## **Описание решения (Solution) NetworkMonitor**

На данный (Solution) NetworkMonitor состит из 4 проектов:

1. NetworkMonitor.Common – Сборка содержит в себе базовые вещи которые используются во всех других сборках (рисунок 1):

* Constants - константы такие как сообщения оба ошибках, сообщения для логирования, значения по умолчанию.
* Dto – (**Data Transfer Object**) - данные для передачи от клиента серверу.
* Enums - содержит перечисления.
* Exceptions – содержит исключения, которые могут возникать в ходе работы.
* ExtensionMethods – содержит методы расширения для базовый классов .net, например для NetworkInterface.
* Interfaces – содержит описание контрактов, которые необходимо реализовать для каждой платформы (Windows, Linux).
* Settings - содержит класса с настрйоками.

1. NetworkMonitor.Implementation – Сборка содержит в себе реализации интерфейсов для различный платформ.
2. NetworkMonitor.Tests – Сборка содержит в себе Unit тесты для проверки корректной работы различный модулей. Папка Builders – содержи в себе реализации Gof паттерна Builder для создания сущностей, которые будут использовать в Unit тестах.
3. NetworkMonitor.WindowsService – Сборка содержит в себе клиент для системы Windows, которая может быть установлена в систему в качестве системной службы (Windows Services).

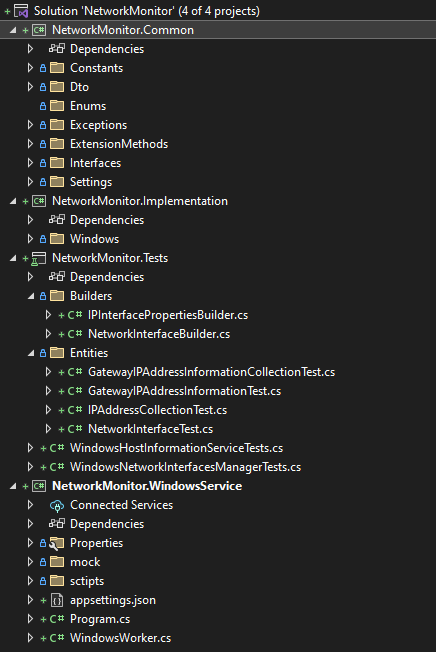


Рисунок 1 – Схема решения (Solution)

## **Описание интерфейсов и Dto.**

1. IHostInformationService – Интерес содержит в себе контракт на получения информации о узле сети (рисунок 2):
   * GetDhcp() - получение IP адреса DHCP Сервера.
   * GetGateway() - получение IP адреса шлюза по-умолчанию.
   * GetHostName() - получение имени машины.
   * GetPv4Address() - получение IP адреса машины.
   * GetDnsList() - получение списка IP адресов DNS.
   * GetTracertTable() - получение таблицы маршрутизации.
   * GetArpTable() - получение ARP таблицы содержащий коллекцию Dto Host(рисунок 3).
   * GetHostInformation() - Получение информации об узле сети в виде Dto HostInformation(рисунок 4).

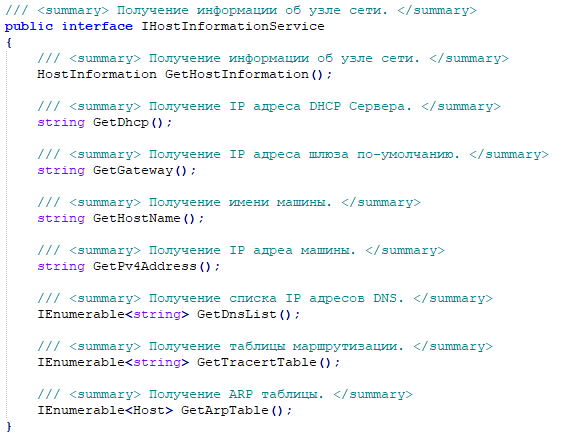


Рисунок 2 – Интерфейс IHostInformationService

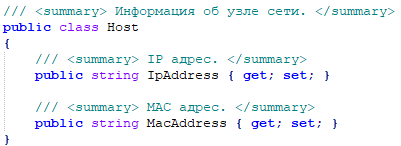


Рисунок 3 – Dto Host

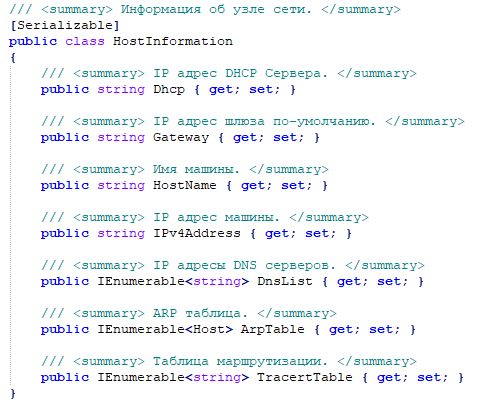


Рисунок 4 – Dto HostInformation

1. IWindowsManager – Интерес содержит в себе контракт для работы со специфическими вещями для ОС Windows такие как работа с протоколами ARP и ICMP (рисунок 5):
   * GetArpTable() - получение ARP таблицы содержащий коллекцию Dto Host(рисунок 3)
   * GetTracertTable() - получение таблицы маршрутизации до шлюза по умолчанию.

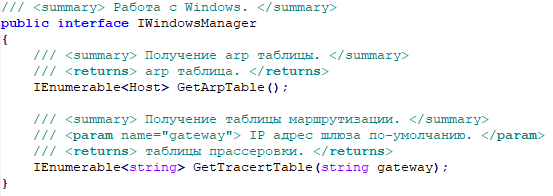


Рисунок 5 – Интерфейс IWindowsManager

1. IHttpClient – Интерес содержит в себе контракт для отправки, данный на сервер посредствам протокола HTTP (рисунок 6). Содержит в себе сигнатуру метода SendHostInformation для отправики Dto HostInformation с информацией о текущем узле сети на сервер для дальнейшей валидации.

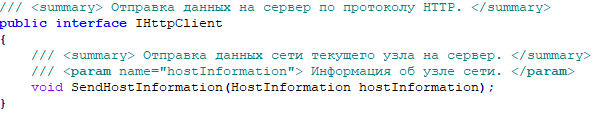


Рисунок 6 – Интерфейс IWindowsManager

## **Описание интерфейсов для Windows.**

WindowsHttpClientMock – класс реализующий интерфейс IHttpClient (рисунок 7), так как сервер еще не реализован, класс представляет собой заглушку для тестирования работоспособности сервиса. Класс WindowsHttpClientMock содержит реализация метода SendHostInformation (рисунок 6), данный метод стерилизует экземпляр класса HostInformation в формат json и сохраняет его в файл с имением текущей даты и времени.

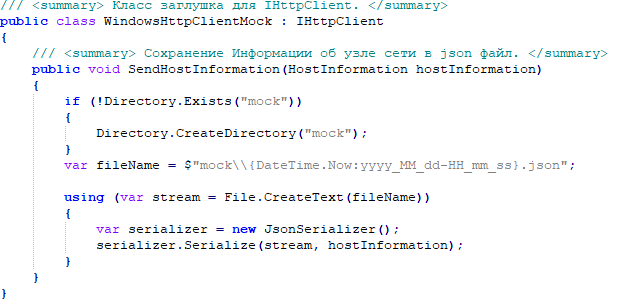


Рисунок 7 – Класс WindowsHttpClientMock

WindowsCmdManager – класс реализующий интерфейс IWindowsManager (рисунок 8). WindowsCmdManager – пример реализации интрефейса IWindowsManager с помощь стандартный утилит windows, а именно arp.exe и tracert.exe. В методах GetTracertTable и GetArpTable происходит вызов этих утилит с перенаправлением потока вывода на методы ParseTracert и ParseArp и дальнейшего парсинга результата работы утилит arp.exe и tracert.exe (рисунок 9).



Рисунок 8 – Класс WindowsCmdManager

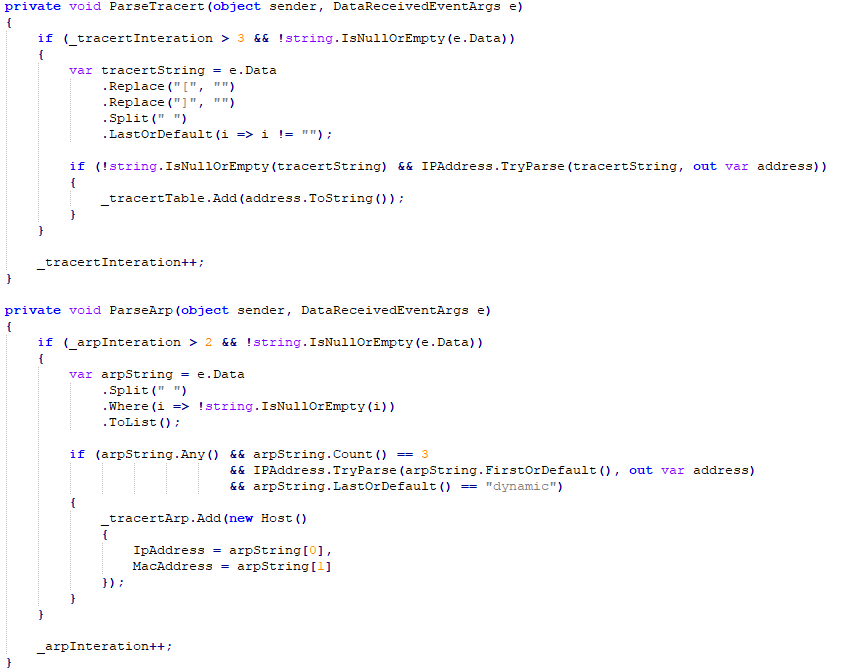


Рисунок 9 – Класс WindowsCmdManager

WindowsHostInformationService – класс реализующий интерфейс IHostInformationService (рисунок 10). Класс WindowsHostInformationService содержит в себе методы получающие информацию об узле сети. В классе WindowsHostInformationService есть две приватный переменный доступных только для чтения \_ipInterfaceProperties типа IPInterfaceProperties и \_manager типа IWindowsManager. В конструкторе с помощью такой техники как DI (Dependency injection) происходит внедрение зависимостей, делается это для большей гибкости и для удобства написания Unit тестов, так как можно указать любую реализацию реализующий интерфейс. Так же класс WindowsHostInformationService сдержит реализацию всех методов интерфейса IHostInformationService. Методы GetTracertTable и GetArpTable внутри себя вызывают методы из перемены \_manager, где может быть любой экземпляр класса реализующий интерфейс IWindowsManager.

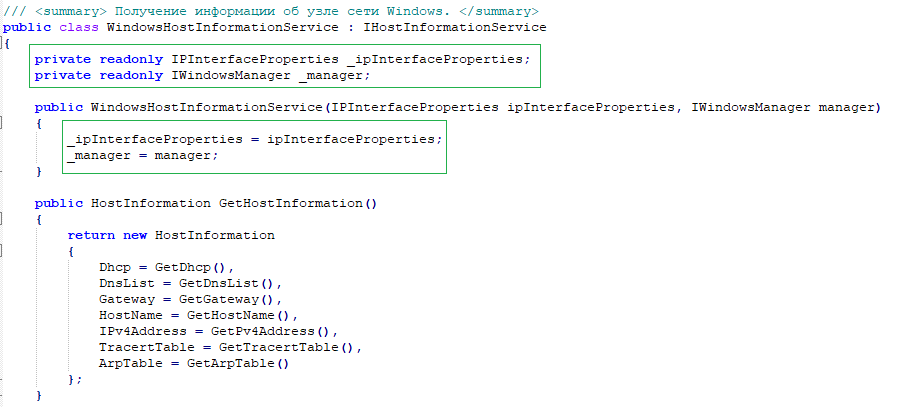


Рисунок 10 – Класс WindowsHostInformationService

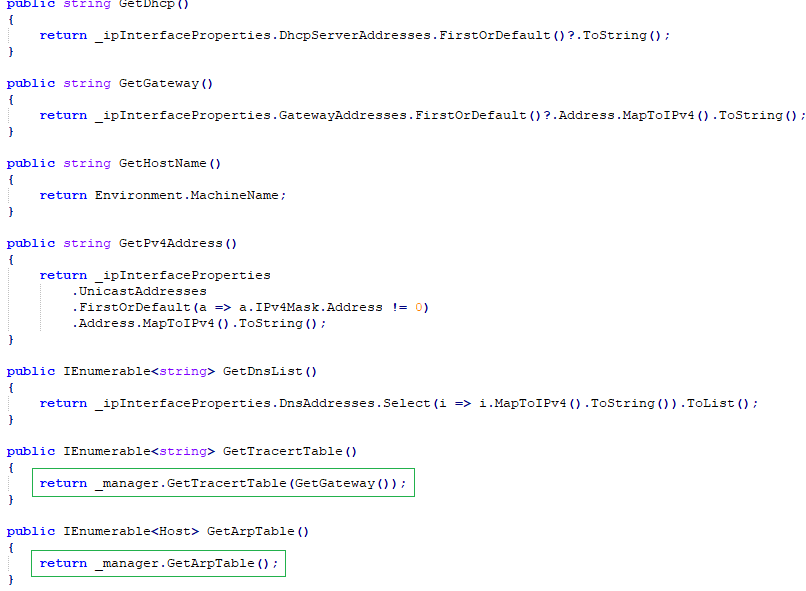


Рисунок 11 – Класс WindowsHostInformationService

## **Описание NetworkMonitor.WindowsService**

NetworkMonitor.WindowsService – сборка для создания клиента в виде службы window. Для этого необходимо добавить к проекту следующие Nuget пакеты Microsoft.Extensions.Hosting, Microsoft.Extensions.Hosting.WindowsServices и ссылка на проекты NetworkMonitor.Common и NetworkMonitor.Implementation. На рисунке 12 изображен кода из файла Program.cs в котором происходит конфигурация сервиса. Получение параметров из файла appsettings.json. И внедрение зависимостей в IoC контейнер. И также добавление WindowsWorker в котором будет выполнятся все логика. Рисунок 13. На рисунке 13 показан код работы метода ExecuteAsync в котором происходит получение настроек узла и отправку их на сервер (пока с помощью заглушки пишется в json файл). Затем идет задержка, значение которой берется из файла конфигурации и итерация повторяется. Для создания сервиса Windows необходимо скомпилировать сборку NetworkMonitor.WindowsService и полнить bat скрипт правами администратора указав в нем путь до NetworkMonitor.WindowsService.exe “create NetworkMonitor.WindowsService.bat” код которого представлен на рисунке 14. После чего можно запустить утилиту services.msc и убедиться, что создана новая служба, которую можно настроить и запускать или выключать. Помечанию службы windows используется папка C:\Windows\System32. Можно увидеть, что NetworkMonitor.WindowsService создал папку mock и сохраняет туда значение HostInformation рисунок 16, рисунок 17.



Рисунок 12 – Конфигурация сервиса NetworkMonitor.WindowsService



Рисунок 13 – Конфигурация сервиса NetworkMonitor.WindowsService



Рисунок 14 – Скрипт create NetworkMonitor.WindowsService.bat

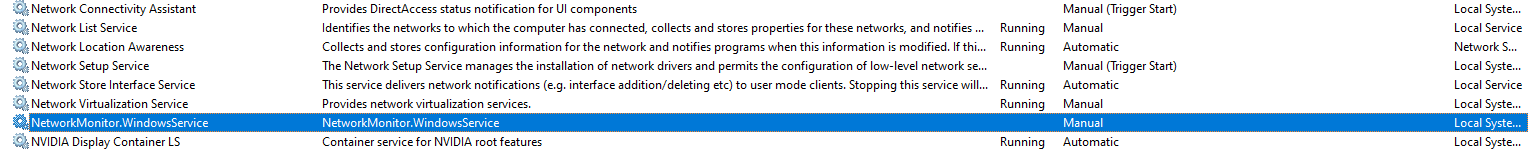


Рисунок 15 – Скрипт services.msc

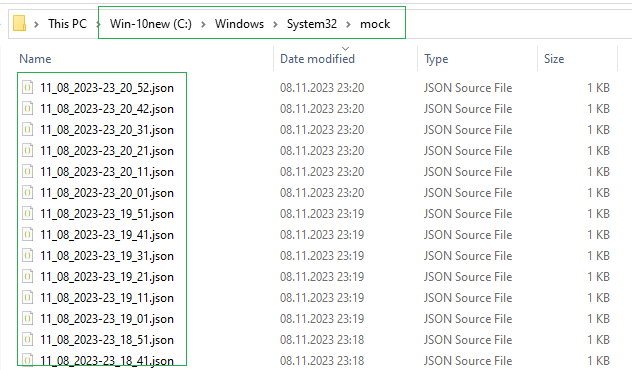


Рисунок 16 – Папка mock.



Рисунок 17 – Пример json файла.

**ЗАКЛЮЧЕНИЕ**

Основная идея разработки написать сетевое клиент-серверное приложение, которое позволяло бы централизованное выявлять атаки MITM (“человек посередине») и оперативно реагировать на инциденты. В статье описывается написание первой части этого приложения, а именно клиента, который планируется устанавливать на каждый узел сети, для отправки информации о сетевых настройках, а также ARP таблицы и таблицы трассировки. Описана архитектура приложения клиента, а также показа процесс компиляции и запуска клиента в качестве системной службы windows.