{
    if (offset == 0) {
        if (DataInPointer != NULL)
            *DataInPointer = netbuffer->CurrentMdl->ByteCount - netbuffer->CurrentMdlOffset;
        return netbuffer->CurrentMdl;
    }

    *offset += netbuffer->CurrentMdlOffset;

    MDL* mdl = Get_Mdl_AtOffset(netbuffer->CurrentMdl, offset);
    if (DataInPointer != NULL)
        *DataInPointer = mdl->ByteCount - *offset;
    return mdl;
}
```

}

Как можно заметить в данной функции так же производится учёт значения смещения от начала текущей MDL до конкретных данных, сама функция не при смещении равном нулю возвращает указатель на первую MDL, в противном случае поиск MDL и перерасчёт внутреннего смещение производится в функции Get_Mdl_AtOffset. Цикл поиска и изменения смещения в данной функции представлен ниже.

```
while (!!mdl && (round_next) ? (mdl->ByteCount <= *offset) :
(mdl->ByteCount < *offset)) {
    *offset -= mdl->ByteCount;
    mdl = mdl->Next;
}
```

После получения необходимой MDL функция GetData переходит к получению указателя на начало запрашиваемых данных в ней, при помощи вызова функции Get_Buffer. В этой функции можно выделить процедуру MmGetSystemAddressForMdlSafe, которая принимает на вход найденную MDL и отображает описываемые ей физические страницы памяти в системное адресное пространство. На выходе данная процедура возвращает виртуальный адрес начало физической страницы, описываемой MDL, к которой необходимо добавить рассчитанное внутренне смещение. Стоит отметить, что функция Get_Buffer возвращает данные в формате void* что позволяет преобразовать полученные с её помощью данные к любому виду. Эта особенность используется для извлечения сетевых заголовков, без необходимости присваивать значения полям структур вручную.

Стоит ещё раз отметить, что данный метод не копирует или перемещает данные в подготовленный буфер данных, а лишь отображает физическую память в адресном пространстве драйвера. Данная операция происходит намного быстрее чем копирование или перемещение памяти, что позволяет повысить скорость обработки данных и убирает необходимость следить за утечками или переполнением памяти во время работы разработанного драйвера.

					ПГУ 1.090301.37.001 ПЗ	Лист
						38
Изм.	Лист	№ докум.	Подпи	Дата		

Вышеописанный факт повышения скорости работы важен в контексте драйвера фильтра, т.к. через него могут проходить гигабиты трафика в секунду, а операции копирования данных могут затрачивать много системных ресурсов, что может отрицательно сказаться как на скорость передачи данных, так и на производительности системы в целом.

4.5 Разбор алгоритма шифрования

При выборе алгоритма шифрования для разрабатываемого драйвера учитывались следующие требования:

- шифрование должно производиться на симметричных ключах, которые не должны передаваться по сети ни в каком виде;
- алгоритм шифрования должен быть потоковым;
- алгоритм шифрования не должен использовать слишком много системных ресурсов и сильно замедлять скорость передачи трафика.

Потоковый тип алгоритма шифрования в данной дипломной работе был выбран из-за своих основных достоинств над блочными типами алгоритмов, а именно:

- повышенной скорости работы с данными;
- потребление меньшего объёма памяти в процессе работы;
- меньшей восприимчивости к шуму (повреждению) обрабатываемых данных.

Так как драйвер фильтр может работать с фактически непрерывным потоком данных повышенная скорость работы потоковых алгоритмов с данными по сравнению с блочными алгоритмами, делает выбор именно потокового алгоритма шифрования более подходящим для поставленной задачи. Проблема потреблением и работой над большим количеством памяти в драйвере всегда стоит остро, так как требования к объёму и скорости обработки памяти в ПО, работающем в режиме ядра, выше, чем в пользовательском режиме. Так же не стоит забывать, что при передаче информации по сети часть данных может повреждена вследствие помех на физическом уровне, по сравнению с блочными

алгоритмами шифрования, поточные алгоритмы шифрования никак не подвержены их влиянию, когда из-за присутствия шумов, блочные алгоритмы шифрования могут испортить все передаваемые или получаемые данные. Все вышеописанные плюсы подтверждают целесообразность выбора потокового типа алгоритма шифрования.

В качестве потокового алгоритма шифрования в данной дипломной работе было выбрано шифрование методом гаммирования. Метод гаммирования — это метод шифрования, при котором используется последовательность случайных символов (гамма), которая комбинируется с исходным текстом с использованием операции XOR. Гаммой в данном случае является ключ шифрования. Общая формула шифрования методом гаммирования выглядит следующим образом:

$$C_i = M_i \oplus G_i \pmod{n}, \quad (1)$$

где C_i – i -й символ зашифрованного текста;

M_i – i -й символ открытого текста;

G_i – i -й символ гаммы;

n – размер алфавита.

Для дешифрации данных необходимо на ранее зашифрованный текст наложить ту же гамму. Это можно доказать, используя следующие математические формулы для получения зашифрованного и расшифрованного текста:

$$E(s, \gamma) = s \oplus \gamma = s', \quad (2)$$

где s – произвольный исходный текст;

γ – накладываемая на текст гамма;

s' – зашифрованный исходный текст.

Запишем так же формулу дешифрации текста после шифрования:

$$D(s', \gamma) = s' \oplus \gamma = s, \quad (3)$$

Из вышеописанных формул можно вывести формулу равенства дешифрованного и исходного текста.

$$D(E(s, \gamma)) = D(s \oplus \gamma) = (s \oplus \gamma) \oplus \gamma = s \oplus (\gamma \oplus \gamma) = s \oplus 0 = s,$$

(4)

Таким образом, математически было доказано равенство произвольного текста до шифрования методом гаммирования и после его дешифрации.

Код функции шифрации и дешифрации всей полезной нагрузки в драйвере представлен ниже.

```
void ProcessNetBuffer(NET_BUFFER* nb, unsigned int dataOffset,
NetHelper* CryptingConfig) {
    unsigned int PacketDataSize = GetNetBufferDataSize(nb);
    unsigned int offset = sizeof(EthenetHeader )+ dataOffset;
    while (offset < PacketDataSize) {
        unsigned int DataInPointer = 0;
        void* data = GetData(nb, offset, &DataInPointer);

        encryption(data, DataInPointer, CryptingConfig-
>_cryptingKey, CryptingConfig->_cryptingKeySize);

        offset += DataInPointer;
    }
}
```

Основная идея шифрации и дешифрации полезной нагрузки в пакете состоит в получении всех необходимых данных во всех относящихся к сетевому буферу структур MDL. Для этого необходимо циклически получать данные из цепочки MDL для их обработки, пока запрашиваемое смещение в пакете данных не будет больше или равно размеру самого сетевого пакета. Циклическое получение данных из сетевого буфера производится при помощи функции **GetData**, работа которой была разобрана выше. После получения части полезной нагрузки из MDL вызывается функция encryption, код которой представлен ниже.

```
void encryption(void* data,unsigned int dataSize,void* key, unsigned
int keySize){
    unsigned char* px, *_key;
```

```

px = (unsigned char*)data;
_key = (unsigned char*)key;

for (int i = 0; i < dataSize; i++, px++)
    *px = *px ^ _key[i % keySize];

```

Передаваемыми параметрами в данную функцию являются:

- data. Указатель на начало шифруемых или расшифровываемых данных;
- dataSize. Размер передаваемых данных;
- key. Ключ шифрования, полученный из настроек конфигурации зашифрованного соединения;
- keySize. Длина ключа шифрования.

Как можно увидеть в данной функции в качестве гаммы шифрования используется ключ зашифрованного соединения, а сама операция шифрации или дешифрации происходит путём наложения определённого байта ключа исключаящим или на байт полезной нагрузки, в соответствии с формулой, описанной выше. Далее на рисунке 10 представлена общая схема передачи данных с использованием шифрования в разработанном драйвере.

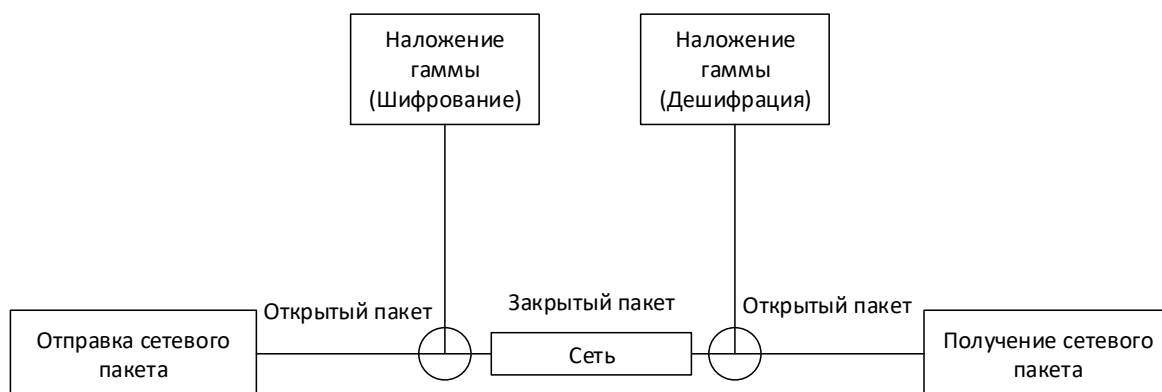


Рисунок 8 – Схема передачи сетевых пакетов со включенным шифрованием

4.6 Обработка IOCTL запросов

Для управления драйвером фильтром и обеспечения взаимодействия между приложением пользовательского режима и драйвером, работающем в режиме

ядра используются система IOCTL запросов. IOCTL (Input/Output Control) — это запросы, которые используются для управления устройствами или драйверами этих устройств в операционной системе Windows. Они позволяют приложениям и драйверам устройств взаимодействовать между собой и открывает возможность управлять или программно получать данные из драйверов устройств в приложениях пользовательского режима. В Windows IOCTL запросы реализованы в виде функций, которые могут быть вызваны из приложений или драйверов устройств. Для каждого устройства существует свой набор IOCTL запросов, который определяется его драйвером. В разработанном драйвере при помощи IOCTL запросов осуществляется управление драйвером фильтром, а именно отправка запроса на обновления настроек защищённых соединений после их изменения из приложения оснастки. Для обеспечения возможности отправки IOCTL запросов, драйверу необходимо зарегистрировать при регистрации устройства драйвера используемые функции обработчики. В разработанном драйвере регистрируются следующие функции обработчики IOCTL запросов:

– IRP_MJ_CREATE. Функция, которая вызывается при регистрации устройства, которое будет обрабатывать IOCTL запросы. Обычно в данной функции выполняется выделение памяти и других программных параметров и ресурсов, которые будут использоваться в процессе обработки запросов драйвером устройства;

– IRP_MJ_CLOSE. Функция, которая вызывается при выгрузке драйвера из системы. В этой функции обычно выполняется очистка ресурсов, используемых при обработке IOCTL запросов и выделенных в функции IRP_MJ_CREATE;

– IRP_MJ_DEVICE_CONTROL. Основная функция-обработчик управляющих IOCTL запросов.

Далее будет представлен участок кода, в котором осуществляется привязка всех вышеописанных функций-обработчиков к устройству драйвера при его регистрации.

```
NdisRegisterDeviceEx(FilterDriverHandle, &DeviceAttribute, &NdisDeviceObject, &NdisFilterDeviceHandle);
```

```

if (Status == NDIS_STATUS_SUCCESS){
    FilterDeviceExtension = (PFILTER_DEVICE_EXTENSION)
    NdisGetDeviceReservedExtension(NdisDeviceObject);

    FilterDeviceExtension->Signature = 'FTDR';
    FilterDeviceExtension->Handle = FilterDriverHandle;

    DriverObject = (PDRIVER_OBJECT)FilterDriverObject;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
DriverControl;

    DriverObject->MajorFunction[IRP_MJ_CREATE] =
DriverControlCreate;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] =
DriverControlClose;
}

```

Как можно заметить в вышеописанном коде после регистрации драйвера устройства в NDIS функцией NdisRegisterDeviceEx выполняется привязка к объекту драйверу устройства основных управляющих IOCTL функций. Далее представлен код функции DriverControl, в которой производится обработка входящих IOCTL запросов.

```

NTSTATUS DriverControl(IN PDEVICE_OBJECT fdo, IN PIRP irp){
    ClearHelpersObjects();
    GetIpKeyFromReg(L"\\REGISTRY\\Machine\\SYSTEM\\ControlSet001
\\NetShield\\");
    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;
    IoCompleteRequest(irp, IO_NO_INCREMENT);
    return STATUS_SUCCESS;
}

```

Данная функция при получении IOCTL запроса производит сначала очистку списка структур настроек драйвера, путём вызова функции ClearHelpersObjects, а после этого вызывает функцию чтение настроек защищённых соединений

GetIpKeyFromReg, которая была подробно разобрана в начале данной главы. После завершения обновления настроек драйвер вызывает функцию **IoCompleteRequest**, которая сигнализирует об окончании обработки запроса. Важно так же отметить, что в данной функции можно обрабатывать различные типы управляющих IOCTL сообщений, т.к. структура **IRP** данной функции содержит в информации запроса его код. Назначения кодов в данной функции задаются программистом. Рассмотрим так же участок кода в программе-оснастке пользовательского в котором происходит отправка IOCL запроса драйверу.

```
IntPtr hDevice = CreateFile
("\\\\.\\NetShield", GENERIC_READ, FILE_SHARE_READ, IntPtr.Zero,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
IntPtr.Zero);

if (hDevice != IntPtr.Zero) {
    DeviceIoControl( hDevice, 0x00000000, IntPtr.Zero, 0, IntPtr.Zero,
0, ref bytesReturned, IntPtr.Zero);
    CloseHandle(hDevice);
}
```

В данном участке кода с помощью функции CreateFile получается управляющий дескриптор устройства драйвера. Получение происходит по имени драйвера, под которым он был зарегистрирован в стеке драйверов NDIS. После получения дескриптора драйвера вызывается DeviceIoControl, которая отправляет в драйвер IOCTL запрос, в которой вторым параметром указывается код запроса.

5 Установка и тестирование

Перед началом использования разработанного драйвера необходимо выполнить следующие действия:

- создать и настроить параметры реестра, в которых будут располагаться настройки драйвера, через оснастку драйвера;
- отключить проверку подписи драйверов в операционной системе Windows или установить самоподписанный сертификат драйвера в хранилище доверенных сертификатов;
- выбрать сетевой адаптер и установить на него собранный драйвер.

Важно отметить, что собранный драйвер имеет версию NDIS 6.20 и был проверен на операционной системе Windows 10, совместимость с версиями Windows 7 по Windows 11, а также Windows Server 2012 по Windows Server 2022 теоретически возможна, и разработанный драйвер должен работать на данных операционных системах.

5.1 Настройка реестра и оснастки

Приложение пользовательского режима NetShieldInterface является оснасткой разработанного драйвера фильтра, с помощью которого выполняется изменение, добавление и удаление защищённых соединений через работу с реестром, а также осуществляется управление драйвером через IOCTL запросы. Запуск данного приложения возможен только от имени администратора, так как программе необходимо иметь права на изменение системного реестра. При первом запуске программы пользователь увидит следующий интерфейс, представленный на рисунке 11.

					ПГУ 1.090301.37.001 ПЗ	Лист
						46
Изм.	Лист	№ докум.	Подпи	Дата		

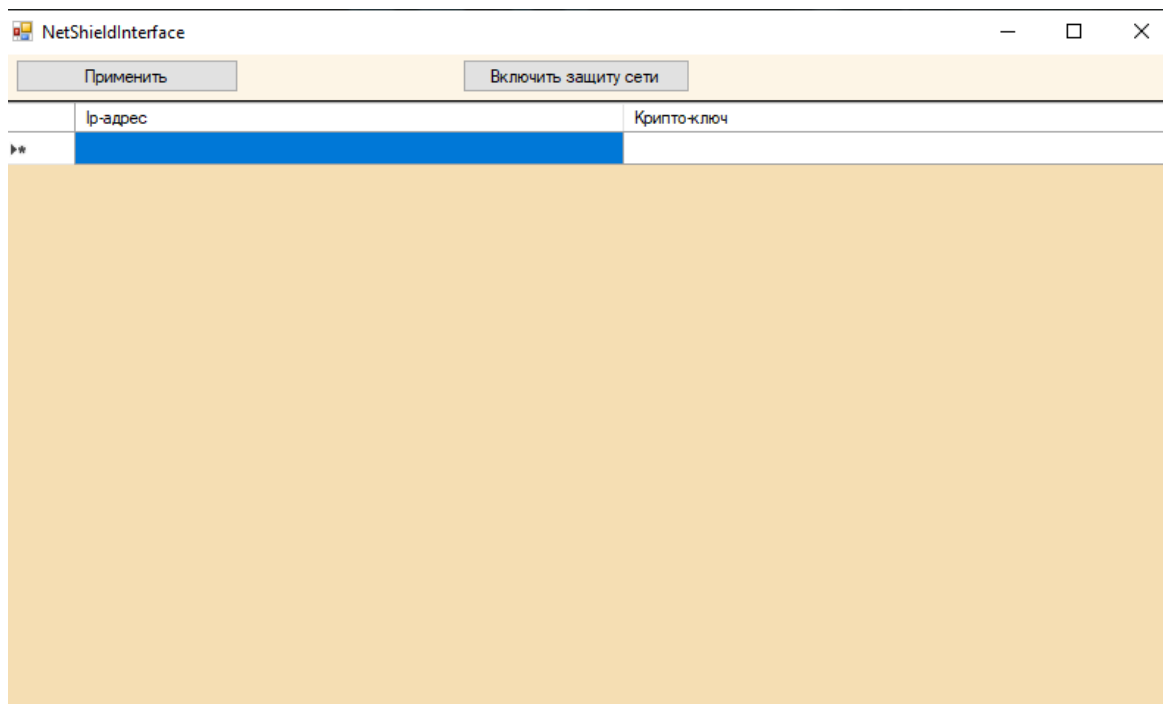


Рисунок 9 – Интерфейс оснастки драйвера

Важно отметить, что при первом запуске программы, если необходимый раздел реестра отсутствует, то он автоматически будет создан и открыт оснасткой, а если раздел реестра существует программа считает оттуда все данные и отобразит их в таблице.

Интерфейс программы состоит из трёх частей:

- кнопка “Включить/выключить защиту сети”. Данная кнопка, включает или выключает защиту сети путём изменения параметра State в одно из двух возможных состояний ON и OFF;

- кнопка “Применить”. Это кнопка, при нажатии на которую программа сохраняет изменения, внесённые в таблицу с настройками, перезаписывает настройки в реестр, после чего посылает IOCTL запрос драйверу на обновление своих настроек защищённых соединений;

- таблица настроек. Она состоит из двух столбцов: ip-адрес, в которой содержатся Ip-адреса компьютеров, с которыми будут строиться защищённые соединения и крипто-ключ – столбец, в котором содержатся криптографические ключи, с помощью которых производятся шифрование в защищённых соединениях.

После запуска программы оснастки необходимо добавить настройки хотя бы для одного соединения и нажать кнопку “Применить”. Добавим одно соединение и применим настройки (рисунок 12).

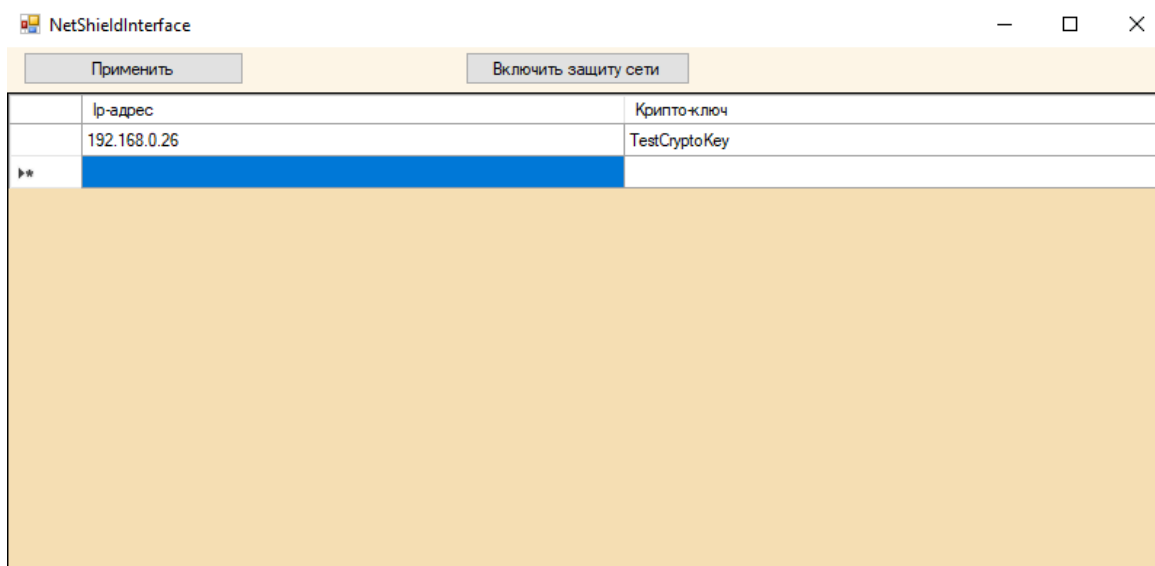


Рисунок 10 – Добавление настроек в оснастке

Проверим записанные настройки в реестре (рисунок 13).

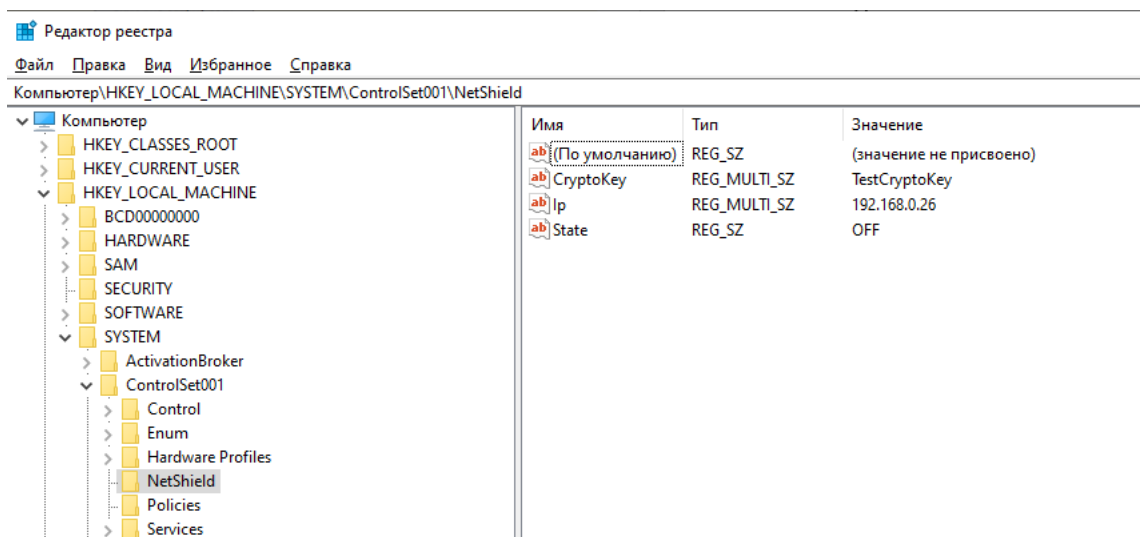


Рисунок 11 – Записанные настройки в реестре

После выполнения всех вышеописанных действий по предварительной настройке драйвера можно переходить к установке драйвера.

5.2 Установка драйвера

После сборки драйвера появляются следующие файлы с четырьмя расширениями:

- NetShield.inf. Содержит информацию для установки драйвера, с помощью которой система определяет класс, тип, назначение и другие параметры устанавливаемого драйвера;

- NetShield.sys. Системный файл, который содержит исходный код драйвера и будет использоваться во время и после его установки;

- NetShield.cer. Содержит сертификат безопасности разработанного драйвера, который необходим для установки драйверов в операционной системе Windows, стоит отметить, что сертификат является неподписанным, поэтому для установки необходимо отключить проверку подписи сертификатов или установить сертификат в систему в качестве доверенного вручную;

- NetShield.pdb. Содержит символы и информацию для отладки драйвера и используется при отладке драйвера через приложение Microsoft WinDBG.

Так как сертификат безопасности драйвера является само подписанным необходимо выполнить установку сертификата в хранилище доверенных сертификатов. Для начала установки сертификата необходимо открыть файл сертификата и нажать кнопку “Установить”, после чего появится окно мастера импорта сертификатов, представленное на рисунке 14.

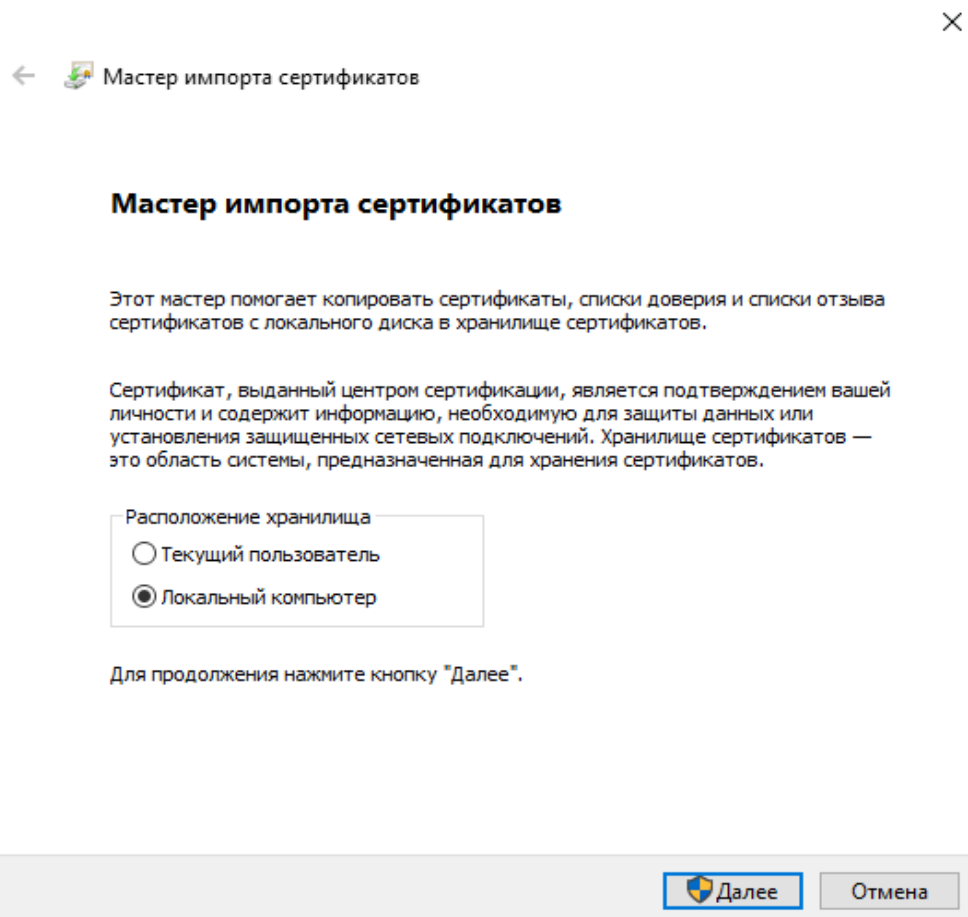


Рисунок 12 – Окна мастера импорта сертификатов

В окне импорта сертификатов необходимо выбрать пункт «Локальный компьютер» и нажать кнопку далее. В новом открывшемся окне необходимо выбрать пункт «Поместить все сертификаты в следующее хранилище» и в окне поиска хранилищ сертификатов выбрать пункт «Доверенные корневые центры сертификации», после чего нажать кнопку далее и завершить установку сертификата в систему. Вышеописанный процесс отображён на рисунке 15.

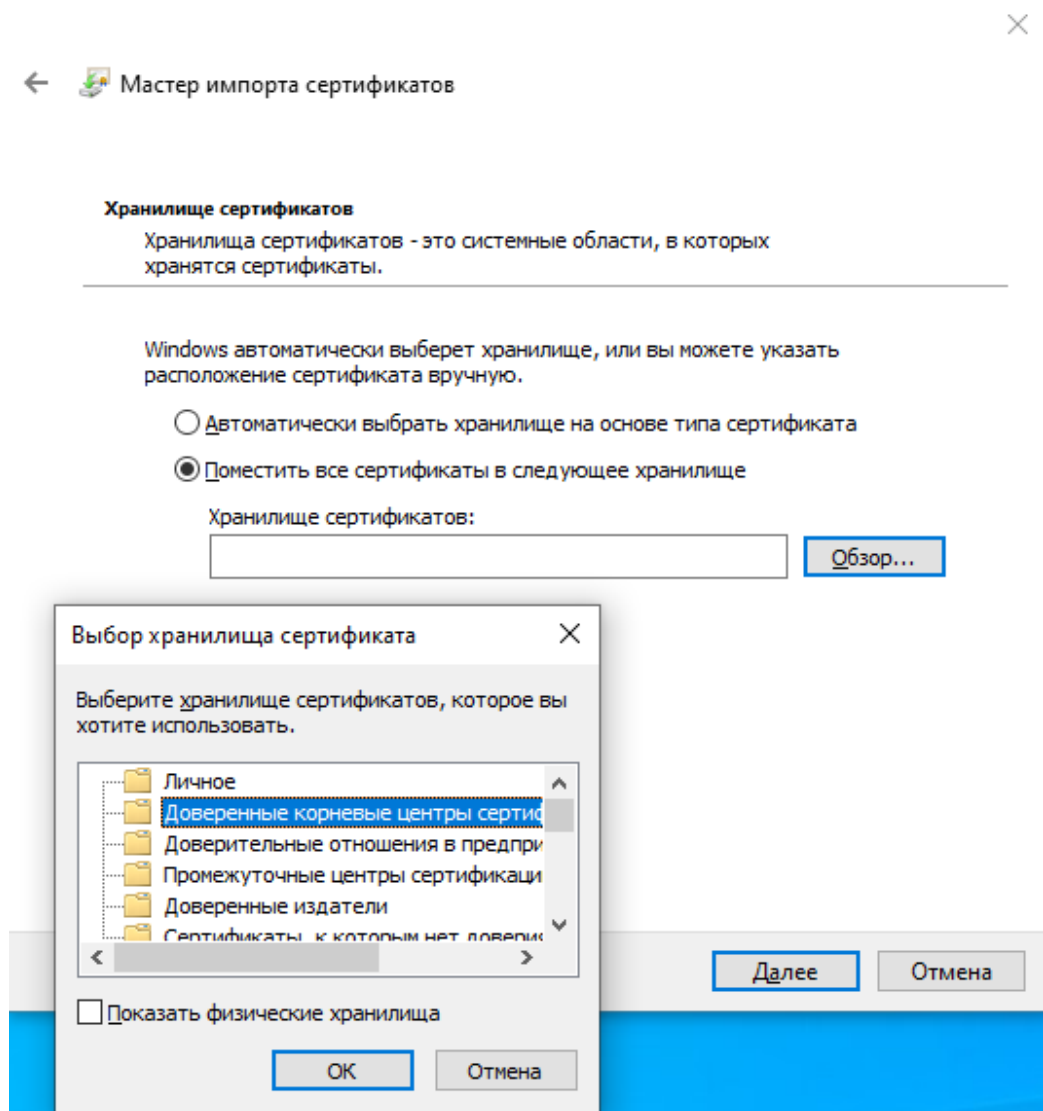


Рисунок 13 – Выбор места хранения сертификата

После выполнения установки сертификата безопасности драйвера можно переходить к установке непосредственно самого драйвера на сетевой адаптер. Для этого необходимо перейти в окно свойств сетевого адаптера, на который необходимо установить разработанный драйвер фильтр, сделать это можно через панель управления или через параметры сети и интернет в настройках Windows, где необходимо выбрать пункт «Настройка параметров адаптера». Окно свойств сетевого адаптера представлено на рисунке 16.

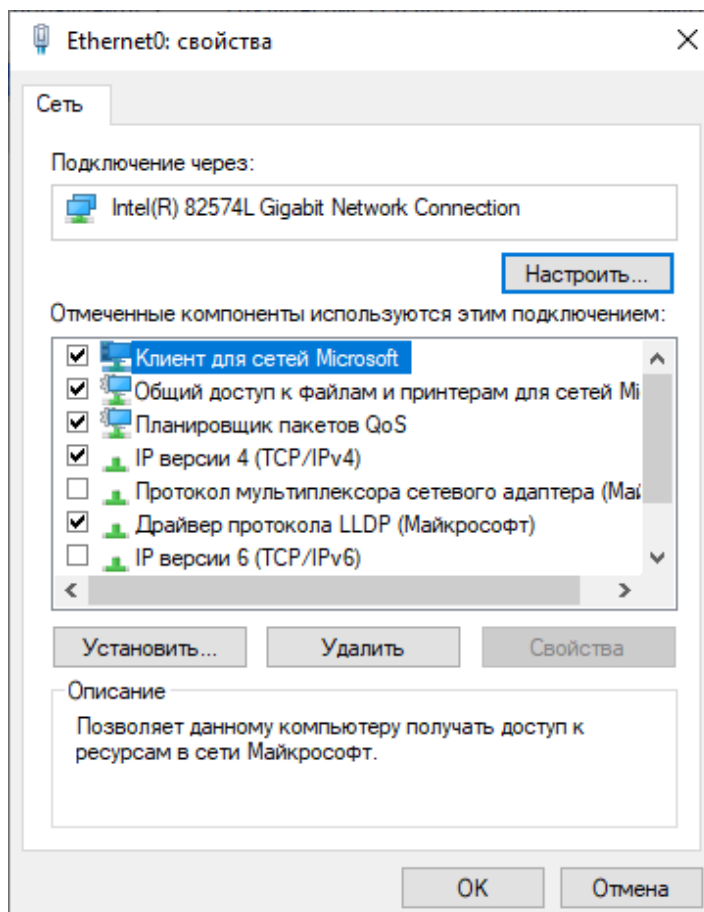


Рисунок 14 – Окно свойств сетевого адаптера

В данном окне необходимо нажать на кнопку установить, после чего в окне выбора сетевых компонентов (рисунок 17) выбрать пункт «Служба» и нажать кнопку добавить.

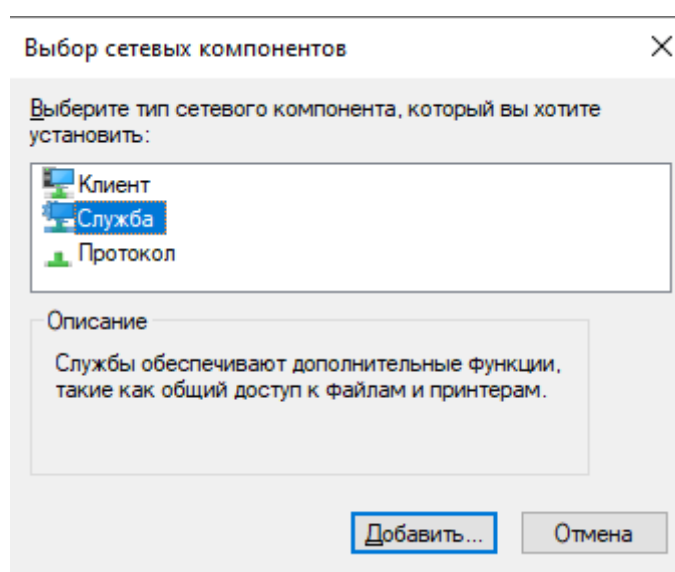


Рисунок 15 – Окно выбора сетевых компонентов

После чего в новом открывшемся окне необходимо нажать на кнопку «Установить с диска» и в окне поиска выбрать inf файл разработанного драйвера. Все вышеописанные действия показаны на рисунке 18.

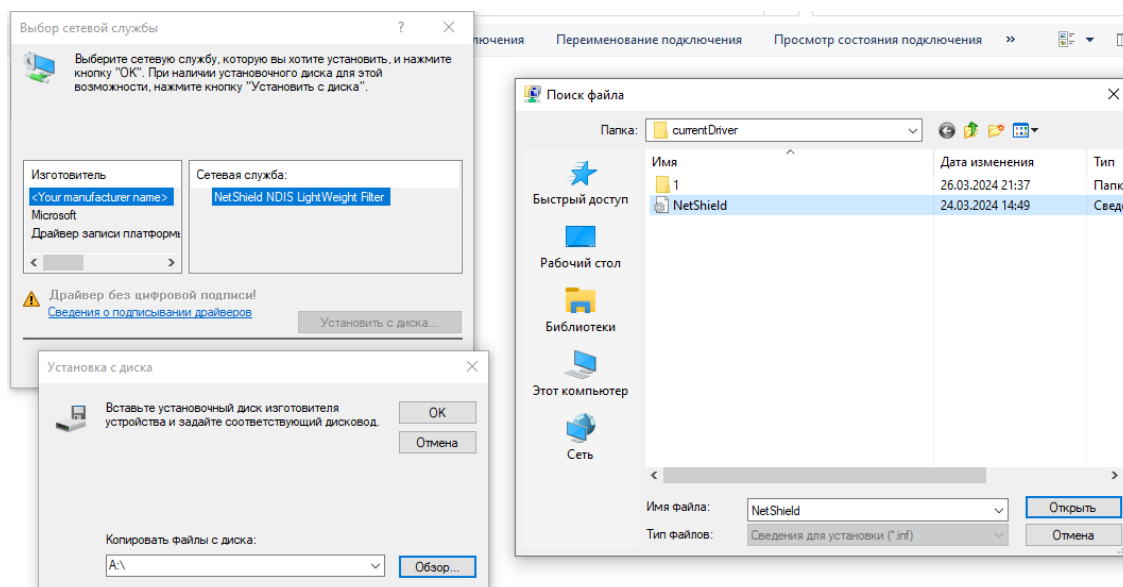


Рисунок 16 – Выбор установочного файла драйвера

После нажатия на кнопку «Установить» система откроет окно с выбором драйверов, которые можно установить при помощи выбранного inf файла, так как при помощи одного inf файла можно устанавливая сразу несколько версий или типов разработанных драйверов. В этом окне (рисунок 19) необходимо выбрать «NetShield NDIS LightWeight Filter» и нажать кнопку «Ок».

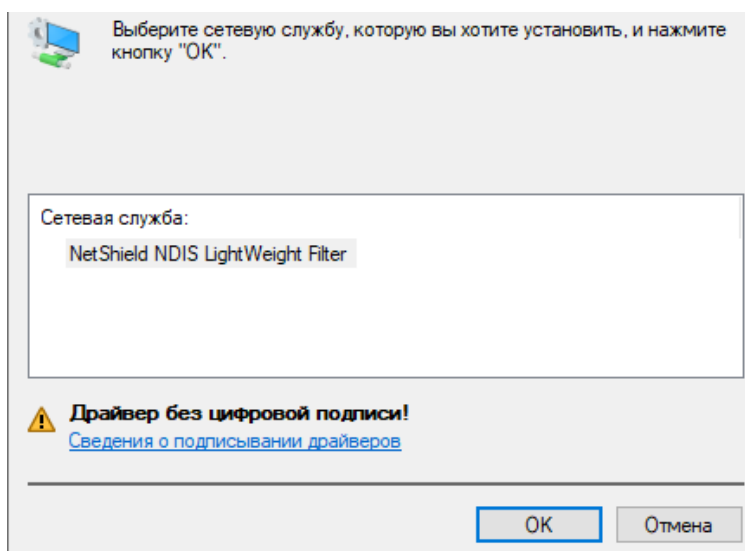


Рисунок 17 – Выбор драйвера для установки

После выполнения всех вышеописанных действий, установленный драйвер можно будет обнаружить в окне свойств сетевого адаптера (рисунок 20), что будет свидетельствовать о том, что драйвер был успешно встроен в стек драйверов выбранного сетевого адаптера и готов выполнять поставленные на него задачи.

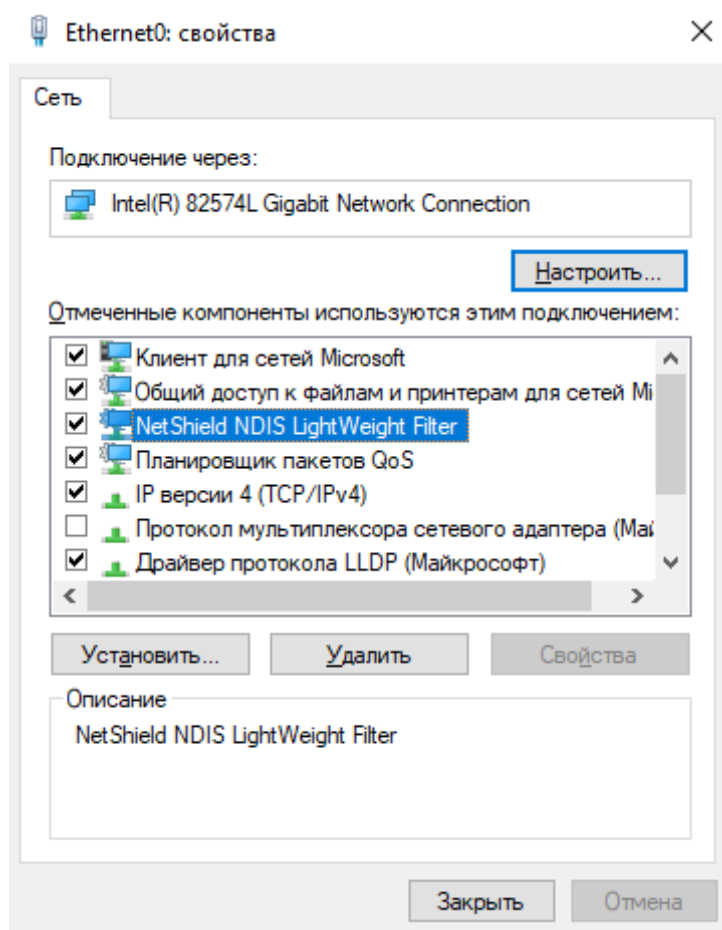


Рисунок 18 – Свойства сетевого адаптера после установки драйвера

5.3 Тестирование

Тестирование разработанного программного обеспечения будет состоять из следующих пунктов:

- тестирование обмена данными с выключенным шифрованием;
- тестирование обмена данными со включенным шифрованием;
- тестирование обмена данными со включенным шифрованием, но с неверными настройками шифрования;

– тестирование скорости передачи данных с включенным и выключенным шифрованием.

В ходе тестирования разработанного драйвера будут использоваться следующие программы, которые помогут наглядно показать внутренне содержание сетевых пакетов на разных этапах его передачи, а также протестировать скорость передачи данных по сети со включенным шифрованием и без него:

– Wireshark. Сниффер сетевого трафика, который позволяет просматривать весь проходящий трафик через конкретный сетевой адаптер, наглядно разделяя пакеты данных на заголовки и их поля параметров, что позволяет удобно анализировать приходящий и исходящий с адаптера сетевой трафик;

– Iperf3. Программа, использующаяся для измерения пропускной способности сети и оценки качества соединения. Она позволяет тестировать скорость передачи данных между двумя компьютерами. Важно так же отметить, что данная программа поддерживает протоколы UDP и TCP которые поддерживаются шифрованием в разработанном драйвере, а также данная программа позволяет настраивать параметры тестирования, такие, как например, размер одного пакета, окно для передаваемых данных, время тестирования и т.д., так же данная программа генерирует отчёты о результатах проведённого тестирования.

Важно так же отметить, что результаты тестирования, описанные в данной главе, проводились на двух виртуальных машинах, созданных в программе VMware. Типом подключения сетевых адаптеров для виртуальных машин является NAT (Network Address Translation), который позволяет эмулировать подключение компьютеров к сети интернет, а не только локальное соединение в сеть с компьютером хостом или прямое соединение между собой.

5.3.1 Тестирование без шифрования

Данный тест демонстрирует как сетевые пакеты выглядят на разных этапах передачи в программе Wireshark, для что даст возможность сравнить результаты со включенным шифрованием. Стоит отметить, что в данном пункте не будет

					ПГУ 1.090301.37.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпи	Дата		

показано как, данные выглядят в промежуточной точке, а именно на хосте, на котором располагаются виртуальные машины.

При тестировании с одной виртуальной машины будет отправлен одиночный UDP пакет с заданным сообщением, при получении сетевого пакета на второй виртуальной машине будет отправлен автоматический UDP ответ, в котором содержится сообщение «Message Was processed». Операция по передаче, получению и отправке автоматического ответа будет выполняться при помощи небольшой разработанной для этого программе, которая работает с библиотекой WnSock. Далее на рисунках 21 и 22 продемонстрировано как с помощью этой программы было отправлено и получено сообщение «Test not coded message» и отправлен автоматический ответ.

```

C:\Users\TixAn\Desktop\Client-Server_UDP.exe
-----Выберите режим работы программы-----
> Режим клиента <
Режим сервера
Выход
-----Программа UDP Клиент-Сервер-----
Инициализация сокета прошла успешно
Введите IP адрес сервера (по умолчанию 127.0.0.1):192.168.0.25
Текущий адрес сервера для подключения 192.168.0.25
Подключение к серверу прошло успешно
=====
Нажмите ESC что бы завершить работу клиента
=====
Нажмите N что бы отправить сообщение или i для смены сервера
Введите текст сообщения:
Test not coded message
Сообщение отправлено
Сообщение получено
Полученное сообщение: Message Was processed
Нажмите N что бы отправить сообщение или i для смены сервера

```

Рисунок 19 – Отправка UDP сообщения без шифрования

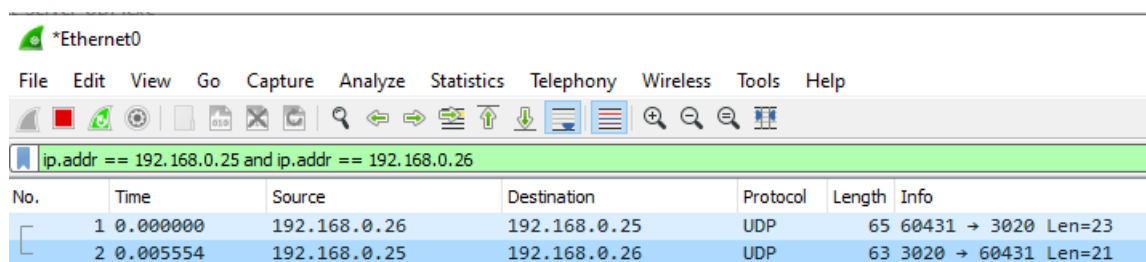
```

C:\Users\TixAn\Desktop\Client-Server_UDP.exe
-----Выберите режим работы программы-----
Режим клиента
> Режим сервера <
Выход
-----Программа UDP Клиент-Сервер-----
Инициализация сокета прошла успешно
Сокет успешно привязан
=====
Нажмите ESC что бы завершить работу сервера
=====
Получено сообщение от клиента 192.168.0.26
Сообщение получено
Содержание сообщения: Test not coded message
Ответ отправлен

```

Рисунок 20 – Получение UDP сообщения без шифрования

Как можно заметить, сообщение и ответ были переданы успешно. Проверим как данные UDP пакеты выглядят в Wireshark (рисунок 23).

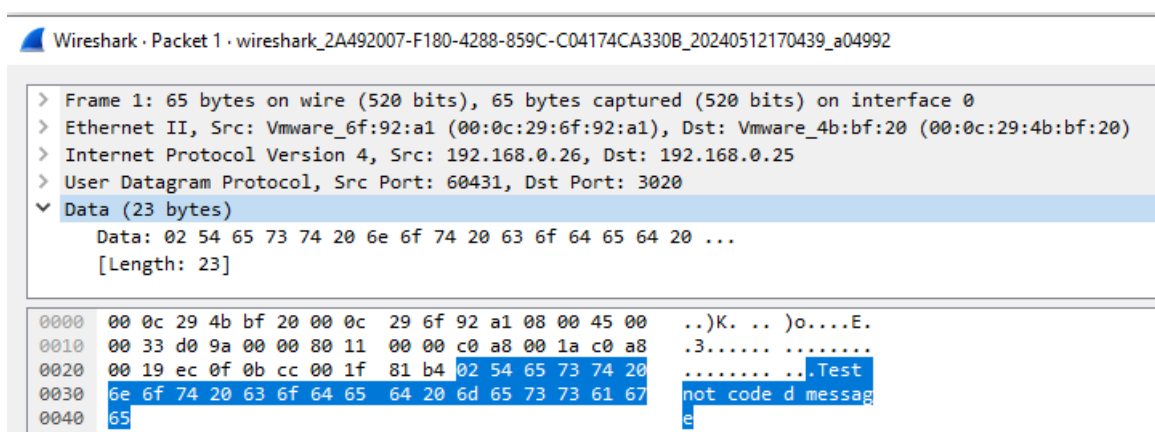


The image shows the Wireshark interface with a packet capture on the *Ethernet0 interface. The filter is set to 'ip.addr == 192.168.0.25 and ip.addr == 192.168.0.26'. Two packets are listed:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.26	192.168.0.25	UDP	65	60431 → 3020 Len=23
2	0.005554	192.168.0.25	192.168.0.26	UDP	63	3020 → 60431 Len=21

Рисунок 21 – Демонстрация пакетов в Wireshark

Далее будет продемонстрировано как выглядят полученные и отправленные пакеты в программе Wireshark. На рисунках 24 и 25 можно увидеть разбор самих пакетов на сетевые заголовки и находящиеся в них данные, а именно на что стоит обратить внимание это поле Data.



The image shows the packet details pane for Packet 1. The packet is a Frame 1 (65 bytes on wire, 65 bytes captured) on interface 0. It is an Ethernet II packet with source VMware_6f:92:a1 and destination VMware_4b:bf:20. It is an Internet Protocol Version 4 packet with source 192.168.0.26 and destination 192.168.0.25. It is a User Datagram Protocol packet with source port 60431 and destination port 3020. The Data field (23 bytes) is expanded, showing the raw data in hexadecimal and ASCII.

Offset	Hex	ASCII
0000	00 0c 29 4b bf 20 00 0c 29 f6 92 a1 08 00 45 00	..)K. ..)o....E.
0010	00 33 d0 9a 00 00 80 11 00 00 c0 a8 00 1a c0 a8	.3..... ..
0020	00 19 ec 0f 0b cc 00 1f 81 b4 02 54 65 73 74 20	Test
0030	6e 6f 74 20 63 6f 64 65 64 20 6d 65 73 73 61 67	not code d messag
0040	65	e

Рисунок 22 – Разбор отправляемого пакета в Wireshark

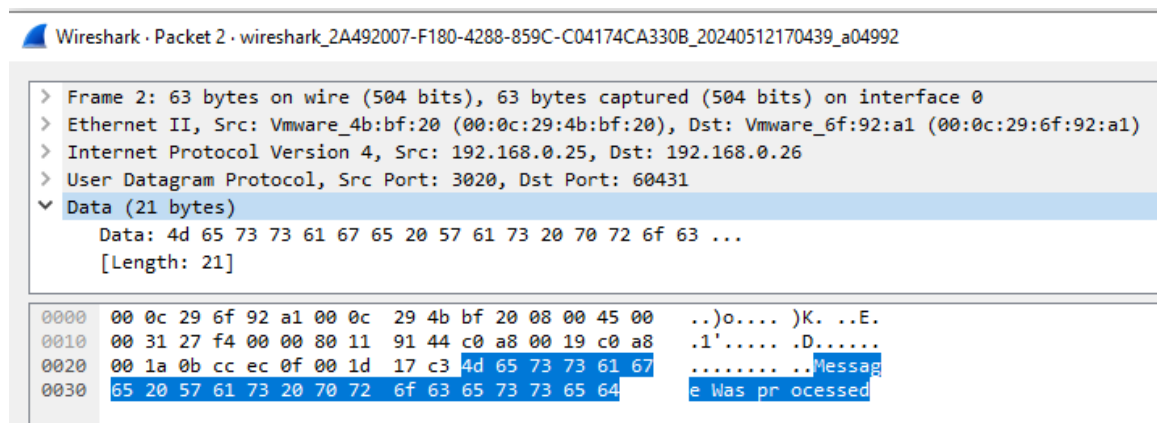


Рисунок 23 – Разбор полученного ответа в Wireshark

В UDP пакетах в этом поле содержится полезная нагрузка с данными, которая будет подвержена шифрованию драйвером, если защита сети включена. Так же в нижней части рисунков отображено как выглядит сетевой пакет в памяти компьютера в шестнадцатеричном виде, а правее ANSI кодировка этой памяти, что позволяет увидеть текст отправляемых UDP сообщений. Представление полезной нагрузки в памяти и её ANSI расшифровка выделены синим цветом. Как можно заметить сообщения в Wireshark и программе идентичны, что помогает удостовериться в том, что программа для обмена UDP пакетами никак не изменяет сетевой трафик и обрабатывает его правильно.

Сообщение в Wireshark на компьютере-получателе выглядят идентичным образом поэтому не отображены на рисунках. Разбор пакетов на обоих компьютерах будут продемонстрированы в двух следующих пунктах тестирования.

5.3.2 Тестирование с шифрованием

Проведём тестовую передачу одного UDP сообщения таким же способом. Для начала необходимо включить шифрование и настроить защищённое соединение. Настройки на виртуальной машине, с Ip-адресом 192.168.0.25 и 192.168.0.26, и реестр выглядят следующим образом, как показано на рисунках 26 и 27.

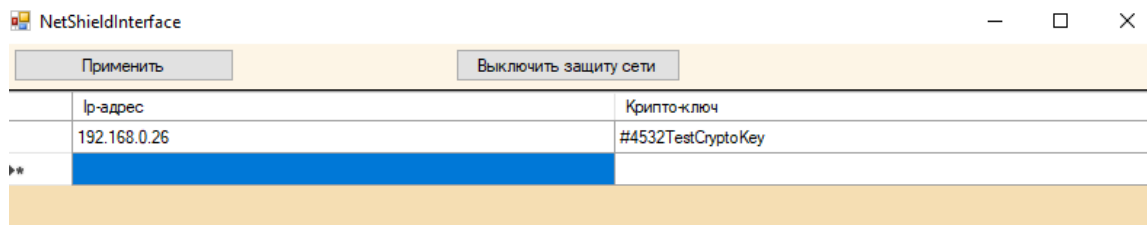


Рисунок 24 – Настройки на второй виртуальной машине

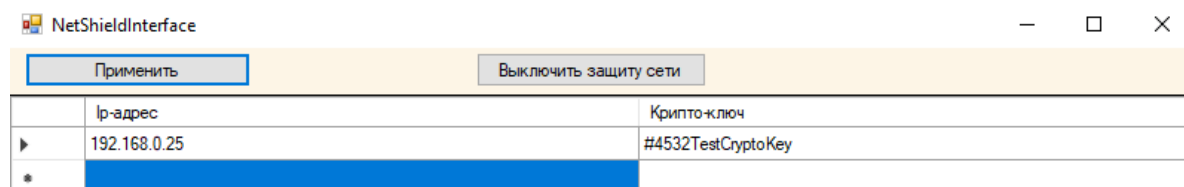


Рисунок 25 – Настройки на первой виртуальной машине

Проверим как выглядит реестр на второй виртуальной машине, что-бы удостовериться что настройки действительно применились (рисунок 28).

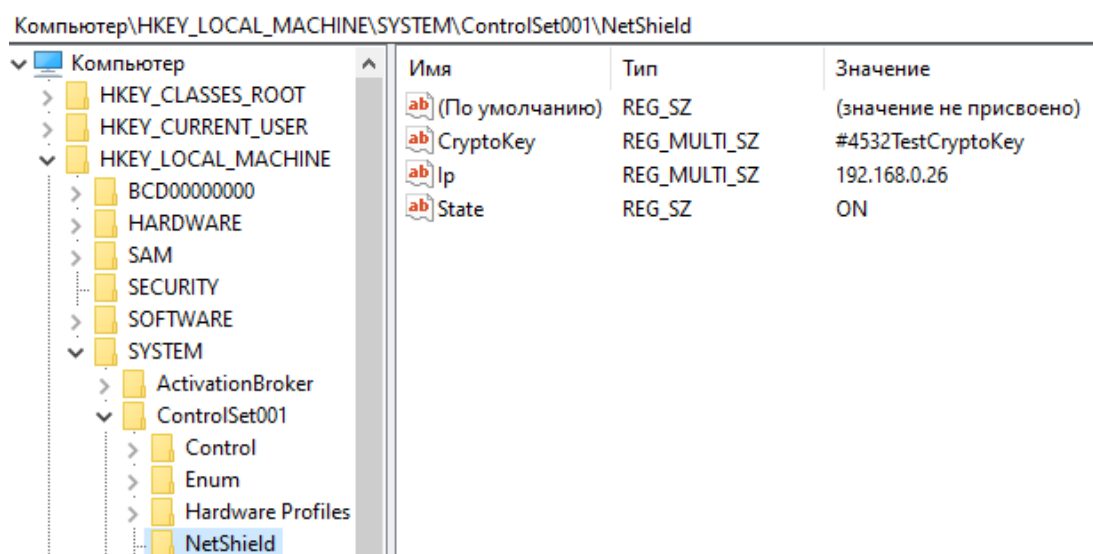


Рисунок 26 – Раздел реестра с настройками

Так как Ip-адреса на обоих компьютерах выставлены верно и криптографические ключи совпадают шифрование будет работать правильно. Отправим сообщение «Test coded message» и проверим результат в программе на обоих виртуальных машинах, мы должны получить такой же результат, как и с выключенным шифрованием. На рисунках 29 и 30 представлен результат отправки и получения сообщения.

```

C:\Users\TixAn\Desktop\Client-Server_UDP.exe
-----Выберите режим работы программы-----
> Режим клиента <
Режим сервера
Выход
-----Программа UDP Клиент-Сервер-----
Инициализация сокета прошла успешно
Введите IP адрес сервера (по умолчанию 127.0.0.1):192.168.0.25
Текущий адрес сервера для подключения 192.168.0.25
Подключение к серверу прошло успешно
=====
Нажмите ESC что бы завершить работу клиента
=====
Нажмите N что бы отправить сообщение или i для смены сервера
Введите текст сообщения:
Test coded message
Сообщение отправлено
Сообщение получено
Полученное сообщение: Message Was processed
Нажмите N что бы отправить сообщение или i для смены сервера

```

Рисунок 27 – Отправка UDP сообщения с шифрованием

```

Выбрать C:\Users\TixAn\Desktop\Client-Server_UDP.exe
-----Выберите режим работы программы-----
Режим клиента
> Режим сервера <
Выход
-----Программа UDP Клиент-Сервер-----
Инициализация сокета прошла успешно
Сокет успешно привязан
=====
Нажмите ESC что бы завершить работу сервера
=====
Получено сообщение от клиента 192.168.0.26
Сообщение получено
Содержание сообщения: Test coded message
Ответ отправлен

```

Рисунок 28 – Получение UDP сообщения с шифрованием

Как можно заметить из рисунков 29 и 30, сообщение и ответ были получены и обработаны правильно. Далее проверим как содержимое данных сетевых пакетов выглядит в Wireshark на обоих виртуальных машинах и хосте (рисунок 31,32), который в данной ситуации является промежуточной точкой передачи пакетов.

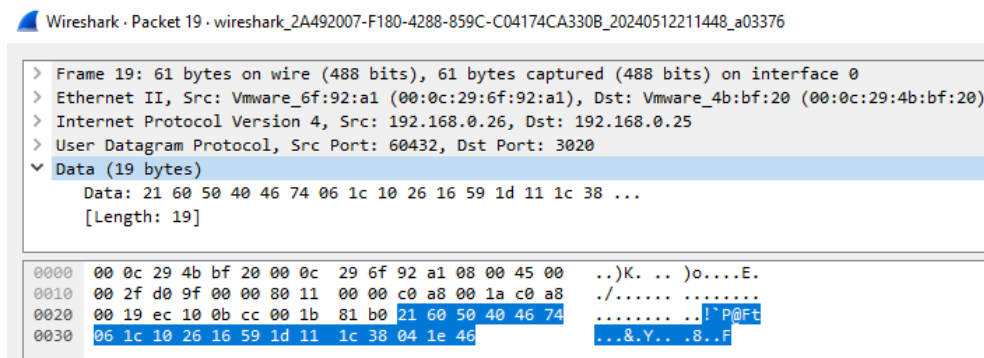


Рисунок 29 – Отправленный сетевой пакет с шифрованием на первой виртуальной машине

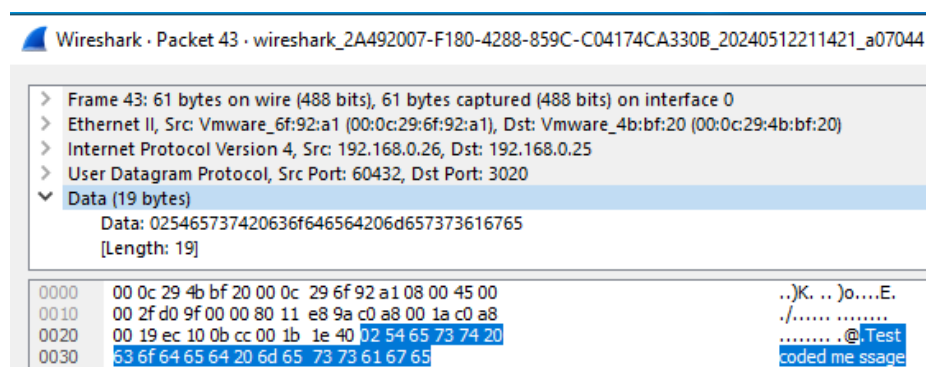


Рисунок 30 – Полученный и расшифрованный сетевой пакет на второй виртуальной машине

Как можно убедиться из рисунков, представленными выше полезная нагрузка в передаваемом UDP пакете была зашифрована на первой виртуальной машине и верно расшифрована на второй. Так как разработанный драйвер в стеке драйверов сетевого адаптера находится перед sniffером Wireshark и успевает проделать операцию шифрования или расшифровки над проходящим и уходящим сетевым трафиком, можно увидеть, как пакет шифруется и расшифровывается в начальной и конечной точке передачи сетевого пакета. Таким же образом шифруется и расшифровывается автоматически отправляемый ответ, что подтверждают рисунки 33 и 34.

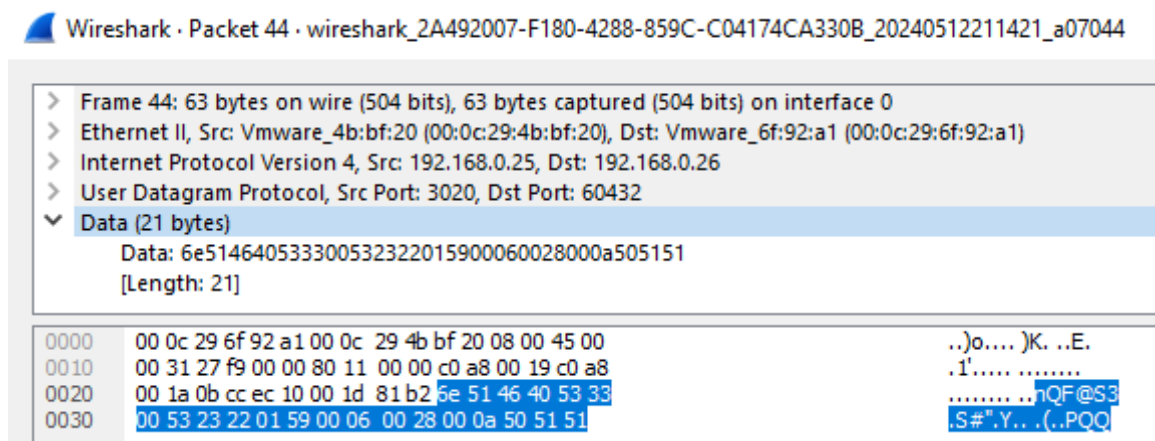


Рисунок 31 – Зашифрованный автоматический ответ на второй виртуальной

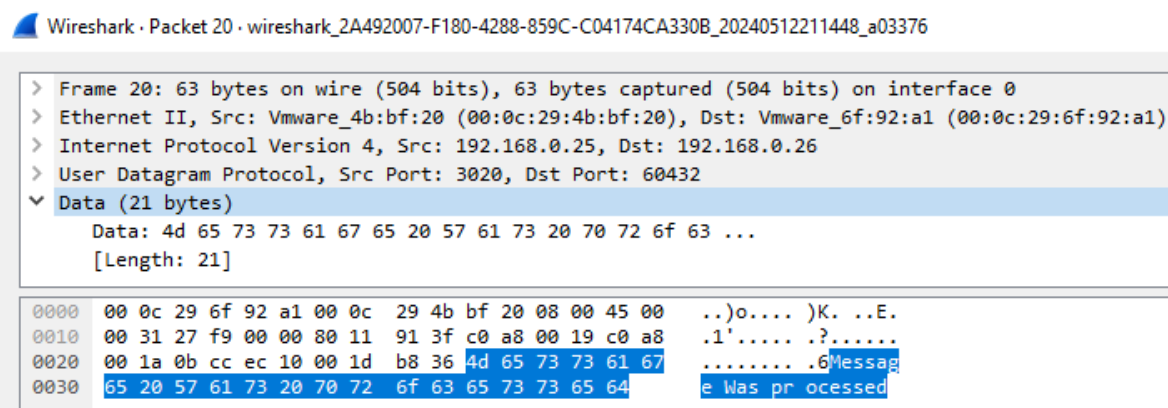


Рисунок 32 – Расшифрованный автоматический ответ на первой виртуальной машине

Что бы убедиться, что данные действительно передаются в зашифрованном виде проверим как данные сетевые пакеты с сообщением и автоматическим ответом выглядят в промежуточной точке, а именно на хосте виртуальных машин. Разбор данных пакетов представлен на рисунках 35 и 36.

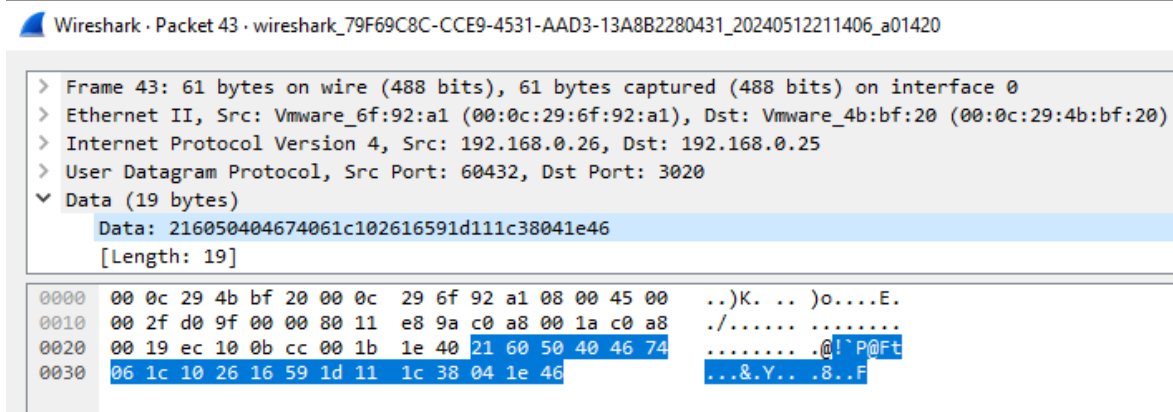


Рисунок 33 – Зашифрованное отправляемое сообщение на хосте

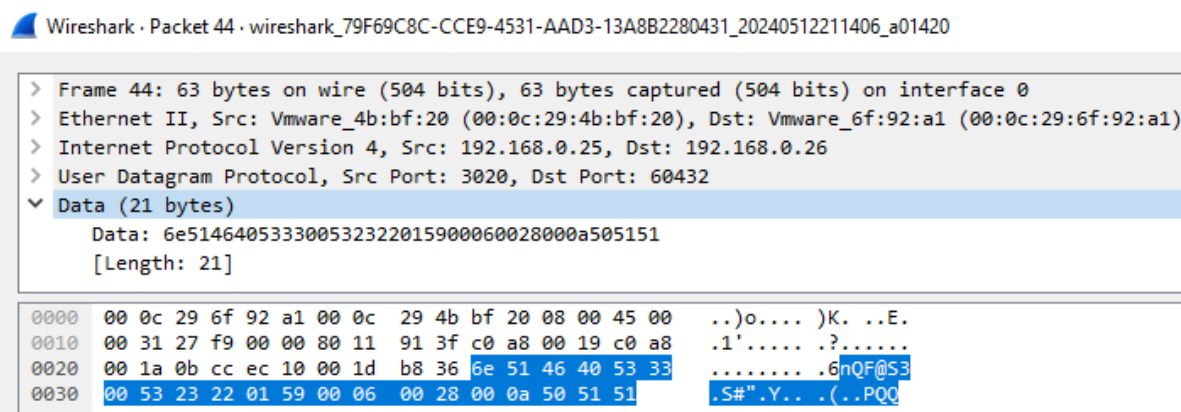


Рисунок 34 – Зашифрованный автоматический ответ на хосте

Как можно убедиться из рисунков 35 и 36, все передаваемые сетевые пакеты были в зашифрованном виде в промежуточной точке передачи данных. Убедиться в том, что это одни и те же пакеты можно сравнив рисунки 35 и 31, 36 и 33 соответственно.

5.3.3 Тестирование с неправильно настроенным шифрованием

Изменим ключ шифрование через оснастку драйвера и проверим как работает шифрование при помощи отправки того же сообщения «Test coded message» и автоматического ответа. После изменения, настройки шифрования на второй виртуальной машине выглядят следующим образом, как показано на рисунке 37.

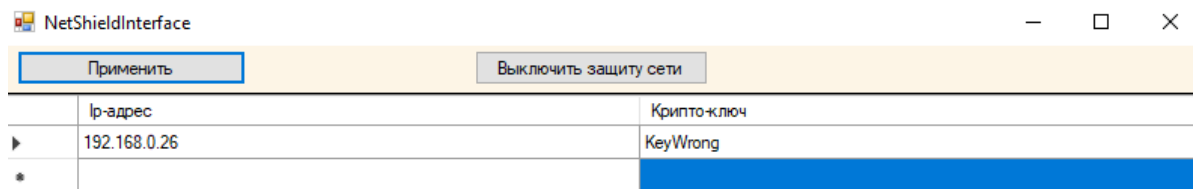


Рисунок 35 – Изменённые настройки драйвера на второй виртуальной машине

После изменения настроек на виртуальных машинах находятся различные ключи шифрования, что должно привести к неправильно расшифровке сетевых пакетов. На рисунках 38 и 39 продемонстрирован результат работы тестовой программы.

```

C:\Users\TixAn\Desktop\Client-Server_UDP.exe
-----Выберите режим работы программы-----
> Режим клиента <
Режим сервера
Выход
-----Программа UDP Клиент-Сервер-----
Инициализация сокета прошла успешно
Введите IP адрес сервера (по умолчанию 127.0.0.1):192.168.0.25
Текущий адрес сервера для подключения 192.168.0.25
Подключение к серверу прошло успешно
=====
Нажмите ESC что бы завершить работу клиента
=====
Нажмите N что бы отправить сообщение или i для смены сервера
Введите текст сообщения:
Test coded message
Сообщение отправлено
Сообщение получено
Полученное сообщение: %4?±!\n4hGx?rin0Ko)♣#
Нажмите N что бы отправить сообщение или i для смены сервера
  
```

Рисунок 36 – Отправка UDP сообщения с неправильным шифрованием

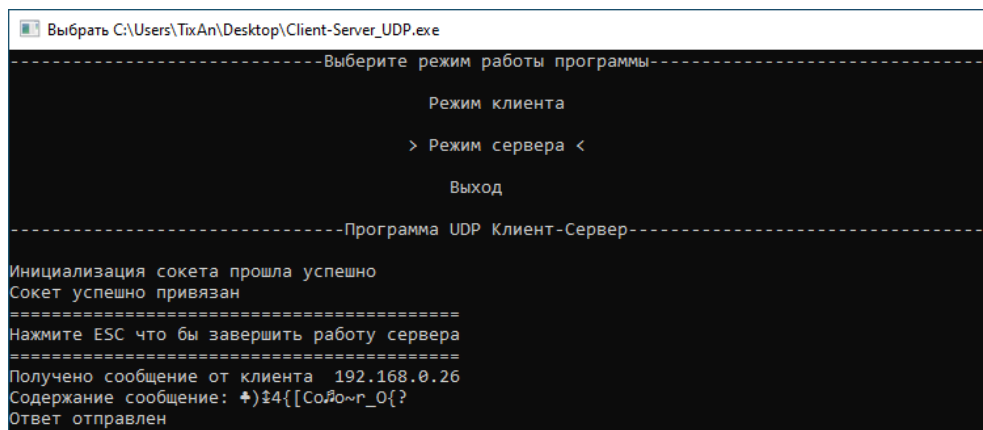


Рисунок 37 – Получение UDP сообщения с неправильным шифрованием

Как можно увидеть из рисунков, представленных выше, сообщения на обеих машинах были расшифрованы неверно. Проверим как передаваемые сетевые пакеты выглядят в программе Wireshark (рисунок 40, 41).

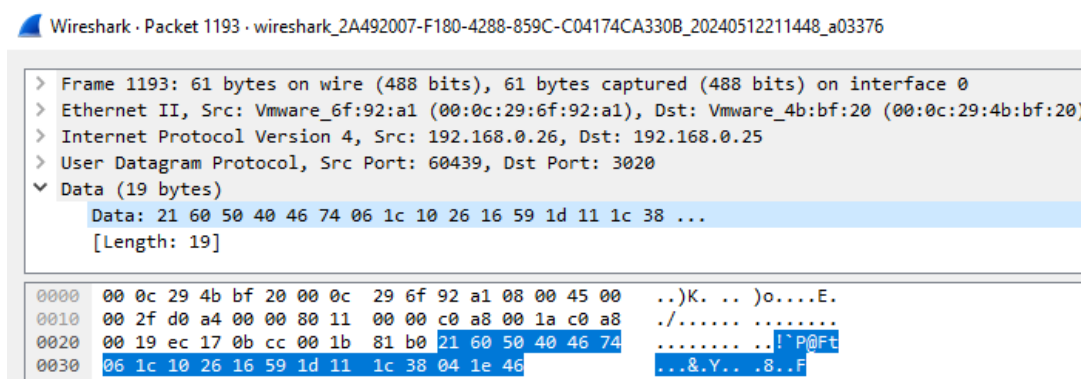


Рисунок 38 – Зашифрованный пакет, отправленный первой виртуальной машиной

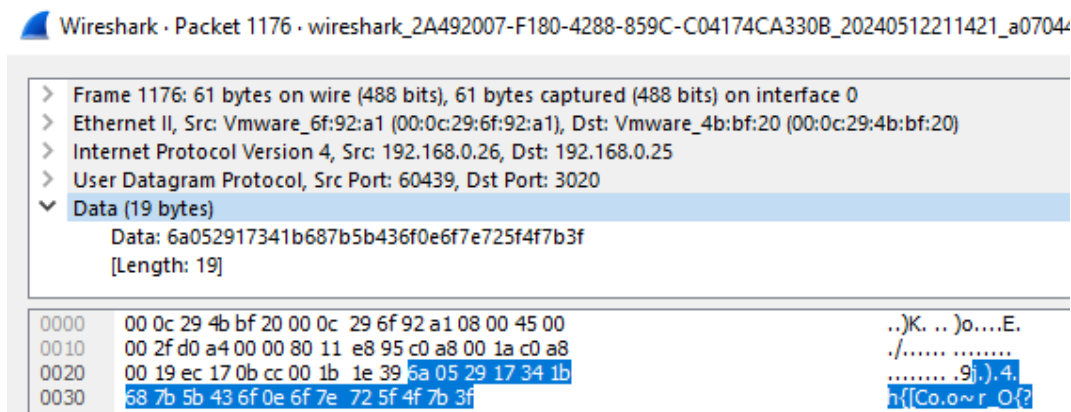


Рисунок 39 – Расшифрованный сетевой пакет на второй виртуальной машине

Как можно понять из сравнения рисунков 32 и 41, полученные сетевые пакеты, с одной и той же полезной нагрузкой были расшифрованы по-разному. Так же стоит отметить, что шифрование полезной нагрузки на первой виртуальной машине никак не изменилось, так как ключи шифрования не менялись на данной виртуальной машине, что подтверждают рисунки 31 и 40. Далее рассмотрим, как выглядит шифрование и расшифровка полезной нагрузки сетевых пакетов с автоматическим ответом. Рисунки с разбором данных сетевых пакетов представлены ниже на рисунках 42 и 43.

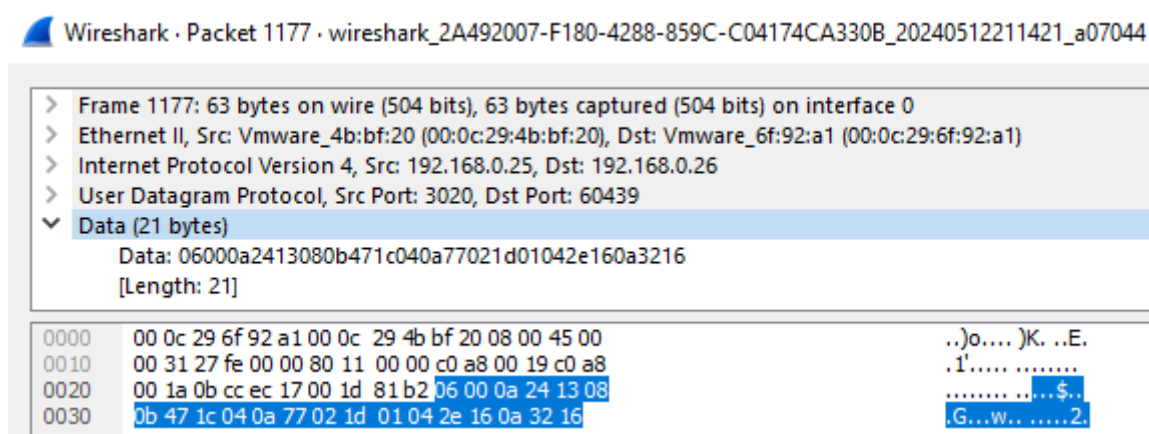


Рисунок 40 – Зашифрованный новым ключом атематический ответ

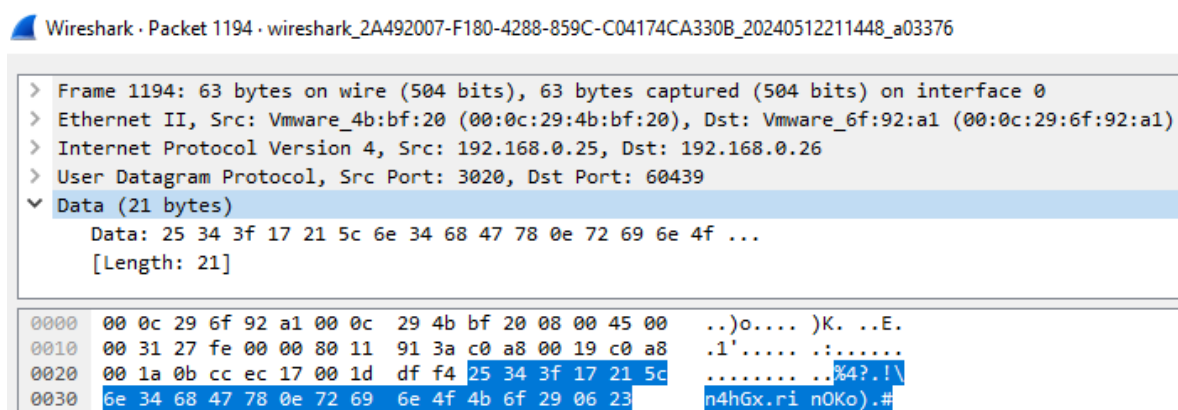
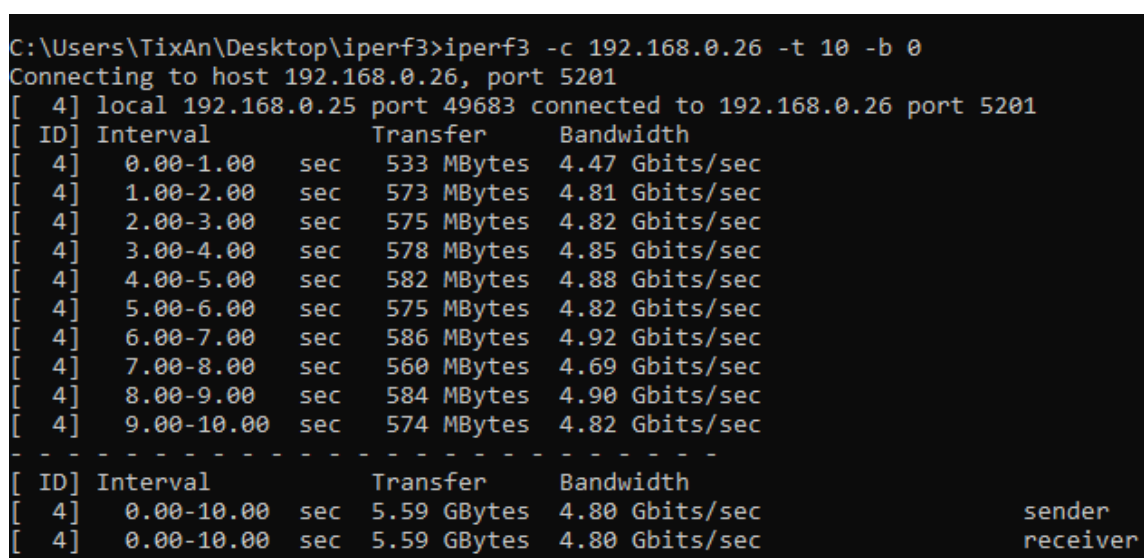


Рисунок 41 – Расшифрование автоматического ответа на первой виртуальной машине

Ввиду различий ключей шифрования на второй виртуальной машине, одинаковый автоматический ответ был зашифрован по-разному, что подтверждают рисунки 33 и 42.

5.3.4 Тестирование скорости передачи данных

В данном пункте представлено тестирование пропускной способности сетевого адаптера до и после включения функции шифрования трафика. Проверка пропускной способности будет проводиться при помощи программы Iperf3 путём отправки TCP трафика в течении 10 секунд. Настройки отправки потока сетевого трафика выставлены так, чтобы они никак не ограничивали пропускную способность сетевого адаптера и использовали максимум возможных ресурсов. На рисунке 44 представлен результат отправки сетевого трафика в течении 20 секунд с выключенным шифрованием.



```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49683 connected to 192.168.0.26 port 5201
[ ID] Interval            Transfer          Bandwidth
[ 4] 0.00-1.00 sec       533 MBytes      4.47 Gbits/sec
[ 4] 1.00-2.00 sec       573 MBytes      4.81 Gbits/sec
[ 4] 2.00-3.00 sec       575 MBytes      4.82 Gbits/sec
[ 4] 3.00-4.00 sec       578 MBytes      4.85 Gbits/sec
[ 4] 4.00-5.00 sec       582 MBytes      4.88 Gbits/sec
[ 4] 5.00-6.00 sec       575 MBytes      4.82 Gbits/sec
[ 4] 6.00-7.00 sec       586 MBytes      4.92 Gbits/sec
[ 4] 7.00-8.00 sec       560 MBytes      4.69 Gbits/sec
[ 4] 8.00-9.00 sec       584 MBytes      4.90 Gbits/sec
[ 4] 9.00-10.00 sec      574 MBytes      4.82 Gbits/sec
-----
[ ID] Interval            Transfer          Bandwidth
[ 4] 0.00-10.00 sec      5.59 GBytes      4.80 Gbits/sec
[ 4] 0.00-10.00 sec      5.59 GBytes      4.80 Gbits/sec
sender receiver
```

Рисунок 42 – Результат тестирования скорости с выключенным шифрованием

Как можно заметить средняя скорость отправки и получения сетевого трафика одинакова и составляет 4.8 Гигабит в секунду. Далее включим шифрование трафика на обоих виртуальных машинах и проверим пропускную способность сетевых адаптеров. Результат тестирования продемонстрирован на рисунке 45.

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49687 connected to 192.168.0.26 port 5201
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-1.00   sec      276 MBytes  2.30 Gbits/sec
[ 4] 1.00-2.00   sec      281 MBytes  2.36 Gbits/sec
[ 4] 2.00-3.00   sec      286 MBytes  2.40 Gbits/sec
[ 4] 3.00-4.00   sec      279 MBytes  2.34 Gbits/sec
[ 4] 4.00-5.00   sec      287 MBytes  2.41 Gbits/sec
[ 4] 5.00-6.00   sec      283 MBytes  2.38 Gbits/sec
[ 4] 6.00-7.00   sec      284 MBytes  2.38 Gbits/sec
[ 4] 7.00-8.00   sec      285 MBytes  2.39 Gbits/sec
[ 4] 8.00-9.00   sec      286 MBytes  2.40 Gbits/sec
[ 4] 9.00-10.00  sec      277 MBytes  2.32 Gbits/sec
-----
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-10.00  sec      2.76 GBytes  2.37 Gbits/sec
[ 4] 0.00-10.00  sec      2.76 GBytes  2.37 Gbits/sec
iperf Done.
```

Рисунок 43 – Результат тестирования пропускной способности со включенным шифрованием

Как видно из проведённого теста, пропускная способность сетевого адаптера уменьшилась при включении шифрования почти в два раза, однако, это не означает что включение шифрования урезает скорость передачи данных в два раза, такое падение заметно ввиду изначально высокой скорости передачи данных. Программно уменьшим максимальную пропускную способность путём изменения параметра *b* при запуске теста. Далее на рисунках 46 и 47 представлены результаты тестирования с программно ограниченной пропускной способностью.

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49707 connected to 192.168.0.26 port 5201
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-1.00   sec      22.2 MBytes  187 Mb/s
[ 4] 1.00-2.00   sec      24.1 MBytes  202 Mb/s
[ 4] 2.00-3.02   sec      23.1 MBytes  191 Mb/s
[ 4] 3.02-4.00   sec      23.9 MBytes  200 Mb/s
[ 4] 4.02-5.00   sec      24.5 MBytes  209 Mb/s
[ 4] 5.00-6.02   sec      23.2 MBytes  192 Mb/s
[ 4] 6.02-7.00   sec      24.2 MBytes  207 Mb/s
[ 4] 7.00-8.02   sec      23.4 MBytes  193 Mb/s
[ 4] 8.02-9.02   sec      24.9 MBytes  209 Mb/s
[ 4] 9.02-10.00  sec      23.4 MBytes  199 Mb/s
-----
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-10.00  sec      237 MBytes  199 Mb/s
[ 4] 0.00-10.00  sec      237 MBytes  199 Mb/s
iperf Done.
```

Рисунок 44 – Результат тестирования скорости с выключенным шифрованием с ограничением пропускной способности

```

C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49739 connected to 192.168.0.26 port 5201
[ ID] Interval           Transfer     Bandwidth
[ 4]  0.00-1.00      sec    22.0 MBytes    184 Mbits/sec
[ 4]  1.00-2.02      sec    23.6 MBytes    195 Mbits/sec
[ 4]  2.02-3.02      sec    24.5 MBytes    206 Mbits/sec
[ 4]  3.02-4.00      sec    23.4 MBytes    199 Mbits/sec
[ 4]  4.00-5.00      sec    23.9 MBytes    200 Mbits/sec
[ 4]  5.00-6.02      sec    23.6 MBytes    195 Mbits/sec
[ 4]  6.02-7.00      sec    24.5 MBytes    209 Mbits/sec
[ 4]  7.00-8.00      sec    23.6 MBytes    198 Mbits/sec
[ 4]  8.00-9.00      sec    23.6 MBytes    198 Mbits/sec
[ 4]  9.02-10.00     sec    23.4 MBytes    199 Mbits/sec
-----
[ ID] Interval           Transfer     Bandwidth
[ 4]  0.00-10.00     sec    237 MBytes    199 Mbits/sec
[ 4]  0.00-10.00     sec    237 MBytes    199 Mbits/sec
sender
receiver
iperf Done.

```

Рисунок 45 – Результат тестирования скорости со включенным шифрованием с ограничением пропускной способности

Как можно заметить, в данной ситуации падение скорости передачи данных не было обнаружено, при всё ещё большом объёме передаваемых данных в секунду, что позволяет сделать вывод, что замедление скорости передачи данных при включении шифрования будет более заметно при увеличении пропускной способности сетевого адаптера.

6 Безопасность жизнедеятельности

С развитием научно-технического прогресса немаловажную роль играет возможность безопасного исполнения людьми своих трудовых обязанностей, в том числе это относится и к людям, рабочая деятельность которых, тесно связана с компьютером. Безопасность жизнедеятельности при работе за компьютером — это комплекс мер, направленных на обеспечение безопасности и сохранение здоровья пользователя при работе с компьютерной техникой. Работа за компьютером может быть связана с определёнными рисками для здоровья и безопасности, поэтому важно соблюдать требования и меры предосторожности.

Рабочее место должно быть хорошо освещено и иметь достаточную площадь для размещения компьютерного оборудования и документов. Это необходимо для обеспечения удобства работы и предотвращения травм, связанных с падением оборудования или спотыканием об провода.

Монитор должен располагаться на оптимальном расстоянии от глаз пользователя. Клавиатура и мышь должны располагаться на уровне локтей или немного ниже. Стул должен быть удобным и регулируемым по высоте. Эти меры необходимы для предотвращения напряжения глаз, шеи, рук и запястий, а также для обеспечения правильной осанки и предотвращения болей в спине.

Необходимо делать перерывы в работе каждые 30–45 минут, чтобы размяться и дать глазам отдохнуть. Это необходимо для предотвращения переутомления и снижения концентрации внимания.

Помимо основных требований безопасности, существуют дополнительные меры, которые могут помочь предотвратить травмы и заболевания при работе за компьютером:

- использование защитного экрана;
- правильное положение монитора;
- использование подставки для ног;
- правильное расположение клавиатуры и мыши.

Соблюдение этих требований поможет обеспечить безопасность и комфорт при работе за компьютером.

					ПГУ 1.090301.37.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпи	Дата		

Заключение

В ходе выполнения данной выпускной квалификационной работы на тему «Программа реализации защищенного соединения» были получены и закреплены навыки разработки сетевых драйверов для операционной системы Windows, взаимодействия с ядром операционной системы и сетевым стеком NDIS. В результате выполнения работы был разработан драйвер сетевого фильтра NDIS, оснастка для его настройки, а также отчёт, в котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор его основных функциональных особенностей.

Разработанный драйвер позволяет создавать защищённые соединения путём шифрования методом гаммирования передаваемых сетевых пакетов, использующих транспортные протоколы UDP или TCP, между двумя компьютерами, а также оснастка для настройки параметров шифрования в данном драйвере. Проведённое тестирование показало, что разработанный драйвер соответствует всем поставленным в начале работы требованиям и работает корректно, не вызывая критических ошибок в системе, что является частой проблемой драйверов операционной системы Windows.

В дальнейшем данный драйвер можно усовершенствовать, добавив поддержку шифрования всех транспортных протоколов, поддержку протокола IPv6, а также увеличить безопасность шифрования, путём усложнения алгоритма генерации и работы с ключами и ключевой информацией для шифрования. Разработанную оснастку в свою очередь можно улучшить путём добавления возможности выключения защиты определённого соединения, добавления поддержки IPv6 адресов, а также общее улучшения качества графического интерфейса.

					ПГУ 1.090301.37.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпи	Дата		

Список использованных источников

1. Утечки данных в 2023 году. [Электронный ресурс]/Режим доступа: https://www.kaspersky.ru/about/press-releases/2023_laboratoriya-kasperskogo-obuyom-utekshih-dannyh-vygos-v-2023-godu-na-33, свободный. (дата обращения 05.04.2024).
2. Эндрю Таненбаум, Дэвид Уэзеролл. Компьютерные сети — Москва: Питер, 2023. — 955 с.
3. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. — СПб.: Питер, 2016. — 992 с.: ил. — (Серия «Учебник для вузов»).
4. Русинович Марк, Соломон Дэвид Внутреннее устройство Microsoft Windows. Основные подсистемы ОС. 6-е изд. — М.: ДМК Пресс, 2018. — 944 с.: ил. — ISBN 978-5-97060-628-1.
5. Стратегия разработки драйверов NDIS. [Электронный ресурс]/Режим доступа: <https://learn.microsoft.com/ru-ru/windows-ardware/drivers/network/roadmap-for-developing-ndis-drivers>, свободный. (дата обращения 12.02.2024)
6. Создание надежных драйверов Kernel-Mode. [Электронный ресурс]/Режим доступа: <https://learn.microsoft.com/ru-ru/windows-hardware/drivers/kernel/creating-reliable-kernel-mode-drivers>, свободный. (дата обращения 13.02.2024)
7. Гордеев А.В. Операционные системы: Учебник для вузов. — 2-е изд. — СПб.: Питер, 2009. — 416 с.: ил. — (Серия «Учебник для вузов»).
8. Спецификация NDIS драйверов. [Электронный ресурс]/Режим доступа: <https://habr.com/ru/articles/111592/>, свободный. (дата обращения 26.04.2024)

Приложение А

(Обязательное)

Программа реализации защищённого соединения

UML-диаграммы

					ПГУ 1.090301.37.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпи	Дата		

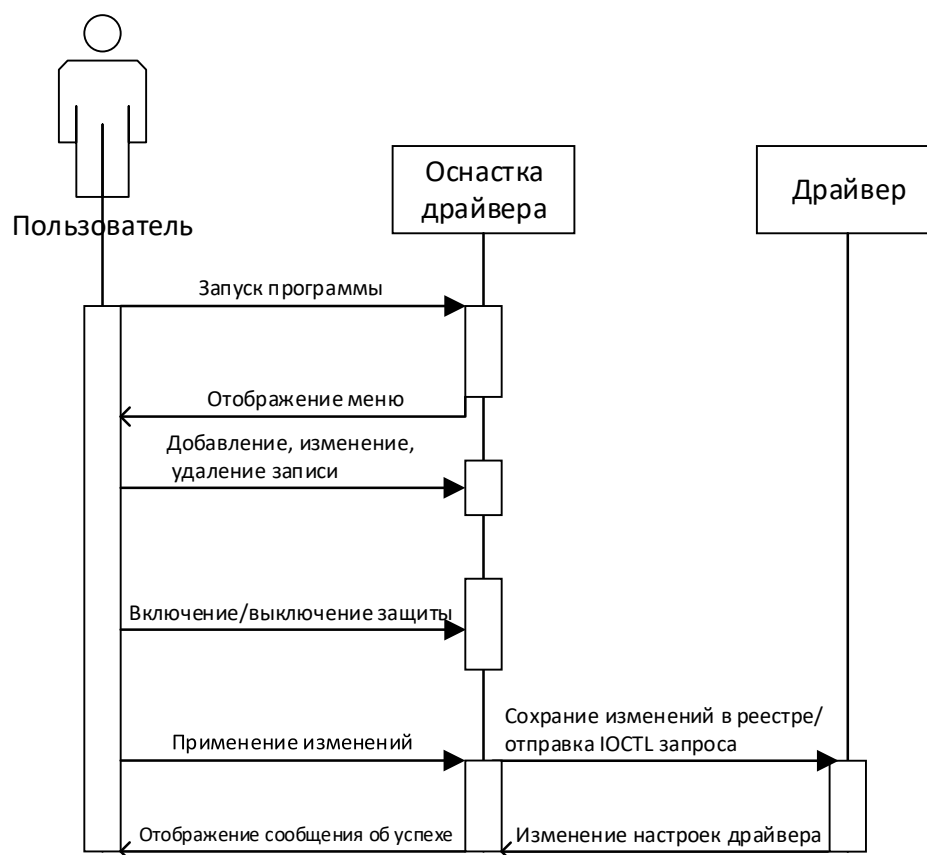
Приложение А.1 – Диаграмма развёртывания

[illegible]

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпи</i>	<i>Дата</i>

ПГУ 1.090301.37.001 ПЗ

Приложение А.2 – Диаграмма последовательностей



					ПГУ 1.090301.37. 003 Д11				
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированное функциональное испытательное задание на разработку и верификацию Драйвера последовательностей	Лит.	Лист	Листов	
Разработчик	Полосов А.А.								
Проверен	Гусин Е.И.								
И. контр.	Басков А.С.								ФЭТ, каф. ВТ пр. 2025/22
Утвержден	Митрошкин М.А.								

Изм.	Лист	№ докум.	Подпи	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

75

Приложение А.3 – Диаграмма деятельности 1



						ПГУ 1.090301.37. 003 Д11		
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированное функционирование информационного обмена индивидуальными и групповыми пользователями (Диагностика последовательностей)	Лит.	Лист	Листов
Разработ		Тихонов Д.А.						
Провер.		Гурин Е.И.						
И. контр.		Басков А.С.			ФЭТ, каф. ВТ пр. 2025/26			
Утвержден		Митрошкин М.А.						

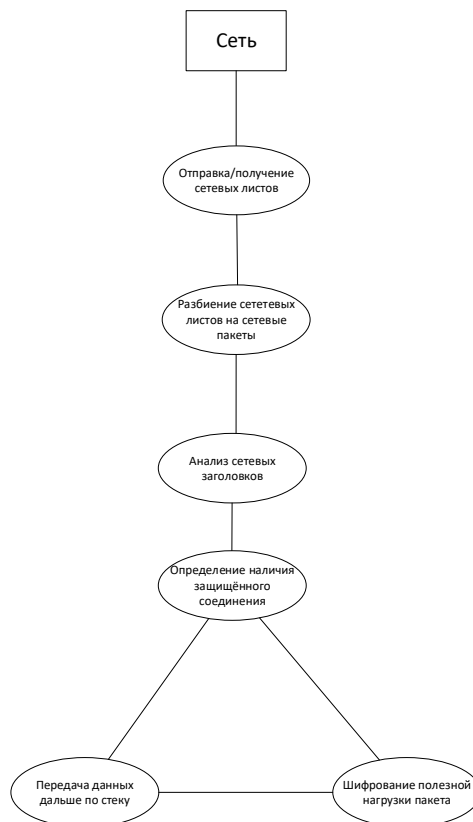
Изм.	Лист	№ докум.	Подпи	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

76

Приложение А.4 – Диаграмма деятельности 2



ПГУ 1.090301.37.003 Д11									
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированное функционирование информационного обмена				
Разработчик	Полосов А.А.				индивидуальный и групповой контроль				
Проверка	Гусев Е.И.				Достоинства и недостатки				
И. контр.	Басков А.С.				ФЭТ, каф. ВТ				
Исполн.	Митрошкин М.А.				пр. 2025/22				

Изм.	Лист	№ докум.	Подпи	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

77

Приложение Б

(Обязательное)

Программа реализации защищённого соединения

Листинг программы

					ПГУ 1.090301.37.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпи	Дата		

Файл filter.c

```
#include "precomp.h"
#define __FILENUMBER 'PNPF'

#pragma NDIS_INIT_FUNCTION(DriverEntry)

NDIS_HANDLE FilterDriverHandle;
NDIS_HANDLE FilterDriverObject;
NDIS_HANDLE NdisFilterDeviceHandle = NULL;
PDEVICE_OBJECT NdisDeviceObject = NULL;

FILTER_LOCK FilterListLock;
LIST_ENTRY FilterModuleList;

NetHelper* HelperObject;
HANDLE InterfaceThread;

NDIS_FILTER_PARTIAL_CHARACTERISTICS DefaultChars = {
    FilterSendNetBufferLists,
    FilterSendNetBufferListsComplete,
    NULL,
    FilterReceiveNetBufferLists,
    FilterReturnNetBufferLists
};

_Use_decl_annotations_
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING
RegistryPath) {
    NDIS_STATUS Status;
    NDIS_FILTER_DRIVER_CHARACTERISTICS FChars;
    NDIS_STRING ServiceName =
RTL_CONSTANT_STRING(FILTER_SERVICE_NAME);
    NDIS_STRING UniqueName =
RTL_CONSTANT_STRING(FILTER_UNIQUE_NAME);
    NDIS_STRING FriendlyName =
RTL_CONSTANT_STRING(FILTER_FRIENDLY_NAME);
    BOOLEAN bFalse = FALSE;

    UNREFERENCED_PARAMETER(RegistryPath);

    DEBUGP(DL_TRACE, "===>DriverEntry...\n");

    FilterDriverObject = DriverObject;
    do {
        NdisZeroMemory(&FChars,
sizeof(NDIS_FILTER_DRIVER_CHARACTERISTICS));
        FChars.Header.Type =
NDIS_OBJECT_TYPE_FILTER_DRIVER_CHARACTERISTICS;
        FChars.Header.Size =
sizeof(NDIS_FILTER_DRIVER_CHARACTERISTICS);
        #if NDIS_SUPPORT_NDIS61
            FChars.Header.Revision =
NDIS_FILTER_CHARACTERISTICS_REVISION_2;
        #else
            FChars.Header.Revision =
NDIS_FILTER_CHARACTERISTICS_REVISION_1;
        #endif
    } while (0);
}
```

					ПГУ 1.090301.37.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпи	Дата		

```

FChars.MajorNdisVersion = FILTER_MAJOR_NDIS_VERSION;
FChars.MinorNdisVersion = FILTER_MINOR_NDIS_VERSION;
FChars.MajorDriverVersion = 1;
FChars.MinorDriverVersion = 0;
FChars.Flags = 0;

FChars.FriendlyName = FriendlyName;
FChars.UniqueName = UniqueName;
FChars.ServiceName = ServiceName;

FChars.SetOptionsHandler = FilterRegisterOptions;
FChars.AttachHandler = FilterAttach;
FChars.DetachHandler = FilterDetach;
FChars.RestartHandler = FilterRestart;
FChars.PauseHandler = FilterPause;
FChars.SetFilterModuleOptionsHandler = FilterSetModuleOptions;
FChars.OidRequestHandler = FilterOidRequest;
FChars.OidRequestCompleteHandler = FilterOidRequestComplete;
FChars.CancelOidRequestHandler = FilterCancelOidRequest;

FChars.SendNetBufferListsHandler = FilterSendNetBufferLists;
FChars.ReturnNetBufferListsHandler =
FilterReturnNetBufferLists;
FChars.SendNetBufferListsCompleteHandler =
FilterSendNetBufferListsComplete;
FChars.ReceiveNetBufferListsHandler =
FilterReceiveNetBufferLists;
FChars.DevicePnPEventNotifyHandler =
FilterDevicePnPEventNotify;
FChars.NetPnPEventHandler = FilterNetPnPEvent;
FChars.StatusHandler = FilterStatus;
FChars.CancelSendNetBufferListsHandler =
FilterCancelSendNetBufferLists;

DriverObject->DriverUnload = FilterUnload;
FilterDriverHandle = NULL;
FILTER_INIT_LOCK(&FilterListLock);

InitializeListHead(&FilterModuleList);

Status = NdisFRegisterFilterDriver(DriverObject,
(NDIS_HANDLE)FilterDriverObject,
                                &FChars,
                                &FilterDriverHandle);
if (Status != NDIS_STATUS_SUCCESS)
    break;

Status = NetShieldRegisterDevice();

if (Status != NDIS_STATUS_SUCCESS){
    NdisFDeregisterFilterDriver(FilterDriverHandle);
    FILTER_FREE_LOCK(&FilterListLock);
    break;
}

}while(bFalse);

```

```

GetIpKeyFromReg(L"\\REGISTRY\\Machine\\SYSTEM\\ControlSet001\\NetShield\\");
    return Status;
}

_Use_decl_annotations_
NDIS_STATUS FilterRegisterOptions(NDIS_HANDLE
NdisFilterDriverHandle,NDIS_HANDLE FilterDriverContext){
    ASSERT(NdisFilterDriverHandle == FilterDriverHandle);
    ASSERT(FilterDriverContext == (NDIS_HANDLE)FilterDriverObject);

    if ((NdisFilterDriverHandle != (NDIS_HANDLE)FilterDriverHandle) ||
(FilterDriverContext != (NDIS_HANDLE)FilterDriverObject))
        return NDIS_STATUS_INVALID_PARAMETER;

    return NDIS_STATUS_SUCCESS;
}

_Use_decl_annotations_
NDIS_STATUS FilterAttach(NDIS_HANDLE NdisFilterHandle,NDIS_HANDLE
FilterDriverContext,PNDIS_FILTER_ATTACH_PARAMETERS AttachParameters){
    PMS_FILTER                pFilter = NULL;
    NDIS_STATUS                Status = NDIS_STATUS_SUCCESS;
    NDIS_FILTER_ATTRIBUTES    FilterAttributes;
    ULONG                     Size;
    BOOLEAN                   bFalse = FALSE;

    do{
        ASSERT(FilterDriverContext ==
(NDIS_HANDLE)FilterDriverObject);
        if (FilterDriverContext != (NDIS_HANDLE)FilterDriverObject){
            Status = NDIS_STATUS_INVALID_PARAMETER;
            break;
        }

        if ((AttachParameters->MiniportMediaType != NdisMedium802_3)
&& (AttachParameters->MiniportMediaType != NdisMediumWan) &&
(AttachParameters->MiniportMediaType != NdisMediumWirelessWan)){
            Status = NDIS_STATUS_INVALID_PARAMETER;
            break;
        }

        Size = sizeof(MS_FILTER) + AttachParameters-
>FilterModuleGuidName->Length +AttachParameters-
>BaseMiniportInstanceName->Length + AttachParameters-
>BaseMiniportName->Length;

        pFilter = (PMS_FILTER)FILTER_ALLOC_MEM(NdisFilterHandle,
Size);
        if (pFilter == NULL){
            DEBUGP(DL_WARN, "Failed to allocate context
structure.\n");
            Status = NDIS_STATUS_RESOURCES;
            break;
        }

        NdisZeroMemory(pFilter, sizeof(MS_FILTER));
    }

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпи	Дата		

```

        pFilter->FilterModuleName.Length = pFilter-
>FilterModuleName.MaximumLength = AttachParameters-
>FilterModuleGuidName->Length;
        pFilter->FilterModuleName.Buffer = (PWSTR)((PUCHAR)pFilter +
sizeof(MS_FILTER));
        NdisMoveMemory(pFilter->FilterModuleName.Buffer,
AttachParameters->FilterModuleGuidName->Buffer, pFilter-
>FilterModuleName.Length);

        pFilter->MiniportFriendlyName.Length = pFilter-
>MiniportFriendlyName.MaximumLength = AttachParameters-
>BaseMiniportInstanceName->Length;
        pFilter->MiniportFriendlyName.Buffer =
(PWSTR)((PUCHAR)pFilter->FilterModuleName.Buffer + pFilter-
>FilterModuleName.Length);
        NdisMoveMemory(pFilter->MiniportFriendlyName.Buffer,
AttachParameters->BaseMiniportInstanceName->Buffer, pFilter-
>MiniportFriendlyName.Length);

        pFilter->MiniportName.Length = pFilter-
>MiniportName.MaximumLength = AttachParameters->BaseMiniportName-
>Length;
        pFilter->MiniportName.Buffer = (PWSTR)((PUCHAR)pFilter-
>MiniportFriendlyName.Buffer + pFilter->MiniportFriendlyName.Length);
        NdisMoveMemory(pFilter->MiniportName.Buffer, AttachParameters-
>BaseMiniportName->Buffer, pFilter->MiniportName.Length);

        pFilter->MiniportIfIndex = AttachParameters-
>BaseMiniportIfIndex;
        pFilter->TrackReceives = TRUE;
        pFilter->TrackSends = TRUE;
        pFilter->FilterHandle = NdisFilterHandle;

        NdisZeroMemory(&FilterAttributes,
sizeof(NDIS_FILTER_ATTRIBUTES));
        FilterAttributes.Header.Revision =
NDIS_FILTER_ATTRIBUTES_REVISION_1;
        FilterAttributes.Header.Size = sizeof(NDIS_FILTER_ATTRIBUTES);
        FilterAttributes.Header.Type =
NDIS_OBJECT_TYPE_FILTER_ATTRIBUTES;
        FilterAttributes.Flags = 0;

        NDIS_DECLARE_FILTER_MODULE_CONTEXT(MS_FILTER);
        Status = NdisFSetAttributes(NdisFilterHandle, pFilter,
&FilterAttributes);
        if (Status != NDIS_STATUS_SUCCESS)
            break;

        pFilter->State = FilterPaused;

        FILTER_ACQUIRE_LOCK(&FilterListLock, bFalse);
        InsertHeadList(&FilterModuleList, &pFilter->FilterModuleLink);
        FILTER_RELEASE_LOCK(&FilterListLock, bFalse);

    }while (bFalse);

    if (Status != NDIS_STATUS_SUCCESS){
        if (pFilter != NULL){

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпи	Дата		

```

        FILTER_FREE_MEM(pFilter);
    }
}
return Status;
}

_Use_decl_annotations_
NDIS_STATUS FilterPause(NDIS_HANDLE FilterModuleContext,
PNDIS_FILTER_PAUSE_PARAMETERS PauseParameters){
    PMS_FILTER          pFilter = (PMS_FILTER)(FilterModuleContext);
    NDIS_STATUS          Status;
    BOOLEAN              bFalse = FALSE;
    UNREFERENCED_PARAMETER(PauseParameters);

    FILTER_ASSERT(pFilter->State == FilterRunning);
    FILTER_ACQUIRE_LOCK(&pFilter->Lock, bFalse);
    pFilter->State = FilterPausing;
    FILTER_RELEASE_LOCK(&pFilter->Lock, bFalse);

    Status = NDIS_STATUS_SUCCESS;
    pFilter->State = FilterPaused;

    return Status;
}

_Use_decl_annotations_
NDIS_STATUS FilterRestart(NDIS_HANDLE FilterModuleContext,
PNDIS_FILTER_RESTART_PARAMETERS RestartParameters){
    NDIS_STATUS          Status;
    PMS_FILTER          pFilter = (PMS_FILTER)FilterModuleContext;
    NDIS_HANDLE          ConfigurationHandle = NULL;

    PNDIS_RESTART_GENERAL_ATTRIBUTES NdisGeneralAttributes;
    PNDIS_RESTART_ATTRIBUTES          NdisRestartAttributes;
    NDIS_CONFIGURATION_OBJECT          ConfigObject;

    FILTER_ASSERT(pFilter->State == FilterPaused);

    ConfigObject.Header.Type = NDIS_OBJECT_TYPE_CONFIGURATION_OBJECT;
    ConfigObject.Header.Revision =
NDIS_CONFIGURATION_OBJECT_REVISION_1;
    ConfigObject.Header.Size = sizeof(NDIS_CONFIGURATION_OBJECT);
    ConfigObject.NdisHandle = FilterDriverHandle;
    ConfigObject.Flags = 0;

    Status = NdisOpenConfigurationEx(&ConfigObject,
&ConfigurationHandle);

    if (Status == NDIS_STATUS_SUCCESS)
        NdisCloseConfiguration(ConfigurationHandle);

    NdisRestartAttributes = RestartParameters->RestartAttributes;

    if (NdisRestartAttributes != NULL){
        PNDIS_RESTART_ATTRIBUTES NextAttributes;
        ASSERT(NdisRestartAttributes->Oid ==
OID_GEN_MINIPORT_RESTART_ATTRIBUTES);
        NdisGeneralAttributes =
(PNDIS_RESTART_GENERAL_ATTRIBUTES)NdisRestartAttributes->Data;

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпи	Дата		

```

        NdisGeneralAttributes->LookaheadSize = 128;

        NextAttributes = NdisRestartAttributes->Next;
        while (NextAttributes != NULL)
            NextAttributes = NextAttributes->Next;
    }

    pFilter->State = FilterRunning;
    Status = NDIS_STATUS_SUCCESS;

    if (Status != NDIS_STATUS_SUCCESS)
        pFilter->State = FilterPaused;

    return Status;
}

_Use_decl_annotations_
VOID FilterDetach(NDIS_HANDLE FilterModuleContext){
    PMS_FILTER                pFilter =
(PMS_FILTER)FilterModuleContext;
    BOOLEAN                    bFalse = FALSE;

    FILTER_ASSERT(pFilter->State == FilterPaused);

    if (InterfaceThread){
        ZwClose(InterfaceThread);
        InterfaceThread = NULL;
    }
    if (pFilter->FilterName.Buffer != NULL)
        FILTER_FREE_MEM(pFilter->FilterName.Buffer);

    FILTER_ACQUIRE_LOCK(&FilterListLock, bFalse);
    RemoveEntryList(&pFilter->FilterModuleLink);
    FILTER_RELEASE_LOCK(&FilterListLock, bFalse);
    FILTER_FREE_MEM(pFilter);

    return;
}

_Use_decl_annotations_
VOID FilterUnload(PDRIVER_OBJECT DriverObject){
    #if DBG
        BOOLEAN                bFalse = FALSE;
    #endif
    UNREFERENCED_PARAMETER(DriverObject);
    NetShieldDeregisterDevice();
    NdisFDeregisterFilterDriver(FilterDriverHandle);

    #if DBG
        FILTER_ACQUIRE_LOCK(&FilterListLock, bFalse);
        ASSERT(IsListEmpty(&FilterModuleList));
        FILTER_RELEASE_LOCK(&FilterListLock, bFalse);
    #endif

    FILTER_FREE_LOCK(&FilterListLock);
    return;
}

```

```

Use_decl_annotations
NDIS_STATUS FilterOidRequest(NDIS_HANDLE
FilterModuleContext, PNDIS_OID_REQUEST Request) {
    PMS_FILTER          pFilter = (PMS_FILTER)FilterModuleContext;
    NDIS_STATUS          Status;
    PNDIS_OID_REQUEST    ClonedRequest=NULL;
    BOOLEAN              bSubmitted = FALSE;
    PFILTER_REQUEST_CONTEXT Context;
    BOOLEAN              bFalse = FALSE;

    do{
        Status = NdisAllocateCloneOidRequest(pFilter->FilterHandle,
Request, FILTER_TAG, &ClonedRequest);
        if (Status != NDIS_STATUS_SUCCESS)
            break;

        Context = (PFILTER_REQUEST_CONTEXT)(&ClonedRequest-
>SourceReserved[0]);
        *Context = Request;
        bSubmitted = TRUE;

        ClonedRequest->RequestId = Request->RequestId;
        pFilter->PendingOidRequest = ClonedRequest;

        Status = NdisFOidRequest(pFilter->FilterHandle,
ClonedRequest);
        if (Status != NDIS_STATUS_PENDING){
            FilterOidRequestComplete(pFilter, ClonedRequest, Status);
            Status = NDIS_STATUS_PENDING;
        }

    }while (bFalse);

    if (bSubmitted == FALSE){
        switch(Request->RequestType){
            case NdisRequestMethod:
                Request->DATA.METHOD_INFORMATION.BytesRead = 0;
                Request->DATA.METHOD_INFORMATION.BytesNeeded = 0;
                Request->DATA.METHOD_INFORMATION.BytesWritten = 0;
                break;
            case NdisRequestSetInformation:
                Request->DATA.SET_INFORMATION.BytesRead = 0;
                Request->DATA.SET_INFORMATION.BytesNeeded = 0;
                break;
            case NdisRequestQueryInformation:
            case NdisRequestQueryStatistics:
            default:
                Request->DATA.QUERY_INFORMATION.BytesWritten = 0;
                Request->DATA.QUERY_INFORMATION.BytesNeeded = 0;
                break;
        }
    }
    return Status;
}

Use_decl_annotations_
VOID FilterCancelOidRequest(NDIS_HANDLE FilterModuleContext, PVOID
RequestId) {

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпи	Дата		

```

        PMS_FILTER                                pFilter =
(PMS_FILTER)FilterModuleContext;
        PNDIS_OID_REQUEST                        Request = NULL;
        PFILTER_REQUEST_CONTEXT                  Context;
        PNDIS_OID_REQUEST                        OriginalRequest = NULL;
        BOOLEAN                                  bFalse = FALSE;
        FILTER_ACQUIRE_LOCK(&pFilter->Lock, bFalse);
        Request = pFilter->PendingOidRequest;

        if (Request != NULL){
            Context = (PFILTER_REQUEST_CONTEXT) (&Request-
>SourceReserved[0]);
            OriginalRequest = (*Context);
        }

        if ((OriginalRequest != NULL) && (OriginalRequest->RequestId ==
RequestId)){
            FILTER_RELEASE_LOCK(&pFilter->Lock, bFalse);
            NdisFCancelOidRequest(pFilter->FilterHandle, RequestId);
        }
        else
            FILTER_RELEASE_LOCK(&pFilter->Lock, bFalse);
    }

    _Use_decl_annotations_
VOID FilterOidRequestComplete(NDIS_HANDLE
FilterModuleContext, PNDIS_OID_REQUEST Request, NDIS_STATUS Status){
    PMS_FILTER                                pFilter =
(PMS_FILTER)FilterModuleContext;
    PNDIS_OID_REQUEST                        OriginalRequest;
    PFILTER_REQUEST_CONTEXT                  Context;
    BOOLEAN                                  bFalse = FALSE;

    Context = (PFILTER_REQUEST_CONTEXT) (&Request->SourceReserved[0]);
    OriginalRequest = (*Context);

    if (OriginalRequest == NULL){
        filterInternalRequestComplete(pFilter, Request, Status);
        return;
    }

    FILTER_ACQUIRE_LOCK(&pFilter->Lock, bFalse);
    ASSERT(pFilter->PendingOidRequest == Request);
    pFilter->PendingOidRequest = NULL;
    FILTER_RELEASE_LOCK(&pFilter->Lock, bFalse);

    switch(Request->RequestType){
        case NdisRequestMethod:
            OriginalRequest-
>DATA.METHOD_INFORMATION.OutputBufferLength = Request-
>DATA.METHOD_INFORMATION.OutputBufferLength;
            OriginalRequest->DATA.METHOD_INFORMATION.BytesRead =
Request->DATA.METHOD_INFORMATION.BytesRead;
            OriginalRequest->DATA.METHOD_INFORMATION.BytesNeeded =
Request->DATA.METHOD_INFORMATION.BytesNeeded;
            OriginalRequest->DATA.METHOD_INFORMATION.BytesWritten =
Request->DATA.METHOD_INFORMATION.BytesWritten;
            break;
        case NdisRequestSetInformation:

```

```

        OriginalRequest->DATA.SET_INFORMATION.BytesRead = Request-
>DATA.SET_INFORMATION.BytesRead;
        OriginalRequest->DATA.SET_INFORMATION.BytesNeeded =
Request->DATA.SET_INFORMATION.BytesNeeded;
        break;
        case NdisRequestQueryInformation:
        case NdisRequestQueryStatistics:
        default:
            OriginalRequest->DATA.QUERY_INFORMATION.BytesWritten =
Request->DATA.QUERY_INFORMATION.BytesWritten;
            OriginalRequest->DATA.QUERY_INFORMATION.BytesNeeded =
Request->DATA.QUERY_INFORMATION.BytesNeeded;
            break;
    }

    (*Context) = NULL;
    NdisFreeCloneOidRequest(pFilter->FilterHandle, Request);
    NdisFOidRequestComplete(pFilter->FilterHandle, OriginalRequest,
Status);
}

_Use_decl_annotations_
VOID FilterStatus(NDIS_HANDLE
FilterModuleContext, PNDIS_STATUS_INDICATION StatusIndication) {
    PMS_FILTER                pFilter = (PMS_FILTER)FilterModuleContext;
#ifdef DBG
    BOOLEAN                    bFalse = FALSE;
    FILTER_ACQUIRE_LOCK(&pFilter->Lock, bFalse);
    ASSERT(pFilter->bIndicating == FALSE);
    pFilter->bIndicating = TRUE;
    FILTER_RELEASE_LOCK(&pFilter->Lock, bFalse);
#endif

    NdisFIndicateStatus(pFilter->FilterHandle, StatusIndication);
}

_Use_decl_annotations_
VOID FilterDevicePnPEventNotify(NDIS_HANDLE
FilterModuleContext, PNET_DEVICE_PNP_EVENT NetDevicePnPEvent) {
    PMS_FILTER                pFilter = (PMS_FILTER)FilterModuleContext;
    NDIS_DEVICE_PNP_EVENT     DevicePnPEvent = NetDevicePnPEvent-
>DevicePnPEvent;

    switch (DevicePnPEvent) {
        case NdisDevicePnPEventQueryRemoved:
        case NdisDevicePnPEventRemoved:
        case NdisDevicePnPEventSurpriseRemoved:
        case NdisDevicePnPEventQueryStopped:
        case NdisDevicePnPEventStopped:
        case NdisDevicePnPEventPowerProfileChanged:
        case NdisDevicePnPEventFilterListChanged:
            break;
        default:
            DEBUGP(DL_ERROR, "FilterDevicePnPEventNotify: Invalid
event.\n");
            FILTER_ASSERT(bFalse);
            break;
    }
}

```

```

        NdisFDevicePnPEventNotify(pFilter->FilterHandle,
NetDevicePnPEvent);
    }

```

```

    Use_decl_annotations
NDIS_STATUS FilterNetPnPEvent(NDIS_HANDLE
FilterModuleContext,PNET_PNP_EVENT_NOTIFICATION
NetPnPEventNotification){
    PMS_FILTER                pFilter =
(PMS_FILTER)FilterModuleContext;
    NDIS_STATUS                Status = NDIS_STATUS_SUCCESS;
    Status = NdisFNetPnPEvent(pFilter->FilterHandle,
NetPnPEventNotification);
    return Status;
}

```

```

    Use_decl_annotations
VOID FilterSendNetBufferListsComplete(NDIS_HANDLE
FilterModuleContext,PNET_BUFFER_LIST NetBufferLists,ULONG
SendCompleteFlags){
    PMS_FILTER                pFilter = (PMS_FILTER)FilterModuleContext;
    ULONG                    NumOfSendCompletes = 0;
    BOOLEAN                  DispatchLevel;
    PNET_BUFFER_LIST         CurrNbl;

    if (pFilter->TrackSends){
        CurrNbl = NetBufferLists;
        while (CurrNbl){
            NumOfSendCompletes++;
            CurrNbl = NET_BUFFER_LIST_NEXT_NBL(CurrNbl);
        }
        DispatchLevel =
NDIS_TEST_SEND_AT_DISPATCH_LEVEL(SendCompleteFlags);
        FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
        pFilter->OutstandingSends -= NumOfSendCompletes;
        FILTER_LOG_SEND_REF(2, pFilter, PrevNbl, pFilter-
>OutstandingSends);
        FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
    }
    NdisFSendNetBufferListsComplete(pFilter->FilterHandle,
NetBufferLists, SendCompleteFlags);
}

```

```

    Use_decl_annotations
VOID FilterSendNetBufferLists(NDIS_HANDLE
FilterModuleContext,PNET_BUFFER_LIST NetBufferLists,NDIS_PORT_NUMBER
PortNumber,ULONG SendFlags){
    PMS_FILTER                pFilter = (PMS_FILTER)FilterModuleContext;
    PNET_BUFFER_LIST         CurrNbl;
    BOOLEAN                  DispatchLevel;
    BOOLEAN                  bFalse = FALSE;
    do{
        DispatchLevel = NDIS_TEST_SEND_AT_DISPATCH_LEVEL(SendFlags);

        if (pFilter->TrackSends){
            FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
            CurrNbl = NetBufferLists;
            while (CurrNbl){

```

```

        if(CryptoState)
            TracePackets(CurrNbl, FALSE);
        pFilter->OutstandingSends++;
        FILTER_LOG_SEND_REF(1, pFilter, CurrNbl, pFilter-
>OutstandingSends);
        CurrNbl = NET_BUFFER_LIST_NEXT_NBL(CurrNbl);
    }
    FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
}

NdisFSendNetBufferLists(pFilter->FilterHandle, NetBufferLists,
PortNumber, SendFlags);
}while (bFalse);
}

_Use_decl_annotations_
VOID FilterReturnNetBufferLists(NDIS_HANDLE
FilterModuleContext, PNET_BUFFER_LIST NetBufferLists, ULONG
ReturnFlags){
    PMS_FILTER          pFilter = (PMS_FILTER)FilterModuleContext;
    PNET_BUFFER_LIST     CurrNbl = NetBufferLists;
    UINT                 NumOfNetBufferLists = 0;
    BOOLEAN              DispatchLevel;
    ULONG               Ref;

    if (pFilter->TrackReceives){
        while (CurrNbl){
            NumOfNetBufferLists ++;
            CurrNbl = NET_BUFFER_LIST_NEXT_NBL(CurrNbl);
        }
    }

    NdisFReturnNetBufferLists(pFilter->FilterHandle, NetBufferLists,
ReturnFlags);

    if (pFilter->TrackReceives){
        DispatchLevel =
NDIS_TEST_RETURN_AT_DISPATCH_LEVEL(ReturnFlags);
        FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
        pFilter->OutstandingRcvS -= NumOfNetBufferLists;
        Ref = pFilter->OutstandingRcvS;
        FILTER_LOG_RCV_REF(3, pFilter, NetBufferLists, Ref);
        FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
    }
}

_Use_decl_annotations_
VOID FilterReceiveNetBufferLists(NDIS_HANDLE
FilterModuleContext, PNET_BUFFER_LIST NetBufferLists, NDIS_PORT_NUMBER
PortNumber, ULONG NumberOfNetBufferLists, ULONG ReceiveFlags){

    PMS_FILTER          pFilter = (PMS_FILTER)FilterModuleContext;
    BOOLEAN              DispatchLevel;
    ULONG               Ref;
    BOOLEAN              bFalse = FALSE;

    do{

```

```

        DispatchLevel =
NDIS_TEST_RECEIVE_AT_DISPATCH_LEVEL(ReceiveFlags);

        ASSERT(NumberOfNetBufferLists >= 1);

        if (pFilter->TrackReceives){
            FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
            pFilter->OutstandingRcvS += NumberOfNetBufferLists;
            Ref = pFilter->OutstandingRcvS;

            FILTER_LOG_RCV_REF(1, pFilter, NetBufferLists, Ref);
            FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
        }

        PNET_BUFFER_LIST CurrNbl = NetBufferLists;
        if (CryptoState)
            TracePackets(CurrNbl, TRUE);

        NdisFIndicateReceiveNetBufferLists(pFilter->FilterHandle,
NetBufferLists, PortNumber, NumberOfNetBufferLists, ReceiveFlags);

        if (NDIS_TEST_RECEIVE_CANNOT_PEND(ReceiveFlags) && pFilter-
>TrackReceives){
            FILTER_ACQUIRE_LOCK(&pFilter->Lock, DispatchLevel);
            pFilter->OutstandingRcvS -= NumberOfNetBufferLists;
            Ref = pFilter->OutstandingRcvS;
            FILTER_LOG_RCV_REF(2, pFilter, NetBufferLists, Ref);
            FILTER_RELEASE_LOCK(&pFilter->Lock, DispatchLevel);
        }
    } while (bFalse);
}

_Use_decl_annotations_
VOID FilterCancelSendNetBufferLists(NDIS_HANDLE
FilterModuleContext, PVOID CancelId){
    PMS_FILTER pFilter = (PMS_FILTER)FilterModuleContext;
    NdisFCancelSendNetBufferLists(pFilter->FilterHandle, CancelId);
}

_Use_decl_annotations_
NDIS_STATUS FilterSetModuleOptions(NDIS_HANDLE FilterModuleContext){
    PMS_FILTER pFilter =
(PMS_FILTER)FilterModuleContext;
    NDIS_FILTER_PARTIAL_CHARACTERISTICS OptionalHandlers;
    NDIS_STATUS Status =
NDIS_STATUS_SUCCESS;
    BOOLEAN bFalse = FALSE;

    if (bFalse){
        UINT i;
        pFilter->CallsRestart++;
        i = pFilter->CallsRestart % 8;
        pFilter->TrackReceives = TRUE;
        pFilter->TrackSends = TRUE;

        NdisMoveMemory(&OptionalHandlers, &DefaultChars,
sizeof(OptionalHandlers));
    }
}

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпи	Дата		

```

        OptionalHandlers.Header.Type =
NDIS_OBJECT_TYPE_FILTER_PARTIAL_CHARACTERISTICS;
        OptionalHandlers.Header.Size = sizeof(OptionalHandlers);
        switch (i){
            case 0:
                OptionalHandlers.ReceiveNetBufferListsHandler = NULL;
                pFilter->TrackReceives = FALSE;
                break;
            case 1:
                OptionalHandlers.ReturnNetBufferListsHandler = NULL;
                pFilter->TrackReceives = FALSE;
                break;
            case 2:
                OptionalHandlers.SendNetBufferListsHandler = NULL;
                pFilter->TrackSends = FALSE;
                break;
            case 3:
                OptionalHandlers.SendNetBufferListsCompleteHandler =
NULL;
                pFilter->TrackSends = FALSE;
                break;
            case 4:
                OptionalHandlers.ReceiveNetBufferListsHandler = NULL;
                OptionalHandlers.ReturnNetBufferListsHandler = NULL;
                break;
            case 5:
                OptionalHandlers.SendNetBufferListsHandler = NULL;
                OptionalHandlers.SendNetBufferListsCompleteHandler =
NULL;
                break;
            case 6:
                OptionalHandlers.ReceiveNetBufferListsHandler = NULL;
                OptionalHandlers.ReturnNetBufferListsHandler = NULL;
                OptionalHandlers.SendNetBufferListsHandler = NULL;
                OptionalHandlers.SendNetBufferListsCompleteHandler =
NULL;
                break;
            case 7:
                break;
        }
        Status = NdisSetOptionalHandlers(pFilter->FilterHandle,
(PNDIS_DRIVER_OPTIONAL_HANDLERS)&OptionalHandlers );
    }
    return Status;
}

_IRQL_requires_max_(DISPATCH_LEVEL)
NDIS_STATUS filterDoInternalRequest(_In_ PMS_FILTER
FilterModuleContext, _In_ NDIS_REQUEST_TYPE RequestType, _In_ NDIS_OID
Oid, _Inout_updates_bytes_to_(InformationBufferLength,
*pBytesProcessed) PVOID InformationBuffer, _In_ ULONG
InformationBufferLength, _In_opt_ ULONG OutputBufferLength, _In_ ULONG
MethodId, _Out_ PULONG pBytesProcessed){
    FILTER_REQUEST FilterRequest;
    PNDIS_OID_REQUEST NdisRequest = &FilterRequest.Request;
    NDIS_STATUS Status;

    ASSERT(pBytesProcessed != NULL);

```

```

    *pBytesProcessed = 0;
    NdisZeroMemory(NdisRequest, sizeof(NDIS_OID_REQUEST));
    NdisInitializeEvent(&FilterRequest.RegEvent);

    NdisRequest->Header.Type = NDIS_OBJECT_TYPE_OID_REQUEST;
    NdisRequest->Header.Revision = NDIS_OID_REQUEST_REVISION_1;
    NdisRequest->Header.Size = sizeof(NDIS_OID_REQUEST);
    NdisRequest->RequestType = RequestType;

    switch (RequestType){
        case NdisRequestQueryInformation:
            NdisRequest->DATA.QUERY_INFORMATION.Oid = Oid;
            NdisRequest->DATA.QUERY_INFORMATION.InformationBuffer =
InformationBuffer;
            NdisRequest-
>DATA.QUERY_INFORMATION.InformationBufferLength =
InformationBufferLength;
            break;
        case NdisRequestSetInformation:
            NdisRequest->DATA.SET_INFORMATION.Oid = Oid;
            NdisRequest->DATA.SET_INFORMATION.InformationBuffer =
InformationBuffer;
            NdisRequest->DATA.SET_INFORMATION.InformationBufferLength
= InformationBufferLength;
            break;
        case NdisRequestMethod:
            NdisRequest->DATA.METHOD_INFORMATION.Oid = Oid;
            NdisRequest->DATA.METHOD_INFORMATION.MethodId = MethodId;
            NdisRequest->DATA.METHOD_INFORMATION.InformationBuffer =
InformationBuffer;
            NdisRequest->DATA.METHOD_INFORMATION.InputBufferLength =
InformationBufferLength;
            NdisRequest->DATA.METHOD_INFORMATION.OutputBufferLength =
OutputBufferLength;
            break;
        default:
            ASSERTMSG("Invalid request type in
filterDoInternalRequest", FALSE);
            break;
    }

    NdisRequest->RequestId = (PVOID)FILTER_REQUEST_ID;

    Status = NdisFOidRequest(FilterModuleContext->FilterHandle,
NdisRequest);

    if (Status == NDIS_STATUS_PENDING){
        NdisWaitEvent(&FilterRequest.RegEvent, 0);
        Status = FilterRequest.Status;
    }

    if (Status == NDIS_STATUS_SUCCESS){
        if (RequestType == NdisRequestSetInformation)
            *pBytesProcessed = NdisRequest-
>DATA.SET_INFORMATION.BytesRead;

        if (RequestType == NdisRequestQueryInformation)
            *pBytesProcessed = NdisRequest-
>DATA.QUERY_INFORMATION.BytesWritten;
    }

```

```

        if (RequestType == NdisRequestMethod)
            *pBytesProcessed = NdisRequest->DATA.METHOD_INFORMATION.BytesWritten;

        if (RequestType == NdisRequestMethod){
            if (*pBytesProcessed > OutputBufferLength)
                *pBytesProcessed = OutputBufferLength;
        }
        else{
            if (*pBytesProcessed > InformationBufferLength)
                *pBytesProcessed = InformationBufferLength;
        }
    }
    return Status;
}

VOID filterInternalRequestComplete(_In_ NDIS_HANDLE
FilterModuleContext, _In_ PNDIS_OID_REQUEST NdisRequest, _In_
NDIS_STATUS Status){
    PFILTER_REQUEST FilterRequest;
    UNREFERENCED_PARAMETER(FilterModuleContext);

    FilterRequest = CONTAINING_RECORD(NdisRequest, FILTER_REQUEST,
Request);
    FilterRequest->Status = Status;

    NdisSetEvent(&FilterRequest->ReqEvent);
}

```

Файл Helpers.c

```

    CryptoState = FALSE;
    NdisFreeMemoryWithTag(KeyInfo, 'EXaM');

    ZwClose(RegKey);

    return 0;
}

void ClearHelpersObjects() {
    NetHelper* currentHelper = HelperObject;
    while (HelperObject != NULL) {
        int num = 0;
        while (currentHelper->next != NULL) {
            currentHelper = currentHelper->next;
            num++;
        }
        ExFreePoolWithTag(currentHelper->_cryptingKey, 'EXhP');
        ExFreePoolWithTag(currentHelper->_IpKey, 'EXhP');
        ExFreePoolWithTag(currentHelper, 'EXgP');

        currentHelper = HelperObject;
        if (num > 0) {
            for (int i = 0; i < num - 1; i++)
                currentHelper = currentHelper->next;

            currentHelper->next = NULL;
        }
    }
}

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпи	Дата		

```

        else
            HelperObject = NULL;
            currentHelper = HelperObject;
    }
}

NTSTATUS DriverControl(IN PDEVICE_OBJECT fdo, IN PIRP irp){
    ClearHelpersObjects();
    GetIpKeyFromReg(L"\\REGISTRY\\Machine\\SYSTEM\\ControlSet001\\NetShield\\");
    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;
    IoCompleteRequest(irp, IO_NO_INCREMENT);

    return STATUS_SUCCESS;
}

NTSTATUS DriverControlCreate(IN PDEVICE_OBJECT fdo, IN PIRP irp){
    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;
    IoCompleteRequest(irp, IO_NO_INCREMENT);
    return STATUS_SUCCESS;
}

NTSTATUS DriverControlClose(IN PDEVICE_OBJECT fdo, IN PIRP irp){
    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;
    IoCompleteRequest(irp, IO_NO_INCREMENT);

    return STATUS_SUCCESS;
}

NetHelper* CheckPacketIp(ULONG ip) {

    NetHelper* CryptingConfiguration = NULL;
    NetHelper* current_helper = HelperObject;

    while (current_helper != NULL) {
        ULONG IpKey = *(ULONG*)current_helper->_IpKey;
        if (IpKey == ip) {
            CryptingConfiguration = current_helper;
            break;
        }
        current_helper = current_helper->next;
    }
    return CryptingConfiguration;
}

void encryption(void* data,unsigned int dataSize,void* key, unsigned int keySize){
    unsigned char* px, *_key;

    px = (unsigned char*)data;
    _key = (unsigned char*)key;

```

```

        for (int i = 0; i < dataSize; i++, px++)
            *px = *px ^ _key[i % keySize];
    }

void ProcessNetBuffer(NET_BUFFER* nb, unsigned int dataOffset,
NetHelper* CryptingConfig) {
    unsigned int PacketDataSize = GetNetBufferDataSize(nb);
    unsigned int offset = sizeof(EthenetHeader) + dataOffset;
    while (offset < PacketDataSize) {
        unsigned int DataInPointer = 0;
        void* data = GetData(nb, offset, &DataInPointer);

        encryption(data, DataInPointer, CryptingConfig-
>_cryptingKey, CryptingConfig->_cryptingKeySize);

        offset += DataInPointer;
    }
}

VOID TraceIPv4Package(PNET_BUFFER CurrNb, BOOLEAN recive) {
    IPv4Header* ipHeader;
    unsigned int DataLimiter = 0;
    ipHeader = GetData(CurrNb, sizeof(EthenetHeader),
&DataLimiter);

    ULONG IpCheck;
    char* dest;

    if (recive) {
        IpCheck = ipHeader->SourAddress;
        dest = "Принят \n";
    }
    else {
        IpCheck = ipHeader->DestAddress;
        dest = "Отправлен \n";
    }

    NetHelper* CryptingConfig = CheckPacketIp(IpCheck);
    if(CryptingConfig != NULL){

        unsigned int DataOffset = 0;

        DataOffset += ipHeader->VerLen.HdrLen * 4;

        DbgPrintEx(DPFLTR_IHVNETWORK_ID, DPFLTR_INFO_LEVEL,
"Пакет отмеченного адреса: %s", dest);

        switch (ipHeader->Protocol){
            case TCP_TRANSPORT_PROTOCOL_TYPE: {
                TCPHeader* transportHeader = GetData(CurrNb,
sizeof(EthenetHeader) + DataOffset, NULL);
                DataOffset += sizeof(TCPHeader) + transportHeader-
>DataOffset*4;
                break;
            }
            case UDP_TRANSPORT_PROTOCOL_TYPE:
                DataOffset += sizeof(UDPHeader);
                break;
            default:

```

```

        return;
    }

    ProcessNetBuffer(CurrNb, DataOffset, CryptingConfig);
}

void TracePackets(PNET_BUFFER_LIST CurrNbl, BOOLEAN recive) {
    PNET_BUFFER CurrNb;
    CurrNb = NET_BUFFER_LIST_FIRST_NB(CurrNbl);
    PNET_BUFFER_LIST currList = CurrNbl;
    while (currList != NULL) {
        while (CurrNb != NULL) {

            EthenetHeader* Ehrenet = NULL;
            Ehrenet = GetData(CurrNb, 0, NULL);

            if (Ehrenet->ProtocolType == ETHERTYPE_IPV4)
                TraceIPv4Package(CurrNb, recive);

            CurrNb = CurrNb->Next;
        }
        currList = currList->Next;
    }
}

```

Файл MdlData.c

```

#include "precomp.h"
MDL* Get_Mdl_AtOffset(MDL* currentMdl, unsigned int* offset ) {

    BOOLEAN round_next = TRUE;
    MDL* mdl = currentMdl;

    if (offset == 0)
        return NULL;

    while (!!mdl && (round_next) ? (mdl->ByteCount <= *offset) :
(mdl->ByteCount < *offset)) {
        *offset -= mdl->ByteCount;
        mdl = mdl->Next;
    }

    return mdl;
}

MDL* Get_Mdl(NET_BUFFER* netbuffer, unsigned int* offset, unsigned
int* DataInPointer) {

    if (offset == 0) {
        if (DataInPointer != NULL)
            *DataInPointer = netbuffer->CurrentMdl->ByteCount
- netbuffer->CurrentMdlOffset;
        return netbuffer->CurrentMdl;
    }

    *offset += netbuffer->CurrentMdlOffset;

    MDL* mdl = Get_Mdl_AtOffset(netbuffer->CurrentMdl, offset);
}

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпи	Дата		

```

        if (DataInPointer != NULL)
            *DataInPointer = mdl->ByteCount - *offset;
        return mdl;
    }

void* Get_Buffer(MDL* mdl, unsigned int offset) {

    if (offset >= mdl->ByteCount)
        return NULL;

    void* buffer = MmGetSystemAddressForMdlSafe(mdl,
NormalPagePriority);
    (unsigned char*)buffer += offset;
    return buffer;
}

void* GetData(NET_BUFFER* netbuffer, unsigned int offset, unsigned
int* DataInPointer) {
    if(DataInPointer != NULL)
        *DataInPointer = 0;
    MDL* mdl = Get_Mdl(netbuffer, &offset, DataInPointer);

    void* data = (mdl) ? Get_Buffer(mdl, offset) : NULL;

    return data;
}

unsigned int GetNetBufferDataSize(NET_BUFFER* nb) {
    MDL* mdl = nb->CurrentMdl;
    unsigned int offset = nb->CurrentMdlOffset;
    unsigned int length = 0;

    while (mdl){
        length += mdl->ByteCount - offset;
        offset = 0;

        mdl = mdl->Next;
    }
    return length;
}

```

Файл Precomp.h

```

#pragma warning(disable:4201)
#include "Helpers.h"
#include "PacketOffsets.h"
#include <ndis.h>
#include <filteruser.h>
#include "flt_dbg.h"
#include "filter.h"

extern void GetIpKeyFromReg(WCHAR* Path);
extern void TracePackets(PNET_BUFFER_LIST CurrNbl, BOOLEAN recive);
extern void* GetData(NET_BUFFER* netbuffer, unsigned int offset,
unsigned int* DataInPointer);
extern unsigned int GetNetBufferDataSize(NET_BUFFER* nb);
extern NTSTATUS DriverControl(IN PDEVICE_OBJECT fdo, IN PIRP irp);
extern NTSTATUS DriverControlCreate(IN PDEVICE_OBJECT fdo, IN PIRP
irp);

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпи	Дата		

```
extern NTSTATUS DriverControlClose(IN PDEVICE_OBJECT fdo, IN PIRP
irp);
extern NDIS_HANDLE FilterDriverHandle;
BOOLEAN CryptoState;
```

Файл device.c

```
#include "precomp.h"
#pragma NDIS_INIT_FUNCTION(NetShieldRegisterDevice)

_IRQL_requires_max_(PASSIVE_LEVEL)
NTSTATUS NetShieldRegisterDevice(VOID) {
    NDIS_STATUS          Status = NDIS_STATUS_SUCCESS;
    UNICODE_STRING        DeviceName;
    UNICODE_STRING        DeviceLinkUnicodeString;
    PDRIVER_DISPATCH      DispatchTable[IRP_MJ_MAXIMUM_FUNCTION+1];
    NDIS_DEVICE_OBJECT_ATTRIBUTES DeviceAttribute;
    PFILTER_DEVICE_EXTENSION FilterDeviceExtension;
    PDRIVER_OBJECT        DriverObject;

    DEBUGP(DL_TRACE, "==>NetShieldRegisterDevice\n");

    NdisZeroMemory(DispatchTable, (IRP_MJ_MAXIMUM_FUNCTION+1) *
sizeof(PDRIVER_DISPATCH));

    DispatchTable[IRP_MJ_CREATE] = NetShieldDispatch;
    DispatchTable[IRP_MJ_CLEANUP] = NetShieldDispatch;
    DispatchTable[IRP_MJ_CLOSE] = NetShieldDispatch;
    DispatchTable[IRP_MJ_DEVICE_CONTROL] = NetShieldDeviceIoControl;

    NdisInitUnicodeString(&DeviceName, NTDEVICE_STRING);
    NdisInitUnicodeString(&DeviceLinkUnicodeString, LINKNAME_STRING);

    NdisZeroMemory(&DeviceAttribute,
sizeof(NDIS_DEVICE_OBJECT_ATTRIBUTES));

    DeviceAttribute.Header.Type =
NDIS_OBJECT_TYPE_DEVICE_OBJECT_ATTRIBUTES;
    DeviceAttribute.Header.Revision =
NDIS_DEVICE_OBJECT_ATTRIBUTES_REVISION_1;
    DeviceAttribute.Header.Size =
sizeof(NDIS_DEVICE_OBJECT_ATTRIBUTES);

    DeviceAttribute.DeviceName = &DeviceName;
    DeviceAttribute.SymbolicName = &DeviceLinkUnicodeString;
    DeviceAttribute.MajorFunctions = &DispatchTable[0];
    DeviceAttribute.ExtensionSize = sizeof(FILTER_DEVICE_EXTENSION);

    Status = NdisRegisterDeviceEx(FilterDriverHandle,
&DeviceAttribute, NdisDeviceObject, &NdisFilterDeviceHandle);

    if (Status == NDIS_STATUS_SUCCESS) {
        FilterDeviceExtension = (PFILTER_DEVICE_EXTENSION)
NdisGetDeviceReservedExtension(NdisDeviceObject);

        FilterDeviceExtension->Signature = 'FTDR';
        FilterDeviceExtension->Handle = FilterDriverHandle;

        DriverObject = (PDRIVER_OBJECT)FilterDriverObject;
```

					ПГУ 1.090301.37.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпи	Дата		

```

        DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
DriverControl;
        DriverObject->MajorFunction[IRP_MJ_CREATE] =
DriverControlCreate;
        DriverObject->MajorFunction[IRP_MJ_CLOSE] =
DriverControlClose;
    }

    DEBUGP(DL_TRACE, "<==NetShieldRegisterDevice: %x\n", Status);

    return (Status);
}

_IRQL_requires_max (PASSIVE_LEVEL)
VOID NetShieldDeregisterDevice(VOID) {
    if (NdisFilterDeviceHandle != NULL){
        NdisDeregisterDeviceEx(NdisFilterDeviceHandle);
    }

    NdisFilterDeviceHandle = NULL;
}

_Use_decl_annotations_
NTSTATUS NetShieldDispatch(PDEVICE_OBJECT DeviceObject,PIRP Irp){
    PIO_STACK_LOCATION      IrpStack;
    NTSTATUS                 Status = STATUS_SUCCESS;

    UNREFERENCED_PARAMETER(DeviceObject);

    IrpStack = IoGetCurrentIrpStackLocation(Irp);

    switch (IrpStack->MajorFunction){
        case IRP_MJ_CREATE:
            break;

        case IRP_MJ_CLEANUP:
            break;

        case IRP_MJ_CLOSE:
            break;

        default:
            break;
    }

    Irp->IoStatus.Status = Status;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);

    return Status;
}

_Use_decl_annotations_
NTSTATUS NetShieldDeviceIoControl(PDEVICE_OBJECT DeviceObject,PIRP
Irp){
    PIO_STACK_LOCATION      IrpSp;
    NTSTATUS                 Status = STATUS_SUCCESS;
    PFILTER_DEVICE_EXTENSION FilterDeviceExtension;
    PCHAR                    InputBuffer;
    PCHAR                    OutputBuffer;

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпи	Дата		

```

        ULONG                InputBufferLength, OutputBufferLength;
        PLIST_ENTRY          Link;
        PCHAR                pInfo;
        ULONG                InfoLength = 0;
        PMS_FILTER           pFilter = NULL;
        BOOLEAN              bFalse = FALSE;

        UNREFERENCED_PARAMETER(DeviceObject);

        IrpSp = IoGetCurrentIrpStackLocation(Irp);

        if (IrpSp->FileObject == NULL){
            return(STATUS_UNSUCCESSFUL);
        }

        FilterDeviceExtension =
        (PFILTER_DEVICE_EXTENSION)NdisGetDeviceReservedExtension(DeviceObject)
        ;

        ASSERT(FilterDeviceExtension->Signature == 'FTDR');

        Irp->IoStatus.Information = 0;

        switch (IrpSp->Parameters.DeviceIoControl.IoControlCode){

            case IOCTL_FILTER_RESTART_ALL:
                break;

            case IOCTL_FILTER_RESTART_ONE_INSTANCE:
                InputBuffer = OutputBuffer = (PCHAR)Irp-
                >AssociatedIrp.SystemBuffer;
                InputBufferLength = IrpSp-
                >Parameters.DeviceIoControl.InputBufferLength;

                pFilter = filterFindFilterModule (InputBuffer,
                InputBufferLength);

                if (pFilter == NULL)
                    break;

                NdisFRestartFilter(pFilter->FilterHandle);

                break;

            case IOCTL_FILTER_ENUMERATE_ALL_INSTANCES:

                InputBuffer = OutputBuffer = (PCHAR)Irp-
                >AssociatedIrp.SystemBuffer;
                InputBufferLength = IrpSp-
                >Parameters.DeviceIoControl.InputBufferLength;
                OutputBufferLength = IrpSp-
                >Parameters.DeviceIoControl.OutputBufferLength;

                pInfo = OutputBuffer;

                FILTER_ACQUIRE_LOCK(&FilterListLock, bFalse);

                Link = FilterModuleList.Flink;

```

```

        while (Link != &FilterModuleList){
            pFilter = CONTAINING_RECORD(Link, MS_FILTER,
FilterModuleLink);

            InfoLength += (pFilter->FilterModuleName.Length +
sizeof(USHORT));

            if (InfoLength <= OutputBufferLength){
                *(PUSHORT)pInfo = pFilter-
>FilterModuleName.Length;
                NdisMoveMemory(pInfo +
sizeof(USHORT), (PUCHAR) (pFilter->FilterModuleName.Buffer), pFilter-
>FilterModuleName.Length);
                pInfo += (pFilter->FilterModuleName.Length +
sizeof(USHORT));
            }
            Link = Link->Flink;
        }

        FILTER_RELEASE_LOCK(&FilterListLock, bFalse);
        if (InfoLength <= OutputBufferLength)
            Status = NDIS_STATUS_SUCCESS;
        else
            Status = STATUS_BUFFER_TOO_SMALL;
        break;

    default:
        break;
}

Irp->IoStatus.Status = Status;
Irp->IoStatus.Information = InfoLength;

IoCompleteRequest(Irp, IO_NO_INCREMENT);

return Status;
}

_IRQL_requires_max_(DISPATCH_LEVEL)
PMS_FILTER filterFindFilterModule(_In_reads_bytes_(BufferLength)
PUCHAR Buffer, _In_ ULONG BufferLength){
    PMS_FILTER            pFilter;
    PLIST_ENTRY            Link;
    BOOLEAN                bFalse = FALSE;

    FILTER_ACQUIRE_LOCK(&FilterListLock, bFalse);

    Link = FilterModuleList.Flink;

    while (Link != &FilterModuleList){
        pFilter = CONTAINING_RECORD(Link, MS_FILTER, FilterModuleLink);

        if (BufferLength >= pFilter->FilterModuleName.Length){
            if (NdisEqualMemory(Buffer, pFilter-
>FilterModuleName.Buffer, pFilter->FilterModuleName.Length)){
                FILTER_RELEASE_LOCK(&FilterListLock, bFalse);
                return pFilter;
            }
        }
    }
}

```

```

    }

    Link = Link->Flink;
}

FILTER_RELEASE_LOCK(&FilterListLock, bFalse);
return NULL;
}

```

Файл Helpers.h

```

#include <ntdef.h>
typedef unsigned char UCHAR;

typedef struct _Helper{
    void* _IpKey;
    int _IpKeySize;
    void* cryptingKey;
    int _cryptingKeySize;
    struct _Helper* next;
} NetHelper;

typedef struct _EthenetHeader{
    UCHAR DestanationMac[6];
    UCHAR SourceMac[6];
    USHORT ProtocolType;
} EthenetHeader;

typedef struct _VersionLendth{
    UCHAR HdrLen : 4;
    UCHAR Version : 4;
} VersionLendth;

typedef union _IPv4Flags{
    struct{
        USHORT FrgmOffset1 : 5;
        USHORT MoreFrgm : 1;
        USHORT DontFrgm : 1;
        USHORT Reserved : 1;
        USHORT FrgmOffset2 : 8;
    };
    USHORT AllFlags;
} IPv4Flags;

typedef struct _IPv4Header{
    VersionLendth VerLen;
    UCHAR ServiceType;
    USHORT PacketLength;
    USHORT Identifier;
    IPv4Flags FlagFragment;
    UCHAR TimeLife;
    UCHAR Protocol;
    USHORT CheckSum;
    ULONG SourAddress;
    ULONG DestAddress;
} IPv4Header;

typedef struct _UDPHeader{

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпи	Дата		

```

        USHORT      PortSrc;
        USHORT      PortDst;
        USHORT      Length;
        USHORT      Checksum;
    } UDPHeader;

typedef struct _TCPHeader{
    USHORT      PortSrc;
    USHORT      PortDst;
    ULONG      SeqNumber;
    ULONG      AckNumber;
    UCHAR      Reserved : 4;
    UCHAR      DataOffset : 4;
    UCHAR      Flags;
    USHORT      Window;
    USHORT      Checksum;
    USHORT      UrgPointer;
} TCPHeader;

typedef struct _ARPHeader{
    USHORT      DevType;
    USHORT      ProtoType;
    UCHAR      HaLen;
    UCHAR      PaLen;
    USHORT      Operation;
    UCHAR      SrcMac[6];
    ULONG      SrcIp;
    UCHAR      DstMac[6];
    ULONG      DstIp;
} ARPHeader;

extern NetHelper* HelperObject;

```

Файл filter.h

```

#ifndef _FILT_H
#define _FILT_H

#pragma warning(disable:28930)
#pragma warning(disable:28931)

#define FILTER_REQUEST_ID      'RTLF'
#define FILTER_ALLOC_TAG      'tliF'
#define FILTER_TAG      'dnTF'

#define FILTER_MAJOR_NDIS_VERSION      6

#if defined(NDIS60)
#define FILTER_MINOR_NDIS_VERSION      0
#elif defined(NDIS620)
#define FILTER_MINOR_NDIS_VERSION      20
#elif defined(NDIS630)
#define FILTER_MINOR_NDIS_VERSION      30
#endif

extern NDIS_HANDLE      FilterDriverHandle;
extern NDIS_HANDLE      FilterDriverObject;
extern NDIS_HANDLE      NdisFilterDeviceHandle;

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпи	Дата		

```

extern PDEVICE_OBJECT      NdisDeviceObject;

extern FILTER_LOCK         FilterListLock;
extern LIST_ENTRY          FilterModuleList;

#define FILTER_FRIENDLY_NAME      L"NetShield NDIS LightWeight Filter"
#define FILTER_UNIQUE_NAME      L"{1ae0650a-0c21-4fe8-8b50-alee76a0deb8}"
#define FILTER_SERVICE_NAME      L"NetShield"

#define LINKNAME_STRING          L"\\DosDevices\\NetShield"
#define NTDEVICE_STRING          L"\\Device\\NetShield"

typedef struct _QUEUE_ENTRY{
    struct _QUEUE_ENTRY * Next;
}QUEUE_ENTRY, *PQUEUE_ENTRY;

typedef struct _QUEUE_HEADER{
    PQUEUE_ENTRY      Head;
    PQUEUE_ENTRY      Tail;
} QUEUE_HEADER, PQUEUE_HEADER;

#if TRACK_RECEIVES
UINT      filterLogReceiveRefIndex = 0;
ULONG_PTR filterLogReceiveRef[0x10000];
#endif

#if TRACK_SENDS
UINT      filterLogSendRefIndex = 0;
ULONG_PTR filterLogSendRef[0x10000];
#endif

#if TRACK_RECEIVES
#define FILTER_LOG_RCV_REF(_O, _Instance, _NetBufferList, _Ref){
    filterLogReceiveRef[filterLogReceiveRefIndex++] =
    (ULONG_PTR) (_O);
    filterLogReceiveRef[filterLogReceiveRefIndex++] =
    (ULONG_PTR) (_Instance);
    filterLogReceiveRef[filterLogReceiveRefIndex++] =
    (ULONG_PTR) (_NetBufferList);
    filterLogReceiveRef[filterLogReceiveRefIndex++] =
    (ULONG_PTR) (_Ref);
    if (filterLogReceiveRefIndex >= (0x10000 - 5))
        filterLogReceiveRefIndex = 0;
}
#else
#define FILTER_LOG_RCV_REF(_O, _Instance, _NetBufferList, _Ref)
#endif

#if TRACK_SENDS
#define FILTER_LOG_SEND_REF(_O, _Instance, _NetBufferList, _Ref) {
    filterLogSendRef[filterLogSendRefIndex++] = (ULONG_PTR) (_O);
    filterLogSendRef[filterLogSendRefIndex++] =
    (ULONG_PTR) (_Instance);
    filterLogSendRef[filterLogSendRefIndex++] =
    (ULONG_PTR) (_NetBufferList);
    filterLogSendRef[filterLogSendRefIndex++] = (ULONG_PTR) (_Ref);
}

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						104
Изм.	Лист	№ докум.	Подпи	Дата		

```

        if (filterLogSendRefIndex >= (0x10000 - 5))
            filterLogSendRefIndex = 0;
    }

#else
#define FILTER_LOG_SEND_REF(_O, _Instance, _NetBufferList, _Ref)
#endif

#define FILTER_ALLOC_MEM(_NdisHandle, _Size) \
    NdisAllocateMemoryWithTagPriority(_NdisHandle, _Size, \
    FILTER_ALLOC_TAG, LowPoolPriority)

#define FILTER_FREE_MEM(_pMem) NdisFreeMemory(_pMem, 0, 0)

#define FILTER_INIT_LOCK(_pLock) NdisAllocateSpinLock(_pLock)

#define FILTER_FREE_LOCK(_pLock) NdisFreeSpinLock(_pLock)

#define FILTER_ACQUIRE_LOCK(_pLock, DispatchLevel){
    if (DispatchLevel)
        NdisDprAcquireSpinLock(_pLock);

    else
        NdisAcquireSpinLock(_pLock);
}

#define FILTER_RELEASE_LOCK(_pLock, DispatchLevel){
    if (DispatchLevel)
        NdisDprReleaseSpinLock(_pLock);
    else
        NdisReleaseSpinLock(_pLock);
}
#endif

#define NET_BUFFER_LIST_LINK_TO_ENTRY(_pNBL)
((PQUEUE_ENTRY) (NET_BUFFER_LIST_NEXT_NBL(_pNBL)))
#define ENTRY_TO_NET_BUFFER_LIST(_pEnt)
(CONTAINING_RECORD((_pEnt), NET_BUFFER_LIST, Next))

#define InitializeQueueHeader(_QueueHeader){
    (_QueueHeader)->Head = (_QueueHeader)->Tail = NULL;
}

#define IsQueueEmpty(_QueueHeader) (( _QueueHeader)->Head == NULL)

#define RemoveHeadQueue(_QueueHeader)
{
    (_QueueHeader)->Head;{
        PQUEUE_ENTRY pNext;
        ASSERT(( _QueueHeader)->Head);
        pNext = (_QueueHeader)->Head->Next;
        (_QueueHeader)->Head = pNext;
        if (pNext == NULL)
            (_QueueHeader)->Tail = NULL;
    }
}

#define InsertHeadQueue(_QueueHeader, _QueueEntry){
    ((PQUEUE_ENTRY) (_QueueEntry))->Next = (_QueueHeader)->Head;
    (_QueueHeader)->Head = (PQUEUE_ENTRY) (_QueueEntry);
}

```

```

        if (( QueueHeader)->Tail == NULL)
            (_QueueHeader)->Tail = (PQUEUE_ENTRY) (_QueueEntry);
    }

#define InsertTailQueue( _QueueHeader, _QueueEntry){
    ((PQUEUE_ENTRY) (_QueueEntry))->Next = NULL;
    if (( _QueueHeader)->Tail)
        (_QueueHeader)->Tail->Next = (PQUEUE_ENTRY) (_QueueEntry);
    else
        (_QueueHeader)->Head = (PQUEUE_ENTRY) (_QueueEntry);
    (_QueueHeader)->Tail = (PQUEUE_ENTRY) (_QueueEntry);
}

typedef enum FILTER_STATE{
    FilterStateUnspecified,
    FilterInitialized,
    FilterPausing,
    FilterPaused,
    FilterRunning,
    FilterRestarting,
    FilterDetaching
} FILTER_STATE;

typedef struct _FILTER_REQUEST{
    NDIS_OID REQUEST      Request;
    NDIS_EVENT             ReqEvent;
    NDIS_STATUS            Status;
} FILTER_REQUEST, *PFILTER_REQUEST;

typedef struct _MS_FILTER{
    LIST_ENTRY              FilterModuleLink;
    ULONG                   RefCount;

    NDIS_HANDLE             FilterHandle;
    NDIS_STRING             FilterModuleName;
    NDIS_STRING             MiniportFriendlyName;
    NDIS_STRING             MiniportName;
    NET_IFINDEX             MiniportIfIndex;

    NDIS_STATUS             Status;
    NDIS_EVENT              Event;
    ULONG                   BackFillSize;
    FILTER_LOCK             Lock;
    FILTER_STATE            State;
    ULONG                   OutstandingSends;
    ULONG                   OutstandingRequest;
    ULONG                   OutstandingRcv;
    FILTER_LOCK             SendLock;
    FILTER_LOCK             RcvLock;
    QUEUE_HEADER            SendNBLQueue;
    QUEUE_HEADER            RcvNBLQueue;

    NDIS_STRING             FilterName;
    ULONG                   CallsRestart;
    BOOLEAN                 TrackReceives;
    BOOLEAN                 TrackSends;
}
#endif DBG

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						106
Изм.	Лист	№ докум.	Подпи	Дата		

```

        BOOLEAN                                bIndicating;
#endif
        PNDIS_OID_REQUEST                      PendingOidRequest;

}MS_FILTER, * PMS_FILTER;

typedef struct _FILTER_DEVICE_EXTENSION{
        ULONG                                Signature;
        NDIS_HANDLE                          Handle;
} FILTER_DEVICE_EXTENSION, *PFILTER_DEVICE_EXTENSION;

#define FILTER_READY_TO_PAUSE( Filter)
        ((Filter)->State == FilterPausing)

typedef struct _FL_NDIS_FILTER_LIST{
        LIST_ENTRY                            Link;
        NDIS_HANDLE                          ContextHandle;
        NDIS_STRING                          FilterInstanceName;
} FL_NDIS_FILTER_LIST, *PFL_NDIS_FILTER_LIST;

typedef struct _NDIS_OID_REQUEST
*FILTER_REQUEST_CONTEXT,**PFILTER_REQUEST_CONTEXT;

EXTERN_C_START
DRIVER_INITIALIZE DriverEntry;
FILTER_SET_OPTIONS FilterRegisterOptions;
FILTER_ATTACH FilterAttach;
FILTER_DETACH FilterDetach;
DRIVER_UNLOAD FilterUnload;
FILTER_RESTART FilterRestart;
FILTER_PAUSE FilterPause;
FILTER_OID_REQUEST FilterOidRequest;
FILTER_CANCEL_OID_REQUEST FilterCancelOidRequest;
FILTER_STATUS FilterStatus;
FILTER_DEVICE_PNP_EVENT_NOTIFY FilterDevicePnPEventNotify;
FILTER_NET_PNP_EVENT FilterNetPnPEvent;
FILTER_OID_REQUEST_COMPLETE FilterOidRequestComplete;
FILTER_SEND_NET_BUFFER_LISTS FilterSendNetBufferLists;
FILTER_RETURN_NET_BUFFER_LISTS FilterReturnNetBufferLists;
FILTER_SEND_NET_BUFFER_LISTS_COMPLETE
FilterSendNetBufferListsComplete;
FILTER_RECEIVE_NET_BUFFER_LISTS FilterReceiveNetBufferLists;
FILTER_CANCEL_SEND_NET_BUFFER_LISTS FilterCancelSendNetBufferLists;
FILTER_SET_MODULE_OPTIONS FilterSetModuleOptions;

_IRQL_requires_max_(PASSIVE_LEVEL) NDIS_STATUS
NetShieldRegisterDevice(VOID);
_IRQL_requires_max_(PASSIVE_LEVEL) VOID
NetShieldDeregisterDevice(VOID);

DRIVER_DISPATCH NetShieldDispatch;
DRIVER_DISPATCH NetShieldDeviceIoControl;

_IRQL_requires_max_(DISPATCH_LEVEL) PMS_FILTER
filterFindFilterModule(_In_reads_bytes_(BufferLength) PUCCHAR
Buffer, _In_ ULONG BufferLength);

```

```

    IRQL requires max (DISPATCH_LEVEL) NDIS STATUS
    filterDoInternalRequest(_In_ PMS_FILTER FilterModuleContext, _In_
    NDIS_REQUEST_TYPE RequestType, _In_ NDIS_OID Oid,
    _Inout_updates_bytes_to_ (InformationBufferLength, *pBytesProcessed)
    PVOID InformationBuffer, _In_ ULONG InformationBufferLength, _In_opt_
    ULONG OutputBufferLength, _In_ ULONG MethodId, _Out_ PULONG
    pBytesProcessed);

VOID
filterInternalRequestComplete(_In_ NDIS_HANDLE
FilterModuleContext, _In_ PNDIS_OID_REQUEST NdisRequest, _In_
NDIS_STATUS Status);

EXTERN_C_END

#endif

```

Файл NetshieldInterface.ch

```

using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading;
using System.Runtime.InteropServices;
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;

namespace NetShieldInterface{
    public partial class MainForm : Form{
        const uint GENERIC_READ = 0x80000000;
        const uint OPEN_EXISTING = 3;
        const uint FILE_SHARE_READ = 1;
        const uint FILE_ATTRIBUTE_NORMAL = 128;
        const uint FILE_FLAG_OVERLAPPED = 0x40000000;

        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern IntPtr CreateFile(
            string lpFileName,
            uint dwDesiredAccess,
            uint dwShareMode,
            IntPtr lpSecurityAttributes,
            uint dwCreationDisposition,
            uint dwFlagsAndAttributes,
            IntPtr hTemplateFile
        );

        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern int DeviceIoControl(
            IntPtr hDevice,
            uint dwIoControlCode,
            IntPtr lpInBuffer,

```

					ПГУ 1.090301.37.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпи	Дата		

```

        uint nInBufferSize,
        IntPtr lpOutBuffer,
        uint nOutBufferSize,
        ref uint lpBytesReturned,
        IntPtr lpOverlapped
    );

[DllImport("kernel32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
public static extern bool CloseHandle(IntPtr hObject);

private DataTable TableData;
RegistryKey RegistryInfoKey = Registry.LocalMachine;
string RegPath = "SYSTEM\\ControlSet001\\NetShield";
Boolean CurrentState;
public MainForm() {
    InitializeComponent();
    InitTable();
}

private void InitTable() {
    TableData = new DataTable();
    TableData.Columns.Add("Ip-адрес", typeof(String));
    TableData.Columns.Add("Крипто-ключ", typeof(String));

    RegistryKey NetShieldReg =
RegistryInfoKey.OpenSubKey(RegPath, true);
    if (NetShieldReg == null) {
        String[] tmp = new String[1];
        tmp[0] = "";
        RegistryInfoKey.CreateSubKey(RegPath);
        RegistryInfoKey.OpenSubKey(RegPath,
true).SetValue("State", "OFF", RegistryValueKind.String);
        RegistryInfoKey.OpenSubKey(RegPath,
true).SetValue("Ip", tmp, RegistryValueKind.MultiString);
        RegistryInfoKey.OpenSubKey(RegPath,
true).SetValue("CryptoKey", tmp, RegistryValueKind.MultiString);

        CurrentState = false;
        StateButton.Text = "Включить защиту сети";
        dataGridView1.DataSource = TableData;
        return;
    }

    String[] IpRegData =
(String[])NetShieldReg.GetValue("Ip", null);
    String[] CryptRegData =
(String[])NetShieldReg.GetValue("CryptoKey", null);

    if (NetShieldReg.GetValue("State").ToString() == "OFF")
        CurrentState = false;
    else {
        CurrentState = true;
        StateButton.Text = "Выключить защиту сети";
    }

    String[] CryptData = CryptRegData[0].Split(';');
    String[] IpData = IpRegData[0].Split(';');

```

```

        for(int i = 0; i < IpData.Length; i++){
            TableData.Rows.Add(IpData[i], CryptData[i]);
        }

        dataGridView1.DataSource = TableData;
        NetShieldReg.Close();
        return;
    }

    private void SaveButton_Click(object sender, EventArgs e){
        RegistryKey NetShieldReg =
        RegistryInfoKey.OpenSubKey(RegPath, true);
        String IpData = "", CryptKeyData = "";

        for(int i = 0; i < TableData.Rows.Count; i++){
            IpData += TableData.Rows[i][0].ToString() + ";";
            CryptKeyData += TableData.Rows[i][1].ToString() + ";";
        }

        NetShieldReg.SetValue("Ip", IpData.Substring(0,
        IpData.Length - 1).Split(' '),RegistryValueKind.MultiString);
        NetShieldReg.SetValue("CryptoKey",
        CryptKeyData.Substring(0,CryptKeyData.Length -1).Split(' '),
        RegistryValueKind.MultiString);
        NetShieldReg.Close();

        IntPtr hDevice = CreateFile("\\\\.\\NetShield",
        GENERIC_READ, FILE_SHARE_READ, IntPtr.Zero, OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, IntPtr.Zero);

        if (hDevice != IntPtr.Zero){
            uint bytesReturned = 0;
            DeviceIoControl( hDevice, 0x00000000, IntPtr.Zero, 0,
            IntPtr.Zero,0, ref bytesReturned, IntPtr.Zero);
            CloseHandle(hDevice);
        }
    }

    private void On_Off_StateButton_Click(object sender, EventArgs
    e){
        RegistryKey NetShieldReg =
        RegistryInfoKey.OpenSubKey(RegPath, true);
        string newState;

        if (CurrentState){
            CurrentState = false;
            StateButton.Text = "Включить защиту сети";
            newState = "OFF";
        }
        else{
            CurrentState = true;
            StateButton.Text = "Выключить защиту сети";
            newState = "ON";
        }
        NetShieldReg.SetValue("State",
        newState,RegistryValueKind.String);
        NetShieldReg.Close();
    }
}

```

Приложение В

(Обязательное)

Программа реализации защищённого соединения

Презентация

					ПГУ 1.090301.37.001 ПЗ	Лист
						111
Изм.	Лист	№ докум.	Подпи	Дата		

Выпускная квалификационная работа на тему «Программа реализации защищённого соединения»

Выполнил:
студент группы 20ВВП2
Тихонов А.А.

Цели работы

Основной целью поставленной данной выпускной квалификационной работы является разработка программного обеспечения, которое будет обеспечивать шифрование входящего и исходящего сетевого трафика, для повышения уровня безопасности при передаче данных по сети.

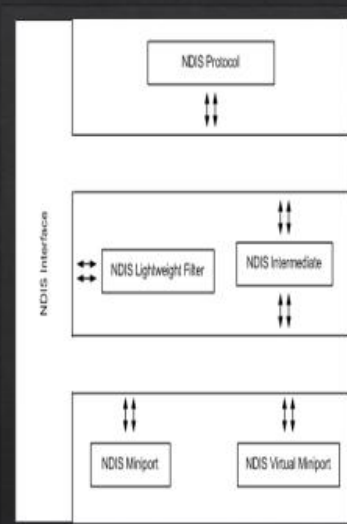
С помощью разработанного программного обеспечения возможно решить следующие проблемы:

- ☐ Защита передаваемых сетевых пакетов от перехвата
- ☐ Предотвращение утечек данных
- ☐ Защита конфиденциальных данных в интернете
- ☐ Повышение общей безопасности передачи данных через сетевой адаптер

Разработанное программное обеспечение

В ходе выполнения выпускной квалификационной работы был разработан следующие программные решения:

- ❑ NDIS драйвер-фильтр – сетевой драйвер, отвечающий за получение, обработку, в которую включено шифрование, и отправку сетевого трафика. Драйвер написан на языке С с использованием функционала библиотеки NDIS.
- ❑ Графический интерфейс настройки драйвера – графический интерфейс, написанный на языке С#, с помощью которого выполняется настройка и управления шифрованием сетевого трафика в разработанном сетевом драйвере.

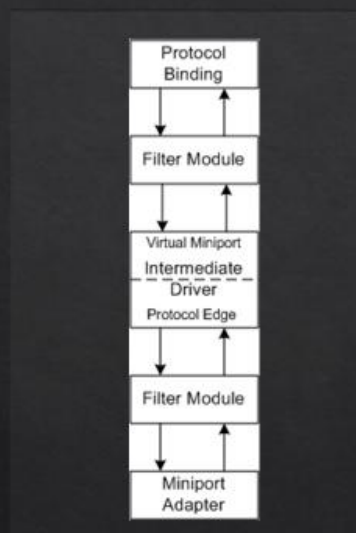


Драйвер-фильтр

Разработанный драйвер представляет собой один из подтипов сетевых драйверов NDIS, а именно NDIS фильтр.

Данный тип драйверов при установке встраивается в стек сетевых драйверов и получает все проходящие через данный стек сетевые пакеты.

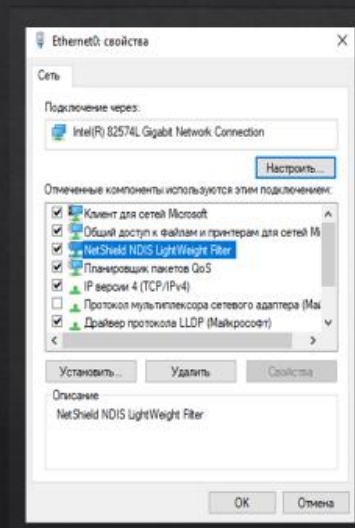
Этот тип драйвера идеально подходит для решения поставленной задачи, так как позволяет перехватить и проанализировать весь проходящий через сетевой адаптер трафик, начиная от системных пакетов, таких как LLDP, ARP и других, заканчивая пакетами которые получают конечные приложения (например различные загрузчики или приложения для видеозвонков, такие как Zoom) или браузеры.



Драйвер-фильтр

Стоит отметить, что установка драйвера производится только на один конкретный сетевой адаптер, что позволяет обеспечивать шифрованием соединения на конкретном сетевом адаптере, если в системе присутствует несколько адаптеров. Но в тоже возможно установить драйвер сразу на несколько сетевых адаптеров.

Так как после установки драйвер относится к конкретному адаптеру, то и весь сетевой трафик он будет перехватывать только на том сетевом адаптере, на который он был установлен.

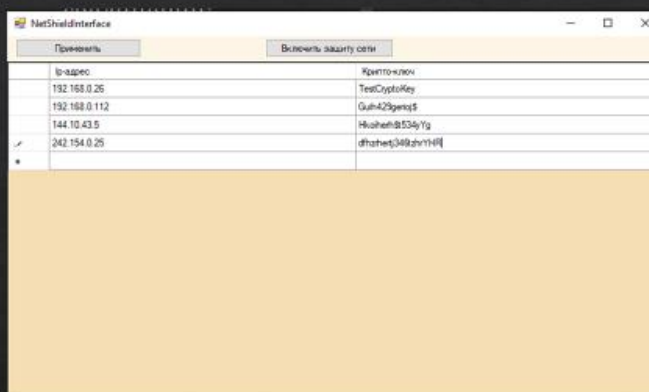


Приложение настроек драйвера

Приложение с настройками драйвера представляет собой таблицу с двумя полями, которые являются настройками защищённых соединений:

- IP-адрес
- Криптографический ключ

Так же через приложение можно включить или выключить шифрование сетевых пакетов и применить настройки в драйвере, отправив ему IOCTL запрос.



Какие сетевые пакеты подлежат шифрованию?

Разработанный драйвер поддерживает шифрование только тех сетевых пакетов в который заголовком сетевого уровня выступает IPv4, а заголовками транспортного уровня являются TCP или UDP. Всё что находится выше TCP или UDP заголовков не имеет для нас особого значения, так как не влияет на передачу данных в сети, а лишь содержит данные, необходимые для работы приложений или самой системы, которые будут расшифрованы перед передачей конечным приложениям.

Стоит отметить что, часть работы драйвера составляет выделение сетевых пакетов которые подлежат шифрованию из всего сетевого трафика.



Анализ сетевого пакета и подготовка к его шифрованию

После того как сетевой пакет попал в драйвер, необходимо выделить из него следующие сетевые заголовки для анализа:

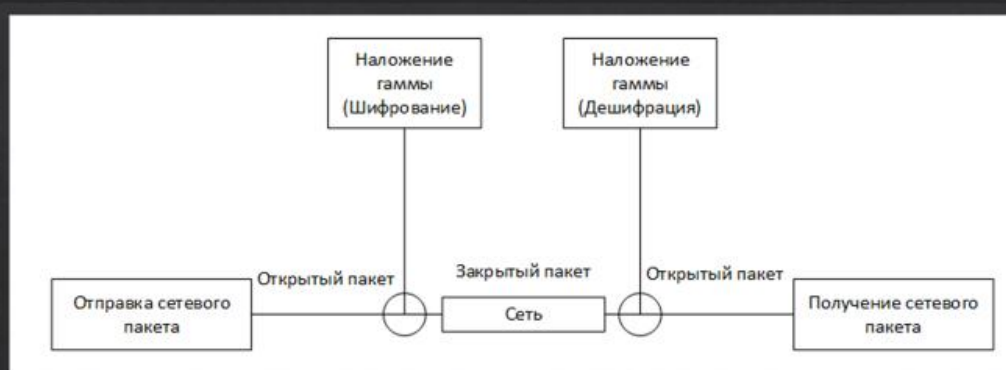
- Ethernet заголовок – в этом заголовке ведётся проверка на следующий заголовок сетевого уровня, если это IPv4 то анализ продолжается;
- IPv4 заголовок – из этого заголовка выделяется IP-адрес, для поиска записи с криптографическим ключом шифрования и проверяется следующий заголовок транспортного уровня, если ключ шифрования найден и заголовок транспортного уровня TCP или UDP то шифрование продолжается.
- Заголовки транспортного уровня TCP или UDP – из этих заголовков рассчитывает только их размер.

Для получения смещения до начала полезной нагрузки, которую драйвер будет шифровать необходимо выделить из всех заголовков их размеры. Если сетевой пакет подходит под все вышеописанные условия драйвер переходит к его шифрованию.

Шифрование полезной нагрузки

После получения указателя на начала полезной нагрузки сетевого пакета начинается процесс шифрования методом гаммирования. Данный метод шифрования является потоковым, что позволяет легко использовать MDL сетевых пакетов для шифрования.

Метод гаммирования — это метод шифрования, при котором используется последовательность случайных символов (гамма), которая комбинируется с исходным текстом с использованием операции *XOR*. Гаммой в данном случае является ключ шифрования. Для расшифрования данных необходимо на ранее зашифрованный текст наложить ту же гамму.



Общая схема работы с зашифрованными данными

Сравнение скорости работы

Скорость передачи данных без шифрования

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49683 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  533 MBytes  4.47 Gbits/sec
[ 4] 1.00-2.00 sec  573 MBytes  4.81 Gbits/sec
[ 4] 2.00-3.00 sec  575 MBytes  4.82 Gbits/sec
[ 4] 3.00-4.00 sec  578 MBytes  4.85 Gbits/sec
[ 4] 4.00-5.00 sec  582 MBytes  4.88 Gbits/sec
[ 4] 5.00-6.00 sec  575 MBytes  4.82 Gbits/sec
[ 4] 6.00-7.00 sec  586 MBytes  4.92 Gbits/sec
[ 4] 7.00-8.00 sec  560 MBytes  4.69 Gbits/sec
[ 4] 8.00-9.00 sec  584 MBytes  4.90 Gbits/sec
[ 4] 9.00-10.00 sec 574 MBytes  4.82 Gbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec 5.59 GBytes  4.80 Gbits/sec
[ 4] 0.00-10.00 sec 5.59 GBytes  4.80 Gbits/sec

sender
receiver
```

Скорость передачи данных с шифрованием

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49687 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  276 MBytes  2.30 Gbits/sec
[ 4] 1.00-2.00 sec  281 MBytes  2.36 Gbits/sec
[ 4] 2.00-3.00 sec  286 MBytes  2.40 Gbits/sec
[ 4] 3.00-4.00 sec  279 MBytes  2.34 Gbits/sec
[ 4] 4.00-5.00 sec  287 MBytes  2.41 Gbits/sec
[ 4] 5.00-6.00 sec  283 MBytes  2.38 Gbits/sec
[ 4] 6.00-7.00 sec  284 MBytes  2.38 Gbits/sec
[ 4] 7.00-8.00 sec  285 MBytes  2.39 Gbits/sec
[ 4] 8.00-9.00 sec  286 MBytes  2.40 Gbits/sec
[ 4] 9.00-10.00 sec 277 MBytes  2.32 Gbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec 2.76 GBytes  2.37 Gbits/sec
[ 4] 0.00-10.00 sec 2.76 GBytes  2.37 Gbits/sec

sender
receiver
```

Сравнение скорости работы

Скорость передачи данных без шифрования

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49787 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  22.2 MBytes  187 Mbits/sec
[ 4] 1.00-2.00 sec  24.1 MBytes  202 Mbits/sec
[ 4] 2.00-3.02 sec  23.1 MBytes  191 Mbits/sec
[ 4] 3.02-4.02 sec  23.9 MBytes  200 Mbits/sec
[ 4] 4.02-5.00 sec  24.5 MBytes  209 Mbits/sec
[ 4] 5.00-6.02 sec  23.2 MBytes  192 Mbits/sec
[ 4] 6.02-7.00 sec  24.2 MBytes  207 Mbits/sec
[ 4] 7.00-8.02 sec  23.4 MBytes  193 Mbits/sec
[ 4] 8.02-9.02 sec  24.9 MBytes  209 Mbits/sec
[ 4] 9.02-10.00 sec 23.4 MBytes  199 Mbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec 237 MBytes  199 Mbits/sec
[ 4] 0.00-10.00 sec 237 MBytes  199 Mbits/sec

iperf Done.
```

Скорость передачи данных с шифрованием

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49739 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  22.0 MBytes  184 Mbits/sec
[ 4] 1.00-2.02 sec  23.6 MBytes  195 Mbits/sec
[ 4] 2.02-3.02 sec  24.5 MBytes  206 Mbits/sec
[ 4] 3.02-4.00 sec  23.4 MBytes  199 Mbits/sec
[ 4] 4.00-5.00 sec  23.9 MBytes  200 Mbits/sec
[ 4] 5.00-6.02 sec  23.6 MBytes  195 Mbits/sec
[ 4] 6.02-7.00 sec  24.5 MBytes  209 Mbits/sec
[ 4] 7.00-8.00 sec  23.6 MBytes  198 Mbits/sec
[ 4] 8.00-9.00 sec  23.6 MBytes  198 Mbits/sec
[ 4] 9.02-10.00 sec 23.4 MBytes  199 Mbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec 237 MBytes  199 Mbits/sec
[ 4] 0.00-10.00 sec 237 MBytes  199 Mbits/sec

iperf Done.
```

Результаты работы

- ❑ В результате выполнения выпускной квалификационной работы был разработан NDIS драйвер-фильтр для семейства операционных систем Windows, который выполняет шифрование входящего и исходящего трафика, создавая тем самым безопасные соединения, обеспечивающие безопасную передачу сетевых пакетов.
- ❑ Разработанный драйвер поддерживает шифрование сетевых пакетов с протоколом сетевого уровня IPv4 и заголовками TCP или UDP на транспортном уровне.
- ❑ Драйвер поддерживает функцию создания более чем одного безопасного соединения с шифрованием.
- ❑ Определение пакетов для шифрации или дешифрации происходит по IPv4 адресу.
- ❑ Для шифрования сетевых пакетов использовался потоковый метод гаммирования, который обеспечивал высокую скорость шифрования и дешифрации, тем самым не сильно нагружая систему и замедляя передачу данных.
- ❑ Разработано приложения с графическим интерфейсом для управления драйвером.

Спасибо за внимание

					ПГУ 1.090301.37.001 ПЗ	Лист
						118
Изм.	Лист	№ докум.	Подпи	Дата		