# Report

**Contents**

[Report 4](#_Toc124326614)

[Introduction 7](#_Toc124326615)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc124326616)

[1.1 Обзор предметной области 8](#_Toc124326617)

[1.2 Постановка задачи 9](#_Toc124326618)

[1.3 Обзор существующих аналогов 10](#_Toc124326619)

[1.3.1 Zecurion DLP 10](#_Toc124326620)

[1.3.2 InfoWatch Traffic Monitor 11](#_Toc124326621)

[1.3.3 FalconGaze SecureTower 12](#_Toc124326622)

[1.3.4 Search Inform «Контур информационной безопасности» 13](#_Toc124326623)

[1.4 Сравнительный анализ 14](#_Toc124326624)

[1.5 Архитектура «клиент-сервер» 15](#_Toc124326625)

[1.6 Общеязыковая среда выполнения CLR 17](#_Toc124326626)

[1.7 Язык программирования C# 17](#_Toc124326627)

[1.8 Платформа .NET Framework 17](#_Toc124326628)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 19](#_Toc124326629)

[2.1 Сервер администрирования 19](#_Toc124326630)

[2.2 Клиентская служба 21](#_Toc124326631)

[2.3 Динамически подключаемая библиотека 23](#_Toc124326632)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 24](#_Toc124326633)

[3.1 Сервер администрирования 24](#_Toc124326634)

[3.1.1 Блок хранения и анализа модулей контроля каналов утечки информации 24](#_Toc124326635)

[3.1.2 Блок работы сервера 26](#_Toc124326636)

[3.1.3 Блок непосредственной передачи данных между клиентской службой и сервером 27](#_Toc124326637)

[3.1.4 Блок взаимодействия с интерфейсом пользователя 30](#_Toc124326638)

[3.2 Клиентская служба 35](#_Toc124326639)

[3.2.1 Блок управления модулями контроля каналов утечки информации 36](#_Toc124326640)

[3.2.2 Блок взаимодействия с сервером 38](#_Toc124326641)

[3.2.3 Блок клиентской службы 39](#_Toc124326642)

[3.2.4 Блок управления службой 40](#_Toc124326643)

[3.2.5 Блок конфигурирования службы 41](#_Toc124326644)

[3.3 Динамически подключаемая библиотека 42](#_Toc124326645)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 48](#_Toc124326646)

[4.1 Контроль интегрируемых модулей 48](#_Toc124326647)

[4.1.1 Считывание и анализированние информации о модуле 48](#_Toc124326648)

[4.1.2 Сохранение модуля 49](#_Toc124326649)

[4.1.3 Удаление модуля с сервера 51](#_Toc124326650)

[4.2 Контроль клиентов 51](#_Toc124326651)

[4.2.1 Идентификация клиентов 51](#_Toc124326652)

[4.2.2 Установка и удаление модулей на клиентской машине 55](#_Toc124326653)

[4.2.3 Конфигурирование модуля и получение данных модуля 57](#_Toc124326654)

[4.2.4 Отправка запросов и получение ответов на запросы 60](#_Toc124326655)

[4.3 Обработка входящих запросов 61](#_Toc124326656)

[4.3.1 Обработка запросов со стороны сервера 61](#_Toc124326657)

[4.3.2 Обработка запросов со стороны клиента 64](#_Toc124326658)

[4.4 Передача данных по сети 66](#_Toc124326659)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 69](#_Toc124326660)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 77](#_Toc124326661)

[6.1 Требования к программному и аппаратному обеспечению 77](#_Toc124326662)

[6.2 Руководство по установке клиентской службы 77](#_Toc124326663)

[6.3 Руководство по установке сервера 79](#_Toc124326664)

[6.4 Руководство по управлению сервером 80](#_Toc124326665)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ МОДУЛЯ МЕЖСЕТЕВОГО ВЗАИМОДЕЙСТВИЯ DLP-СИСТЕМЫ 89](#_Toc124326666)

[7.1 Характеристика разработанного программного средства 89](#_Toc124326667)

[7.2. Расчет инвестиций в разработку программного средства 89](#_Toc124326668)

[7.3 Расчет результата от разработки и использования программного средства 91](#_Toc124326669)

[7.3.1 Экономический эффект от разработки программного средства для организации-разработчика 91](#_Toc124326670)

[7.3.2 Экономический эффект от использования программного средства для организации-заказчика 91](#_Toc124326671)

[7.4 Расчет показателей экономической эффективности разработки и использования программного средства 92](#_Toc124326672)

[7.4.1 Оценка экономической эффективности для организации-разработчика 93](#_Toc124326673)

[7.4.2 Расчет экономической эффективности для организации-заказчика 93](#_Toc124326674)

[ЗАКЛЮЧЕНИЕ 95](#_Toc124326675)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 96](#_Toc124326676)

[ПРИЛОЖЕНИЕ А 97](#_Toc124326677)

[ПРИЛОЖЕНИЕ Б 98](#_Toc124326678)

[ПРИЛОЖЕНИЕ В 99](#_Toc124326679)

# Introduction

# 1 Subject area research

1.1 Studying the concept of web scraping [[1](https://en.wikipedia.org/wiki/Web_scraping)]

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. Web scraping software may directly access the World Wide Web using the Hypertext Transfer Protocol or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, extraction can take place. The content of a page may be parsed, searched and reformatted, and its data copied into a spreadsheet or loaded into a database. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be finding and copying names and telephone numbers, companies and their URLs, or e-mail addresses to a list (contact scraping).

As well as contact scraping, web scraping is used as a component of applications used for web indexing, web mining and data mining, online price change monitoring and price comparison, product review scraping (to watch the competition), gathering real estate listings, weather data monitoring, website change detection, research, tracking online presence and reputation, web mashup, and web data integration.

Web pages are built using text-based mark-up languages (HTML and XHTML), and frequently contain a wealth of useful data in text form. However, most web pages are designed for human end-users and not for ease of automated use. As a result, specialized tools and software have been developed to facilitate the scraping of web pages.

Newer forms of web scraping involve monitoring data feeds from web servers. For example, JSON is commonly used as a transport storage mechanism between the client and the web server.

There are methods that some websites use to prevent web scraping, such as detecting and disallowing bots from crawling (viewing) their pages. In response, there are web scraping systems that rely on using techniques in DOM parsing, computer vision and natural language processing to simulate human browsing to enable gathering web page content for offline parsing.

## 1.2 Web scraping techniques [[2](https://en.wikipedia.org/wiki/Web_scraping)]

**Human copy-and-paste**

The simplest form of web scraping is manually copying and pasting data from a web page into a text file or spreadsheet. Sometimes even the best web-scraping technology cannot replace a human's manual examination and copy-and-paste, and sometimes this may be the only workable solution when the websites for scraping explicitly set up barriers to prevent machine automation.

**Text pattern matching**

A simple yet powerful approach to extract information from web pages can be based on the UNIX grep command or regular expression-matching facilities of programming languages.

**HTTP programming**

Static and dynamic web pages can be retrieved by posting HTTP requests to the remote web server using socket programming.

**HTML parsing**

Many websites have large collections of pages generated dynamically from an underlying structured source like a database. Data of the same category are typically encoded into similar pages by a common script or template. In data mining, a program that detects such templates in a particular information source, extracts its content and translates it into a relational form, is called a wrapper. Wrapper generation algorithms assume that input pages of a wrapper induction system conform to a common template and that they can be easily identified in terms of a URL common scheme. Moreover, some semi-structured data query languages, such as XQuery and the HTQL, can be used to parse HTML pages and to retrieve and transform page content.

**DOM parsing**

By embedding a full-fledged web browser, such as the Internet Explorer or the Mozilla browser control, programs can retrieve the dynamic content generated by client-side scripts. These browser controls also parse web pages into a DOM tree, based on which programs can retrieve parts of the pages. Languages such as Xpath can be used to parse the resulting DOM tree.

**Vertical aggregation**

There are several companies that have developed vertical specific harvesting platforms. These platforms create and monitor a multitude of "bots" for specific verticals with no "man in the loop" (no direct human involvement), and no work related to a specific target site. The preparation involves establishing the knowledge base for the entire vertical and then the platform creates the bots automatically. The platform's robustness is measured by the quality of the information it retrieves (usually number of fields) and its scalability (how quick it can scale up to hundreds or thousands of sites). This scalability is mostly used to target the Long Tail of sites that common aggregators find complicated or too labor-intensive to harvest content from.

**Semantic annotation recognizing**

The pages being scraped may embrace metadata or semantic markups and annotations, which can be used to locate specific data snippets. If the annotations are embedded in the pages, as Microformat does, this technique can be viewed as a special case of DOM parsing. In another case, the annotations, organized into a semantic layer, are stored and managed separately from the web pages, so the scrapers can retrieve data schema and instructions from this layer before scraping the pages.

**Computer vision web-page analysis**

There are efforts using machine learning and computer vision that attempt to identify and extract information from web pages by interpreting pages visually as a human being might.

## 1.3 Methods to prevent web scraping [[3](https://en.wikipedia.org/wiki/Web_scraping)]

The administrator of a website can use various measures to stop or slow a bot. Some techniques include:

1. Blocking an IP address either manually or based on criteria such as geolocation and DNSRBL. This will also block all browsing from that address.
2. Disabling any web service API that the website's system might expose.
3. Bots sometimes declare who they are (using user agent strings) and can be blocked on that basis using robots.txt; 'googlebot' is an example. Other bots make no distinction between themselves and a human using a browser.
4. Bots can be blocked by monitoring excess traffic
5. Bots can sometimes be blocked with tools to verify that it is a real person accessing the site, like a CAPTCHA. Bots are sometimes coded to explicitly break specific CAPTCHA patterns or may employ third-party services that utilize human labor to read and respond in real-time to CAPTCHA challenges.
6. Commercial anti-bot services: Companies offer anti-bot and anti-scraping services for websites. A few web application firewalls have limited bot detection capabilities as well. However, many such solutions are not very effective.
7. Locating bots with a honeypot or other method to identify the IP addresses of automated crawlers.
8. Obfuscation using CSS sprites to display such data as telephone numbers or email addresses, at the cost of accessibility to screen reader users.
9. Because bots rely on consistency in the front-end code of a target website, adding small variations to the HTML/CSS surrounding important data and navigation elements would require more human involvement in the initial set up of a bot and if done effectively may render the target website too difficult to scrape due to the diminished ability to automate the scraping process.
10. Websites can declare if crawling is allowed or not in the robots.txt file and allow partial access, limit the crawl rate, specify the optimal time to crawl and more.
11. Load database data straight into the HTML DOM via AJAX, and use DOM methods to display it, forcing crawlers to either reproduce those AJAX requests or use browser rendering (e.g. a headless browser).

## 1.4 Disclaimer

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После изучения теоретических аспектов и рассмотрения аналогов, оптимальным решением для разработки модуля межсетевого взаимодействия является разбиение его на 3 части: сервер администрирования, клиентская служба и динамически подключаемой библиотекой. Сервер администрирования и клиентская служба взаимодействуют посредством протокола передачи данных TCP.

Сервер администрирования представляет собой модуль программного обеспечения с графическим интерфейсом, осуществляющий контроль и хранение используемых модулей. конфигурирование и анализ работы клиентов, а также управление исполнением установленных на клиентские машины модули контроля каналов утечки информации.

Вторая часть будет реализована в виде клиентской службы, которая взаимодействует с сервером администрирования и следит за исполнением и конфигурированием модулей контроля каналов утечки информации и отправляет на сервер сведения о работе данных модулей. Взаимодействие между клиентскими службами и сервером администрирования будет осуществляться на основе универсального протокола, специально разработанного для задач интегрирования новых модулей в DLP-систему и взаимодействия элементов системы. Передача данных между сервером и клиентскими службами будет осуществляется на основе транспортного протокола TCP, а передаваемые данные между модулями будут представлены в виде JSON объектов.

Третья часть системы представлена в виде набора общей функциональности для сервера администрирования и клиентской службы собранной в динамически подключаемую библиотеку. Использование данного типа программного обеспечения позволяет экономить оперативную память машины используемую системой, а также ускоряет разработку и отладку программного кода.

Использование универсального протокола взаимодействия позволяет организовать модульную архитектуру системы, а представление клиента в виде службы Windows позволяет устранить работу администратора службы безопасности, заключающуюся в контроле за запуском и завершением модулей.

Структурная схема, иллюстрирующая все перечисленные далее блоки и связи между ними, приведена на чертеже ГУИР.400201.017 C1.

Далее более подробно рассматриваются функциональные блоки каждого из компонентов системы.

## 2.1 Сервер администрирования

В сервере администрирования можно выделить следующие блоки:

* блок хранения и анализа модулей контроля каналов утечки информации;
* блок работы сервера;
* блок непосредственной передачи данных между клиентской службой и сервером;
* блок взаимодействия с интерфейсом пользователя;
* блок контроля и взаимодействия с клиентской службой.

Рассмотрим функциональные блоки сервера.

*Блок хранения и анализа модулей контроля каналов утечки информации* анализирует переданные ему модули контроля каналов утечки информации, путем чтения конфигурационных файлов. Далее регистрирует модуль в своих внутренних файлах конфигурации и формирует конечный модуль, готовый для инсталяции на клиентскую машину. При необходимости установки конкретного модуля контроля канала утечки информации возвращает пакет сформированный из модуля и файлов конфигурации.

*Блок работы сервера* является главным блоком сервера, в котором находится основные функции запуска и остановки сервера. Данные блок использует новые возможности многопоточного исполнения кода от .NET Framework – объединение потоков в пул, через библиотеку TPL (Task Parallel Library). Данная библиотека исполняет код в представлении задач. При назначении задачи менеджер пула "инжектирует" новые потоки, чтобы справиться с дополнительной параллельно работающей полезной нагрузкой, пока не будет достигнут определенный максимальный предел. После значительного периода отсутствия активности менеджер пула может "удалять" потоки, если есть подозрение, что это приведет к улучшению производительности.

Использование многопоточной архитектуры позволяет распределить вычислительную нагрузку от модулей на ядра процессора, что благоприятно влияет на скорость исполнения системы.

*Блок непосредственной передачи данных между клиентской службой и сервером* представляет собой набор функций, вынесенных в отдельный функциональный блок, представленный сборкой DLL (Dynamic Link Library). Библиотека инкапсулирует функциональность данного блока, что обеспечивает возможность добавления функциональности не изменяя общей архитектуры системы. Общение сервера и клиента осуществляется посредством протокола TCP. Данный протокол является ориентированным на соединение и гарантирует, что все отправленные данные дойдут до получателя в том же порядке, что и были отправлены. Что не может гарантировать датаграмнный протокол UDP. Но протокол TCP не хранит состояние соединения и передает данные потоком, которые сложно разделить на логические сообщения. Для этих целей предназначен этот блок, который контролирует передачу данных между хостами, делит данные на логически понятные сообщения и обрабатывает исключения возникающие при работе в сети. А сами данные передаются в виде объектов сериализованных в формат JSON.

*Блок взаимодействия с интерфейсом пользователя* данный блок представляет собой набор API (Application programming interface) для взаимодействия с интегрируемым интерфейсом пользователя. Через данный набор API возможно управление системой, получение данных, хранящихся в системе, а также ее конфигурирование. Интерфейс пользователя, реализующий данный набор API получает доступ к контролю над системой. Наличие универсального программного интерфейса взаимодействия с системой не ограничивает возможность реализации любого типа интерфейса: web-приложение, декстопное или мобильное. В целях безопасности данный набор программного интерфейса взаимодействия скомпилирован в закрытую сборку.

Сам API, передача данных и взаимодействие между сервером и интегрированным интерфейсом пользователя происходит за счет использования межпроцессного взаимодействия IPC (Inter-process communication). Операционная система Windows и платформа .NET Framework предоставляет огромное количество способов межпроцессного взаимодействия, такие как именованные каналы, разделяемая память или IPC каналы. Эти способы межпроцессного взаимодействия основываются на использовании оперативной памяти, что приводит к высокой скорости отклика пользовательского интерфейса, которая будет очень близка к той, если бы интерфейс пользователя был интегрирован непосредственно в модуль.

*Блок контроля и взаимодействия с клиентской службой* в данном блоке реализована возможность удалённого контроля клиентов, в том числе удаленная установка на клиентскую ЭВМ различных модулей для защиты тех или иных каналов утечки информации, которые необходимы администратору, а также рассмотрена функция верификации конфигураций клиентской службы.

.NET Framework поддерживает технологию рефлексии. Эта технология позволяет во временя выполнения приложения определить тип объекта. Данный блок реализует расширяемый протокол взаимодействия сервера и клиентских служб на основе рефлексии. Благодаря данному протоколу существует возможность интегрирования модулей контроля каналов утечки информации в непосредственно работающую систему.

## 2.2 Клиентская служба

В клиентской службе можно выделить следующие блоки:

* блок непосредственной передачи данных между клиентской службой и сервером;
* блок управления модулями контроля каналов утечки информации;
* блок взаимодействия с сервером;
* блок клиентской службы;
* блок управления службой;
* блок конфигурирования службы.

Блок непосредственной передачи данных между клиентской службой и сервером описан при рассмотрении функциональных блоков сервера администрирования.

Более подробно рассмотрим остальные функциональные блоки клиентской службы.

*Блок управления модулями контроля каналов утечки информации* данный блок отвечает за корректное исполнение, установку, удаление, конфигурирование и удаление модулей контроля каналов утечки информации.

*Блок взаимодействия с сервером* данный блок представляет собой набор функциональности отвечающий за парсинг различных команд от сервера, таких как установка новых модулей, конфигурация, остановка и удаление уже существующих модулей. Данный модуль реализует расширяемый протокол взаимодействия клиентской службы и сервера.

*Блок клиентской службы* представляет собой главную программу службы иотвечает за реализацию функциональности службы. Программа имеет несколько важных функций, необходимых для работы службы: главная функция, главная служебная функция и обработчика. Главная функция является точкой входа в программу. Она может содержать в себе одну или более зарегистрированную главную служебную функцию. В главной функции с помощью Диспетчера управления службами должны быть зарегистрированы все точки входа для её главных служебных функций. Главная служебная функция содержит основную функциональность службы. Так как программа службы может предоставлять несколько служб, то каждой из этих служб должна быть предоставлена главная служебная функция программы. Также одной из важнейших задач главной служебной функции является регистрация обработчика в Диспетчере управления службами. Основная задача обработчика заключается в том, чтобы он умел отвечать на события Диспетчера, такие как запуск, остановка, приостановка и возобновление работы службы.

*Блок управления службой* позволяет управлять службой. Он также реализован в виде программы, которая после регистрации обработчика может отправлять Диспетчеру управления службами запросы на завершение, возобновление или приостановку службы. Она также позволяет запрашивать текущее состояние службы и реализовывать специальный код обработки.

*Блок конфигурирования службы* предназначен для установки и настройки службы и представляет собой программу конфигурирования. Компоненты .NET могут устанавливаться с помощью простой команды xcopy, однако, установка служб требует соответствующей настройки в системном реестре, чего данная команда не реализует. В системном реестре содержится параметр, который отвечает за запуск службы: то ли это автоматический запуск, то ли запуск вручную. Программа конфигурирования также производит настройку пользователя службы и зависимостей службы, то есть тех служб, которые должны запускаться перед запуском данной службы. Использование данной программы возможно как в программе установки для первоначальной настройки службы, так и вне программы установки для изменений конфигураций службы.

## 2.3 Динамически подключаемая библиотека

Данная часть системы представлена в виде динамически подключаемой библиотеки DLL (Dynamic Link Library). Динамически подключаемая библиотека позволяет многократное использование различными программными компонентами. Данная технология экономит оперативную память ЭВМ, благодаря размещению одного общего экземпляра в оперативной памяти, которым могут пользоваться внешние модули.

В данной библиотеке вынесен общий функционал для всей системы, такой как соединение по каналам передачи данных, обработка файлов, обработка модулей контроля каналов утечки информации, типы данных, которыми работает система.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Для более подробного рассмотрения функционирования модуля следует провести анализ всех классов, входящих в состав исходного кода модуля. Так как при проектировании модуля для создания алгоритмов был использован .NET Framework, то большое количество классов предоставляются самим фреймворком. Поэтому следует останавливаться лишь на самых основных частях модуля, представленных либо рядом классов, либо набором функций, следующих объектно-ориентированной парадигме программирования.

Классы, представленные в исходном коде модулей, представлены в

диаграммах классов: для сервера администрирования (см. чертеж ГУИР.400201.017 РР.1) и клиентской службы (см. чертеж ГУИР.400201.017 РР.2).

Последовательность работы блоков системы представлена в диаграмме последовательности (см. чертеж ГУИР.400201.017 РР.3).

## 3.1 Сервер администрирования

Как отмечалось ранее, сервер администрирования состоит из 5 логических блоков: блок хранения и анализа модулей контроля каналов утечки информации, блок работы сервера, блок непосредственной передачи данных между клиентской службой и сервером, блок взаимодействия с интегрируемым интерфейсом пользователя, блок контроля и взаимодействия с клиентской службой. Графический интерфейс пользователя реализован на основе технологии Microsoft – WPF (Windows Presentation Foundation).

Теперь стоит подробнее рассмотреть каждый из перечисленных блоков.

### 3.1.1 Блок хранения и анализа модулей контроля каналов утечки информации

Для реализации функциональности данного блока выбрана объектно-ориентированная парадигма программирования, так как она позволяет описывать объекты окружающего мира на основе их свойств и выполняемых действий.

Данный блок представлен классом ModulesAnalyzer, который выполняет функции парсинга атрибутов и свойств модуля, сохранения его на сервере в указанную директорию и дальнейшее управление модулем. Добавленный модуль описывается классом DescriptionModule и в конечном итоге представляет объект этого класса, свойства которого описаны в таблице 3.1.

Данный блок при невозможности чтения атрибутов и свойств модуля, либо при несовместимости модуля с системой возвращает ошибку и оповещает администратора о возникших проблемах.

Таблица 3.1 – Свойства класса DescriptionModule

|  |  |  |
| --- | --- | --- |
| Тип данных, хранящихся в свойстве | Свойство | Описание |
| string | FullName | Полное имя модуля. |
| string | FileVersion | Версия сборки модуля. |
| string | Path | Путь к исполняемому файлу модуля. |
| string | Title | Имя модуля. |
| string | Description | Хранит описание модуля. |
| string | Company | Имя компании, разрабатывающей модуль. |
| string | Copyright | Авторское право на разработанное программное обеспечение. |
| string | Product | Тип программного обеспечения, представленное модулем. |
| string | Trademark | Товарный знак на разработанное программное обеспечение. |
| string | Configuration | Описание конфигурации модуля. |
| string | TargetFramework | Тип и версия фреймворка, необходимого для корректной работы модуля. |

При добавлении сборки модуля контроля каналов утечки информации со всеми необходимыми зависимостями, манифестами сборки и файлами конфигурации, класс ModulesAnalyzer формирует файл конфигурации модуля, представленный типом данных DescriptionModule. После создания и заполнения файла конфигурации, модуль контроля каналов утечки информации архивируется в один пакет, готовый для передачи и установки на клиентскую ЭВМ. Функциональность класса ModulesAnalyzer представлена в таблице 3.2.

Таблица 3.2 – Методы класса ModulesAnalyzer

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| ModulesDescription() | Возвращает файл конфигурации модулей. Сканирует модули. |

Продолжение таблицы 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| StoreModule(vs) | Анализирует сборку модуля со всеми зависимостями, создает запись в файле конфигурации модулей, формирует пакет готовый к установке на клиентские машины и сохраняет этот пакет в директорию на сервере. |
| DeleteModule(module) | Удаляет запись о модуле из файла конфигурации и пакет из директории. |
| GetModule(module) | Возвращает пакет модуля готового для установки на клиентскую машину. |
| GetAssemblyAttribute(assembly) | Считывает атрибут сборки модуля и возвращает его вызывающему коду. |

### 3.1.2 Блок работы сервера

Для осуществления всех функциональных возможностей сервера администрирования реализована своеобразная API, основой которой является класс ServerEngine, который предоставляет функциональность для получения параметров методов, содержащихся в классах библиотеки. Реализация подобного API основана на механизмах интроспекции, представленных в стандартных библиотеках языка C#. Механизм интроспекции позволяет получать доступ к классам, методам и переменным непосредственно во время работы интерпретатора. Данный класс является своеобразным ядром сервера администрирования, через который происходит запуск, остановка, перезапуск сервера. Данный класс делегирует функциональность всех остальных классов системы и контролирует время жизни экземпляров классов. Централизация управления временем жизни экземпляров классов позволяет быстро уничтожать неиспользуемые экземпляры классов, что приводит к экономии оперативной памяти.

Оптимизация работы системы достигается за счет использования многопоточный среды исполнения кода. Класс ServerEngine использует функционал мощной библиотеки TPL (Task Parallel Library), которая инкапсулирует функционал работы с многопоточной средой исполнения кода.

При использовании многопоточной среды необходимы средства синхронизации потоков, а также необходимо организовать потокобезопасность. Для организации синхронизации и потокобезопасности используются события ManualResetEventSlim и критические секции. Данные типы синхронизации являются объектами ядра операционной системы, что предоставляет достоинства в виде ресурсоемкости и быстроты работы. ServerEngine хранит ссылки на объекты других блоков системы. Благодаря наличию объектов классов возможно управление их состоянием.

Соответственно данный блок является центральным исполнительным блоком сервера, функционал которого описан в таблице 3.3.

Таблица 3.3 – Методы класса ServerEngine

|  |  |
| --- | --- |
| Метод | Описание |
| Start() | Метод сервера, в которой происходит создание всех необходимых объектов для работы сервера, запуск исполняемых модулей и управление их работой. |
| Dispose() | Метод предназначен для завершения работы сервера администрирования. При ее вызове все события сигнализирующие о завершении работы сервера устанавливаются в сигнальное состояние, сохраняются все файлы конфигурации, уничтожаются используемые объекты, очищается выделенная оперативная память, а потоки завершают свою работу. |

### 3.1.3 Блок непосредственной передачи данных между клиентской службой и сервером

Блок непосредственной передачи данных использует функционал динамически подключаемой библиотеки DLPEngineLibary. Данный функционал представлен классом DataTransfer, методы данного класса описаны в таблице 3.4, а поля в таблице 3.5. При необходимости передачи данных между сервером и клиентом, данный модуль формирует пакет данных, представленным объектом класса Packet, свойства которого описаны в таблице 3.6, сериализует его в бинарный файл JSON – BSON (Binary JSON) и передает его слою, отвечающему за передачу данных.

При необходимости принять данные от удаленного хоста класс DataTransfer принимает массив байт, который десериалзуется в объект класса Packet.

Таблица 3.4 – Методы класса DataTransfer

|  |  |
| --- | --- |
| Метод | Описание |
| ListenerCallback(result) | Данный метод работает асинхронно относительно основного кода класса в отдельном потоке. При наличии входящего соединения данный метод принимает его и добавляет в очередь обработки входящих запросов. |
| SendTaskToHost(remoteIp, remotePort, packet, recieveTimeOutMillisec) | Отправляет сформированный пакет на удаленный хост, удаленный хост определяется парой IP-адрес и порт прослушивания. Также возможно изменять время отведенное на получение ответа от удаленного хоста. Возвращает ответ от хоста, сформированный в пакет. |
| DataTrasfer(localIp, localPort, addNewConnMethed) | Данный метод инициализирует необходимые поля, регистрирует сервер на паре порт, IP-адрес и начинает прослушивание входящих соединений. |

Таблица 3.5 – Поля класса DataTransfer

|  |  |
| --- | --- |
| Поле | Описание |
| 1 | 2 |
| localEndPointIp | IP-адрес локального хоста. |
| localEndPointPort | Порт локального хоста. |
| localHostName | Имя локального хоста. |
| defaultLocalEndPointIp | IP-адрес локального хоста по умолчанию. Используется в случаях, если администратор указал неверный IP-адрес либо произошла ошибка запуска сервера. |
| defaultLocalEndPointPort | Порт локального хоста по умолчанию. Используется в случаях, если администратор указал неверный порт либо произошла ошибка запуска сервера. |
| listener | Является объектом класса, который абстрагирует использование ресурсов операционной системы по доступу к сети. |

Продолжение таблицы 3.5

|  |  |
| --- | --- |
| 1 | 2 |
| AddNewConnection | Делегат, который вызывается при наличии нового соединения. Используется для добавления соединения в очередь обработки. |

Таблица 3.6 – Свойства класса Packet

|  |  |
| --- | --- |
| Свойство | Описание |
| GUID | Идентификатор хоста, который отправляет пакет. |
| HostName | Имя хоста, который отправляет пакет. |
| Task | Тип данных, который описывает задачу, а также хранит данные, необходимые для ее исполнения и поле для записи результата выполнения задачи. |

Как было сказано выше передаваемые данные формируются в объект класса Packet. Так как в данный классе не предустановлен тип данных, который он может в себе хранить, то его возможно использовать для передачи любых типов данных. Принимающая сторона в зависимости от типа задачи, указанной в пакете, расшифровывает данные под нужный тип. Тип задачи и передаваемые данные инкапсулирует объект класса TaskModel, свойства и методы, которого описаны в таблице 3.7 и 3.8 соответственно.

Таблица 3.7 – Свойства класса TaskModel

|  |  |
| --- | --- |
| Свойство | Описание |
| TaskType | Хранит тип задачи. |
| Request | Хранит данные, необходимые для исполнения задачи. |
| RequestArgs | Здесь указываются параметры выполнения команды при их необходимости. |
| Answer | В данное свойство записывается результат выполнения задачи, если такой присутствует. |
| AnswerException | Данное свойство хранит ошибку исполнения задачи, необходимую для оповещения запрашиваемой стороны для дальнейшего ее устранения. |

Таблица 3.8 – Методы класса TaskModel

|  |  |
| --- | --- |
| Метод | Описание |
| GetRequest<T>() | Преобразует данные указанные в свойстве Request в необходимый тип данных. |
| GetRequestArgs<T>() | Преобразует данные указанные в свойстве RequestArgs в необходимый тип данных |
| GetAnswer<T>() | Преобразует данные указанные в свойстве Answer в необходимый тип данных. |

### 3.1.4 Блок взаимодействия с интерфейсом пользователя

Данный блок представлен набором API функциональности, реализация которой позволяет взаимодействовать с сервером. Этот API основывается на технологии Microsoft-WPF (Windows Presentation Foundation). Данная технология основывается на паттерне проектирования MVVM (Model-View-ViewModel) и позволяет разделить разработку модельной части приложения от графического интерфейса. Это приводит к таким преимуществам, как ускорение времени разработки программного обеспечения, упрощении отладки и разделении работы программиста от дизайнера. Взаимодействие между графическим интерфейсом и модельной частью происходит через класс API, который оповещает графическую и модельную части о изменениях их состояний. Также оповещение графической части происходит через технология привязки элементов графического интерфейса к полям и методам класса API. Данный класс именуется MainViewModel.

Методы класса MainViewModel описаны в таблице 3.9.

Таблица 3.9 – Методы класса MainViewModel

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| ShowModulesDirectoryMethod() | Открывает директорию хранения модулей на сервере. |
| ShowModulesHelpMethod() | Открывает подсказку по настройке модулей и их конфигурирования. |
| SetServerSettingsMethod() | Открывает окно по настройке сервера администрирования. |
| GetServerDescriptionMethod() | Открывает описание программного обеспечения сервера. |
| OpenErrorLogFileMethod() | Открывает файл логирования ошибок. |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| 1 | 2 |
| ShowClientModulesMethod() | Формирует задачу по запросу получения всех установленных модулей на клиентской машине блоку контроля и взаимодействия с клиентской службой. После получения результата оповещает интерфейс о необходимости отображения результата. |
| DeleteModuleFromClientMethod() | Формирует задачу по удалению определенного установленного модуля на клиентской машине блоку контроля и взаимодействия с клиентской службой. После получения результата оповещает интерфейс о необходимости отображения результата. |
| LoadModuleToServerMethod() | Открывает диалоговое окно, через которое необходимо выбрать модуль для регистрации его на сервере. После выбора модуля формирует задачу блоку хранения и анализа модулей контроля каналов утечки информации для регистрации модуля на сервере. После получения результата оповещает интерфейс о необходимости отображения результата. |
| GetModuleInfoMethod() | Открывает описание выбранного модуля, в котором содержится вся необходимая информация о модуле контроля канала утечки информации. Эту информацию администратор может использовать для более точной конфигурации защиты клиентской машины. |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| 1 | 2 |
| DeleteModuleFromServerMethod() | Формирует задачу блоку хранения и анализа модулей контроля каналов утечки информации по удалению определенного установленного модуля на сервере. После получения результата оповещает интерфейс о необходимости отображения результата. |
| InstallModuleToClientMethod() | Формирует задачу блоку контроля и взаимодействия с клиентской службой по установке выбранного администратором модуля на клиентскую машину. После получения результата оповещает интерфейс о необходимости отображения результата. |
| GetClientModuleWorkResult() | Формирует задачу блоку контроля и взаимодействия с клиентской службой по получению результатов работы модуля установленного на клиентской машине. После получения результата оповещает интерфейс о необходимости отображения результата. |

Рассмотренные выше методы представляют собой API для взаимодействия с сервером, который предоставляет возможность реализации любого вида графического интерфейса для управления сервером администрирования.

Помимо методов данный класс имеет свойства, которые хранят отображаемую информацию в графическом интерфейсе либо данные вводимые пользователем.

Свойства класса MainViewModel описаны в таблице 3.10.

Все свойства хранят временную информацию и при перезагрузке сервера не сохраненная информация будет утеряна.

Таблица 3.10 – Свойства класса MainViewModel

|  |  |
| --- | --- |
| Свойство | Описание |
| ServerIP | Хранит IP-адрес сервера. |
| ServerPort | Хранит порт сервера. |
| DefaultClientPort | Хранит порт по умолчанию, по которому слушает клиентская служба. |
| ServerVersion | Хранит версию программного обеспечения сервера администрирования. |
| Clients | Хранит список всех зарегистрированных клиентов. |
| SelectedClient | Хранит объект клиента, выбранного пользователем. |
| IsClientModulesLoaded | Хранит флаг: загружен ли файл конфигурации модулей, установленных на клиентской машине. |
| LoadedClient | Хранит объект клиента, чья конфигурация загружена на сервер. |
| ClientModules | Хранит список установленных модулей на клиентской машине, загруженного клиента. |
| SelectedClientModule | Хранит объект модуля, установленного на клиентской машине и выбранного пользователем. |
| Modules | Хранит список модулей, находящихся на сервере. |
| SelectedModule | Хранит объект модуля, хранимого на сервере и выбранного пользователем. |

Интерфейс пользователя, который будет взаимодействовать с данным классом для полной совместимости должен поддерживать технологии WPF и привязки элементов. Microsoft предоставляет удобный язык для описания графического интерфейса – XAML (eXtensible Application Markup Language), который поддерживает данные технологии. Это расширяемый язык разметки приложений предоставляющий огромный спектр возможностей по созданию графического интерфейса. Плюсами данного языка являеются быстрота разработки интерфейса, удобство привязки элементов интерфейса к элементам кода C#, а также наличие уже встроенных анимаций.

**3.1.5 Блок контроля и взаимодействия с клиентской службой.**

Взаимодействие с клиентскими службами является важной составляющей DLP – системы. Конфигурирование служб, установка необходимых модулей, своевременное оповещение о произошедших инцидентах и незамедлительное выполнение протокола действий. Для этих целей реализован блок контроля и взаимодействия с клиентскими службами. Данный блок представлен классом ClientController, содержащий методы для взаимодействия с клиентской службой. Общение сервера с клиентом представляет собой получение клиентских запросов, их обработку и отправку ответа на данный запрос. Эти методы также позволяют получать конфигурации клиента, осуществлять добавление новых клиентов и устанавливать на клиентские ЭВМ дополнительные модули, используя при этом протокол TCP. Методы класса ClientController, представлены в таблице 3.11.

Таблица 3.11 – Методы класса ClientController

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| SendTaskToClient(client, taskType, request, recieveTimeOut) | Данный метод отправляет задачу и соответствующий экземпляр класса, содержащий данные для выполнения задачи, удаленному хосту. После обработки и исполнения команды клиентом данный метод возвращает результат работы клиента вызывающему коду. |
| SendTaskToClient(client, taskModel, recieveTimeOut) | Данный метод является перегруженным методом SendTaskToClient, который исполняет тот же функционал, только принимает объект класса TaskModel, в котором уже указана вся необходимая информация для клиента. |
| AddNewConnection(client) | Добавляет запрос клиента в очередь обработки запросов. |
| ProcessNewConnection() | При наличии запросов от клиентов извлекает их из очереди и передает его обработчику. При завершении работы системы закрывает все соединения и удаляет все запросы. |

Продолжение таблицы 3.11

|  |  |
| --- | --- |
| 1 | 2 |
| ProcessClient(socket) | Обрабатывает запрос клиента. Принимает сокет, описывающий соединение с клиентской машиной по сети. |
| CheckAndGetClient(guid, hostname, dnsname, ip) | Проверяет на соответствие клиента. При отсутствии клиента в файле конфигурации регистрирует его. При наличии клиента проверяет записанную о нем информацию, в случае наличия новой информации: обновляет ее. Возвращает объект описания клиента для дальнейшей работы с ним. |

Через метод SendTaskToClient возможно управлять работой клиентской службы, при использовании протокола взаимодействия: управление включает в себя установку новых модулей, удаление старых, конфигурирование уже установленных модулей, запуск или прекращение работы модулей.

Как было сказано выше каждый клиент регистрируется в системе. Для описания состояний клиентов реализован класс Client, свойства которого описаны в таблице 3.12.

Таблица 3.12 – Свойства класса Client

|  |  |
| --- | --- |
| Метод | Описание |
| GUID | Хранит идентификатор клиента, по которому происходит регистрация в системе. |
| HostName | Хранит имя клиентской машины. |
| DnsName | Хранит DNS-имя клиентской машины. |
| IP | Хранит IP-адрес клиенткой машины. |

## Клиентская служба

Клиентская служба состоит из 6 частей: блок непосредственной передачи данных между клиентской службой и сервером, блок контроля модулей защиты каналов утечки информации, блок взаимодействия с сервером, блок клиентской службы, блок управления службой, блок конфигурирования службы. Блок непосредственной передачи данных между клиентской службой и сервером отвечает за сериализацию и десериализацию данных, преобразования массива байтов, представленных форматом – бинарный JSON и наоборот. Эти действия выполняются при помощи функционала библиотеки DLPEngineLibrary. Данный блок описан выше: в разделе описания серверной части системы. Далее описана функциональность остальных блоков клиентской службы.

### 3.2.1 Блок управления модулями контроля каналов утечки информации

Данный блок отвечает за корректное исполнение, установку, удаление, завершение и получение информации о работе модулей защиты каналов утечки информации. Данный блок состоит из 2 классов ModulesController и ModulesExecuter. Методы класса ModulesController описаны в таблице 3.13.

Таблица 3.13 – Методы класса ModulesController

|  |  |
| --- | --- |
| Метод | Описание |
| ModulesDescription() | Возвращает файл конфигурации модулей. При отсутствии файла или при его повреждении начинает сканирование всех установленных модулей. |
| StoreModule(vs) | Анализирует сборку модуля со всеми зависимостями, создает запись в файле конфигурации модулей и устанавливает модуль в определённую директорию. |
| DeleteModule(module) | Удаляет запись о модуле из файла конфигурации и сам модуль из директории. |
| GetModuleWorkResult(module, args) | Ищет указанный модуль, в случае, если модуль не запущен – запускает его с определенными аргументами, и возвращает результат исполнения модуля. |
| GetAssemblyAttribute(assembly) | Считывает атрибут сборки модуля и возвращает его вызывающему коду. |

Класс ModulesController при исполнении или запуске модуля передает контроль над работай модуля классу ModulesExecuter, который в свою очередь совершает всю необходимую проверку перед запуском и контроль работы модуля. Методы класса ModulesExecuter описаны в таблице 3.14.

Таблица 3.14 – Методы класса ModulesExecuter

|  |  |
| --- | --- |
| Метод | Описание |
| SetStartUpArgs(module, args) | Формирует всю необходимую информацию для корректного запуска и исполнения модуля, при необходимости формирует командную строку, которая передается модулю при его запуске. |
| WaitProcessData() | Ожидает завершения работы модуля, при наличии данных возвращаемых модулем сохраняет их в своих полях.  Данный метод выполняется в отдельном потоке, что позволяет продолжить работу системы, пока выполняется модуль. |
| ModulesExecuter(module, args) | Начинает процесс инициализации модуля и его последующий запуск с контролем работы модуля. |

Помимо методов класс ModulesExecuter имеет свойства, при обращении к которым можно получить данные исполняемого модуля. Свойства класса ModulesExecuter описаны в таблице 3.15.

Таблица 3.15 – Свойства класса ModulesExecuter

|  |  |
| --- | --- |
| Свойство | Описание |
| IsDataReady | Данное свойство хранит флаг готовности данных модуля. |
| ProcessResult | Через данное свойство возможно получить результат выполнения модуля. При наличии данных метод сразу же их возвращает. В случае если данные не готовы то вызывающий код будет заблокирован до момента готовности данных. |

### 3.2.2 Блок взаимодействия с сервером

Взаимодействие с сервером является важной частью DLP-системы. Своевременная реакция администратора службы безопасности может предотвратить инциденты утечки информации. Блок взаимодействия с сервером представлен классом ServerInteraction, методы которого описаны в таблице 3.15.

Таблица 3.15 – Методы класса ServerInteraction

|  |  |
| --- | --- |
| Метод | Описание |
| SendTaskToServer(taskType, request, recieveTimeOut) | Данный метод отправляет задачу и соответствующий экземпляр класса, содержащий данные для выполнения задачи, серверу. После обработки и исполнения команды клиентом данный метод возвращает результат работы клиента вызывающему коду. |
| SendTaskToServer(taskModel, recieveTimeOut) | Данный метод является перегруженным методом SendTaskToServer, который исполняет тот же функционал, только принимает объект класса TaskModel, в котором уже указана вся необходимая информация для клиента. |
| AddNewConnection(server) | Добавляет запрос от сервера в очередь обработки запросов. |
| ProcessNewConnection() | При наличии запросов от сервера извлекает их из очереди и передает его обработчику. При завершении работы системы закрывает все соединения и удаляет все запросы. |
| ProcessServer(server) | Обрабатывает запрос клиента. Принимает сокет, описывающий соединение с клиентской машиной по сети. |

Для корректного выполнения данных методов в классе определены поля, хранящие необходимую информацию. Поля класса ServerInteraction, описаны в таблице 3.16.

Некоторые поля данного класса конфигурируются во время запуска службы через параметры запуска.

Таблица 3.16 – Поля класса ServerInteraction

|  |  |
| --- | --- |
| Поле | Описание |
| serviceGuid | Хранит идентификатор службы. |
| serverName | Хранит IP-адрес или имя сервера. |
| serverPort | Хранит порт сервера. |
| request | Хранит очередь, в которую добавляется запросы от сервера. |

### 3.2.3 Блок клиентской службы

Данный блок является главным блоком клиентской службы, из которого происходит управление всей работой клиентской службы. Блок представлен классом ServiceEngine. В данном классе создаются все необходимые экземпляры классов и потоки, выполняющие асинхронно задачи. Методы данного класса представлены в таблице 3.17.

Таблица 3.17 – Методы класса ServiceEngine

|  |  |
| --- | --- |
| Метод | Описание |
| Start() | При запуске службы, вызывается данный метод , который создает все необходимые экземпляры используемых классов и в отдельных потоках запускает остальные блоки службы. |
| Stop() | При необходимости остановки службы вызывается данный метод, который сохраняет все необходимые конфигурации службы и очищает используемые объекты. |

Далее в таблице 3.18 описаны поля класса ServiceEngine.

Таблица 3.18 – Поля класса ServiceEngine

|  |  |
| --- | --- |
| Поле | Описание |
| defaultPathToDir | Путь по умолчанию к директории, в котором находятся файлы службы. |
| serviceDirName | Директории, в котором находятся файлы службы. |
| pathToLogFile | Путь к файлу записи событий. |
| serviceName | IP-адрес или имя клиентской машины, по которому происходит подключение сервера. |
| servicePort | Порт клиентской машины. |

### 3.2.4 Блок управления службой

Данный блок позволяет управлять службой. Функциональность блока представлена в виде программы, которая после регистрации обработчика может отправлять Диспетчеру управления службами запросы на завершение, возобновление или приостановку службы. Для реализации взаимодействия с Диспетчером управления службами данный блок наследует функциональность от класса ServiceBase и переопределяет методы управления службой. Однако, по спецификации Microsoft, выполнение методов управления службами Windows не должно превышать 30 секунд, в противном случае диспетчер управления решит, что служба работает некорректно и завершит ее выполнение. Данные методы описаны в таблице 3.19.

Таблица 3.19 – Методы блока управления службой

|  |  |
| --- | --- |
| Метод | Описание |
| InitializeComponent() | Данный метод инициализирует все необходимые объекты службы, взаимодействующие с операционной системой, в частности с диспетчером управления службами. |
| OnStart() | При необходимости запуска службы диспетчер задач вызывает данный метод, в котором создается главный поток исполнения кода класса ServiceEngine. |
| OnStop() | При завершении службы диспетчер управления службами вызывает данный метод, в котором происходит оповещение главного потока исполнения кода класса ServiceEngine о необходимости завершения работы службы. |
| OnShutdown() | При завершении работы операционной системы диспетчер управления службами вызывает данный метод, в котором происходит оповещение главного потока исполнения кода класса ServiceEngine о необходимости завершения работы службы, который в свою очередь завершает работу всех остальных блоков. |

### 3.2.5 Блок конфигурирования службы

Данный блок предназначен для установки и настройки службы и представляет собой программу конфигурирования. Компоненты .NET могут устанавливаться с помощью простой команды xcopy, однако, установка служб требует соответствующей настройки в системном реестре, чего данная команда не реализует, по этой причине необходима реализация установщика службы, который выполняет все выше перечисленные действия. Методы данного блока описаны в таблице 3.20.

Таблица 3.20 – Методы блока управления службой

|  |  |
| --- | --- |
| Метод | Описание |
| InitializeComponent() | Данный метод выполняет те же действия, что и метод InitializeComponent блока управления, только инициализирует данные, необходимые операционной системе для корректной установки службы. |

В основном данный блок инициализирует служебную информацию, необходимую для установки службы. Свойства, которые описывает служебную информацию приведены в таблице 3.21.

Таблица 3.21 – Свойства блока управления службой

|  |  |
| --- | --- |
| Свойство | Описание |
| Account | Представляет собой структуру данный, описывающую пользователя операционной системы. |
| Description | Представляет описание службы, которое будет находится в соответствующей колонке списка запущенных служб диспетчера управления службами. |
| DisplayName | Имя службы, отображаемое в списке служб. |
| ServiceName | Имя службы, идентифицирующее службу в операционной системе. |
| StartType | Тип запуска службы, например, запустить службу автоматически при запуске операционной системы. |

## Динамически подключаемая библиотека

Библиотека DLPEngineLibrary реализует общий функционал всей системы. Это дает возможности: не повторять написанный функционал, удобство отладки так как функционал представлен одной сборкой, а также использование одного общего блока функциональности, что экономит использование оперативной памяти.

Библиотека состоит из 2 пространств имен: Controllers – реализует методы обработки объектов, работы с файлами и директориями и инкапсулирует реализацию передачи данных по сети; и Models – описывает типы данных, используемых в системе.

Первый класс пространства имен Controllers – это Connection. Как было сказано выше, он реализует передачу данных по сети. Методы данного класса описаны в таблице 3.22.

Данный класс является один из самых необходимых и поддерживает архитектуру интегрирования новых модулей контроля каналов утечки информации

Таблица 3.22 – Методы класса Connection

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| ConnectToHost(hostname, port) | Данный метод при наличии соединения подключается к удаленному хосту и возвращает структуру TcpClient. Принимает 2 параметра: DNS-имя хоста либо IP-адрес и номер порта. |
| StartListener(hostname, port) | Создает абстракцию сокет и начинает прослушивать входящие подключения к хосту. Принимает 2 параметра: DNS-имя либо IP-адрес и номер порта. Возвращает объект класса TcpListener, через который возможно принимать входящие подключения и организовывать передачу данных. |
| SendData(stream, data) | Совершает передачу данных на указанный хост. Принимает 2 параметра: объект класса NetworkStream,который абстрагирует передачу данных по сети и массив байт, представляющих данные для передачи. |

Продолжение таблицы 3.22

|  |  |
| --- | --- |
| 1 | 2 |
| RecieveData(stream) | Данный метод принимает данные от удаленного хоста и возвращает их в виде массива байт. Принимает объект класса NetworkStream. |
| SendData(socket, data) | Совершает передачу данных на указанный хост. Принимает 2 параметра: объект класса Socket,который абстрагирует передачу данных по сети и массив байт, представляющих данные для передачи. |
| RecieveData(socket) | Данный метод принимает данные от удаленного хоста и возвращает их в виде массива байт. Принимает объект класса Socket. |

Следующий класс пространства имен Controllers – это FileController . Данный класс реализует работу с файлами и директориями. Методы данного класса описаны в таблице 3.23.

Таблица 3.23 – Методы класса FileController

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| WriteLogInfo(path, info) | Данный метод необходим для ведения файла логирования событий системы. Принимает 2 параметра: путь к файлу и информацию, записываемую в файл. |
| WriteLogInfo(path, exception) | Данный метод является перегруженным, принимает 2 параметра: путь к файлу логировния и объект класса Exception. Записывает в файл логирования информацию о переданной ошибке в файл логирования. |
| WriteBinaryFile(pathToFile, data) | Записывает бинарный файл, переданный в параметре data, по указанному в параметре pathToFile пути. |

Продолжение таблицы 3.23

|  |  |
| --- | --- |
| 1 | 2 |
| GetSId(wellKnownSidType) | Метод принимает один параметр, являющийся общеиспользуемым идентификатором безопасности. Возвращает данный идентификатор в виде строки. |
| AddDirectorySecurity(dirName, account, rights, controlType) | Метод необходим для добавления политики безопасности для директории. Принимает 4 параметра: dirName – путь к директории, account – имя пользователя или группа, к которому будет относиться политика безопастности, rights – определяет права доступа на указанную директорию для определенного пользователя или группы, controlType – определяет тип политики безопасности, например, запретить доступ или предоставить. |
| RemoveDirectorySecurity(dirName, account, rights, controlType) | Метод необходим для удаления политики безопасности для директории. Принимает 4 параметра: dirName – путь к директории, account – имя пользователя или группа, к которому будет относиться политика безопастности, rights – определяет права доступа на указанную директорию для определенного пользователя или группы, controlType – определяет тип политики безопасности. |

Продолжение таблицы 3.23

|  |  |
| --- | --- |
| 1 | 2 |
| ReadObjectFromFile<T>(pathToFile, reader) | Читает объект из файла при помощи переданного делегата и возвращает прочитанный объект. Принимает 2 параметра pathToFile – путь к файлу, содержащему нужный объект и reader – делегат, который является ссылкой на метод, читающий нужный формат данных. |
| WriteObjectToFile<T>(pathToFile, @object, writer) | Записывает объект в файл при помощи переданного делегата. Принимает 3 параметра pathToFile – путь к файлу, в который будет записан объект, @object – записываемый объект и writer – делегат, который является ссылкой на метод, записывающий объект в нужный формат данных. |
| SaveAsDocxFile(pathToFile, data) | Данный метод сохраняет информацию, переданную в параметре data в файл с расширение .docx по указанному в параметре pathToFile пути. |
| SaveAsPdfFile(pathToFile, data) | Данный метод сохраняет информацию, переданную в параметре data в файл с расширение .pdf по указанному в параметре pathToFile пути. |

Система для передачи данных, хранения временных файлов и записи на диск файлов конфигурации использует тип данных JSON. Для работы с данным типом используется класс JsonController. Методы класса JsonController описаны в таблице 3.24.

Таблица 3.24 – Методы класса JsonController

|  |  |
| --- | --- |
| Метод | Описание |
| ReadObjectFromJsonFile<T>(path) | Данный метод читает JSON объект из файла, находящегося по пути указанному в параметре path и возвращает его вызывающему коду. |
| WriteObjectToJsonFile<T>(path, @object) | Данный метод записывает объект, переданный в параметре @object, в JSON файл, находящийся по пути указанному в параметре path. |
| SerializeToBson<T>(@object) | Сериализует объект @object в формат BSON, который используется для передачи данных по сети. |
| DeserializeFromBson<T> | Десериализует данные в формате BSON в объект и возвращает его. |

Для глубокого сравнение объектов, а также для контроля корректности полей пользовательских типов данных используется класс ObjectController, методы которого описаны в таблице 3.25.

Таблица 3.25 – Методы класса ObjectController

|  |  |
| --- | --- |
| Метод | Описание |
| GetDataAvailabity<T1, T2>(@object, property, comparator) | Данный метод проверяет равенство значения свойства, указанного в параметре property, условию, указанному в параметре comparator, объекта @object. При верности условия возвращает значение свойства. |
| CompareObjects<T>(object1, object2) | Выполняет глубокое сравнивание двух объектов. При равенстве всех полей объектов возвращает правду, в противном случае – ложь. |

Первый класс пространства имен Models – TransmittedData. Данный класс используется для чтения данных из сокета. Он инкапсулирует логику выделения памяти под считанные данные и записи их. Свойства класса TransmittedData, описаны в таблице 3.26.

Таблица 3.26 –Свойства класса TransmittedData

|  |  |
| --- | --- |
| Свойство | Описание |
| Data | Возвращает массив байт, в который записываются данные. |
| Offset | Хранит смешение по массиву байт. |
| Freespace | Хранит размер свободного пространства в массиве байт. |

Все остальные классы пространства имен Models были описаны выше.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Контроль интегрируемых модулей

Так как модули контроля каналов утечки информации являются важной частью данной системы, нужно спроектировать и реализовать алгоритм контроля за этими модулями. Необходимость в контроле за модулями также вытекает из того, что модули могут реализовываться и поставляться разными независимыми разработчиками и для проверки совместимости их с системой нужен предварительный анализ их сборок.

Контроль над интегрируемыми модулями заключается в добавлении их в систему, удалении и считывании информации о модуле, такую как подключаемые зависимости, манифесты, целевую платформу, версию программного продукта, информацию о компании разработчика. Данный контроль осуществляется блоком хранения и анализа модулей контроля каналов утечки информации. Ниже описан каждый алгоритм по отдельности.

Схема программы представлена на чертеже ГУИР 400201 017 ПД.

### 4.1.1 Считывание и анализированние информации о модуле

Для целей интегрирования и корректного использования модулей контроля каналов утечки информации необходимо организовать предварительный анализ модулей. Это является важным неотъемлемым этапом интегрирования модулей.

Перед интегрированием модуля в систему, регистрации и записи выполняется алгоритм получения информации о сборке, описанный ниже.

Первый этап данного алгоритма заключается в поиске файла исполнения и получение сборки модуля, который представлен далее:

Assembly assembly = null;

var archive = ZipFile.OpenRead(path);

foreach(var entry in archive.Entries)

{

var extension = Path.GetExtension(entry.Name);

if(extension == ".exe")

{

using(MemoryStream ms = new MemoryStream())

{

var stream = entry.Open();

stream.CopyTo(ms);

assembly = Assembly.Load(ms.ToArray());

stream.Close();

ms.Close();

}

break;

}

}

archive.Dispose();

return assembly;

Данный алгоритм возвращает сборку модуля, для дальнейшего анализа ее.

На втором и заключительном этапе из полученной сборки читаются ее атрибуты. Данный алгоритм представлен ниже:

object[] attributes = assembly.GetCustomAttributes(typeof(T), true);

if ((attributes == null) || attributes.Length == 0)

return null;

return (T)attributes[0];

Данный алгоритм принимает сборку модуля представленной классом Assembly и тип T, который ограничен базовым классом Attribute. При передачи атрибута наследованного от класса Attribute, данный алгоритм проверяет его наличие и в случае успеха возвращает объект переданного типа атрибута.

В случае, если невозможно получить сборку модуля или один из необходимых атрибутов, а также несовместимости модуля с системой происходит исключение, оповещающее администратора о возникшей ошибке и возможных способах ее решения.

### 4.1.2 Сохранение модуля

После успешного анализа сборки модуля выполняется следующий этап: регистрация модуля в системе, сохранения информации о сборке и записи модуля в базу данных, которая может быть представлена, как локальной директорией так и удаленной на сервере в локальной сети компании или предприятия.

Алгоритм регистрации модуля и записи информации о сборке представлен далее:

var title = GetAssemblyAttribute<AssemblyTitleAttribute>(assembly).Title;

path = Path.Combine(pathToFolder, title + moduleExtension);

description[title] = new DescriptionModule

{

Company = GetAssemblyAttribute<AssemblyCompanyAttribute>(assembly).Company,

Copyright = GetAssemblyAttribute<AssemblyCopyrightAttribute>(assembly).Copyright,

Description = GetAssemblyAttribute<AssemblyDescriptionAttribute>(assembly).Description,

FileVersion = GetAssemblyAttribute<AssemblyFileVersionAttribute>(assembly).Version,

FullName = assembly.FullName,

Product = GetAssemblyAttribute<AssemblyProductAttribute>(assembly).Product,

Title = title,

Trademark = GetAssemblyAttribute<AssemblyTrademarkAttribute>(assembly).Trademark,

Configuration = GetAssemblyAttribute<AssemblyConfigurationAttribute>(assembly).Configuration,

TargetFramework = "",

Path = path

};

После этапа регистрации и записи информации выполняется запись модуля в директорию. Алгоритм описан ниже:

var tempPath = Path.Combine(pathToFolder, "temp", "temp" + moduleExtension);

var tempDir = Directory.CreateDirectory(Path.GetDirectoryName(tempPath));

File.WriteAllBytes(tempPath, vs);

vs = null;

GC.Collect();

if (File.Exists(newPath))

File.Delete(newPath);

File.Move(tempPath, newPath);

if (File.Exists(Path.Combine(pathToFolder, modulesFile)))

File.Delete(Path.Combine(pathToFolder, modulesFile));

FileController.WriteObjectToFile(Path.Combine(pathToFolder, modulesFile), description,

JsonController.WriteObjectToJsonFile);

Directory.Delete(tempDir.FullName);

При сохранении уже существующего модуля алгоритм проверяет его наличие и перезаписывает его данные. Это необходимо по следующим причинам: если вышла более новая версия модуля, то его информация в файле конфигурации модулей обновиться вместе с файлами модуля, или если файлы модуля были повреждены, то возможность перезаписи позволяет быстро исправить данные ошибки путем перезаписи целостной версией.

### 4.1.3 Удаление модуля с сервера

В случае ненадобности модуля или профилактики системы есть возможность удалить модуль с сервера. При этом удаляется как сам модуль так и вся информация о сборке и регистрации модуля. Данный алгоритм описан ниже:

var desc = ModulesDescription;

var res = desc.Remove(pair.Key);

if (res)

{

if (File.Exists(pair.Value.Path))

File.Delete(pair.Value.Path);

if (File.Exists(Path.Combine(pathToFolder, modulesFile)))

File.Delete(Path.Combine(pathToFolder, modulesFile));

FileController.WriteObjectToFile(Path.Combine(pathToFolder, modulesFile), desc,

JsonController.WriteObjectToJsonFile);

}

return res;

## 4.2 Контроль клиентов

Следующей важной частью данной системы является контроль за клиентами. Контроль подразумевает собой установку и конфигурирование модулей на клиенте, удаление модулей, получение информации о клиенте для более легкой идентификации клиентов и получение результатов работы модулей.

### 4.2.1 Идентификация клиентов

Так как все выше перечисленные действия выполняется дистанционно по сети, для предотвращения несанкционированных подключений к системе или атак из сети реализован алгоритм идентификации клиентов, который позволяет определять и игнорировать несанкционированные подключения. Данный алгоритм состоит из нескольких частей.

Первая часть включает в себя соблюдение протокола взаимодействия между клиентами и сервером. Каждый запрос от клиента к серверу, либо наоборот от сервера к клиенту представлен классом TaskModel, который инкапсулирует данные запроса и ответа на запрос. А сам запрос представляет собой транзакцию. В зависимости от типа запроса конечный узел преобразует переданные данные в нужный тип. При неправильности преобразования данных запроса транзакция отменяется, а существующее соединение разрывается в одностороннем порядке. Функционал класса TaskModel описан ниже:

public class TaskModel

{

private TaskModel()

{}

private TaskModel(TaskType task)

{}

private TaskType taskType;

private object request = null;

private object requestArgs = null;

private object answer = null;

private Exception answerException = null;

public TaskModel(TaskType type, object request)

{

this.request = request;

this.taskType = type;

}

public TaskModel(TaskType type, object request, object requestArgs) : this(type, request)

{

this.requestArgs = requestArgs;

}

public TaskType TaskType

{

get => taskType;

set => taskType = value;

}

public object Request

{

get => request;

set => request = value;

}

public object RequestArgs

{

get => requestArgs;

set => requestArgs = value;

}

public object Answer

{

get => answer;

set => answer = value;

}

public Exception AnswerException

{

get => answerException;

set => answerException = value;

}

public T GetAnswer<T>()

{

switch(answer)

{

case JArray array:

return (T)JsonConvert.DeserializeObject<T>(array.ToString());

case JObject jObject:

return (T)JsonConvert.DeserializeObject<T>(jObject.ToString());

default:

return default(T);

}

}

public T GetRequest<T>()

{

switch (request)

{

case JArray array:

return (T)JsonConvert.DeserializeObject<T>(array.ToString());

case JObject jObject:

return (T)JsonConvert.DeserializeObject<T>(jObject.ToString());

default:

return default(T);

}

}

public T GetRequestArgs<T>()

{

switch (requestArgs)

{

case JArray array:

return (T)JsonConvert.DeserializeObject<T>(array.ToString());

case JObject jObject:

return (T)JsonConvert.DeserializeObject<T>(jObject.ToString());

case String str:

return (T)requestArgs;

default:

return default(T);

}

}

}

Вторая часть заключается в формировании каждым клиентом глобального идентификатора на основе аппаратного обеспечения. Глобальный идентификатор предоставляется клиентом серверу в запросе. Это позволяет устранить возможность представления одного клиента другим. При регистрации клиента происходит проверка именно его идентификатора, поэтому изменение параметров клиента, таких как подстановка IP-адреса конечной точки, изменение имени машины либо DNS-имени не влияют на процесс идентификации машины.

Перед выполнением запроса от клиента выполняется проверка: зарегистрирован ли клиент и имеет ли он необходимый идентификатор. Данный алгоритм описан ниже:

var vr = (from i in Clients

where i.Guid == guid

select i).FirstOrDefault();

if (vr == null)

{

vr = new Client()

{

Guid = guid,

HostName = hostname,

IP = ip,

DnsName = dnsname

};

}

else

{

if (vr.HostName != hostname)

vr.HostName = hostname;

if (vr.DnsName != dnsname)

vr.DnsName = dnsname;

if (vr.IP != ip)

vr.IP = ip;

}

return vr;

Также каждый клиент предоставляет информацию о своем локальном IP-адресе, имени хоста и DNS-имени, которая отображается администратору для более легкого определения, с каким именно клиентом он работает.

### 4.2.2 Установка и удаление модулей на клиентской машине

Каждый зарегистрированный клиент возможно конфигурировать путем установки необходимых модулей контроля каналов утечки информации, либо удаления ненужных. Для начала конфигурирования клиента нужно загрузить конфигурацию клиента на сервер для дальнейшей его настройки. Загрузка конфигурации клиента происходит путем отправки команды GetModules, на которую клиент отвечает отправкой списком модулей установленных на нем.

Для получения списка установленных клиент читает конфигурационный файл модулей, имеющий тип Dictionary<string, DescriptionModule>. Алгоритм чтения конфигурационного файла описан ниже:

Dictionary<string, DescriptionModule> description = null;

try

{

description = JsonController.ReadObjectFromJsonFile<Dictionary<string, DescriptionModule>>(

Path.Combine(pathToFolder, modulesDescriptionFile));

}

catch (Exception e)

{

description = new Dictionary<string, DescriptionModule>();

}

return description;

После загрузки конфигурации возможно выполнить установку. Установка выполняется путем отправки команды InstallModule. При получении данной команды, клиент выполняет установку модуля, переданного в поле данных запроса. Установка выполняется методом void StoreModule(byte[] vs), алгоритм которого описан ниже:

var tempFile = Path.Combine(pathToFolder, "temp" + modulePacketExtension);

var tempDir = Path.Combine(pathToFolder, "temp");

Directory.CreateDirectory(tempDir);

File.WriteAllBytes(tempFile, vs);

vs = null;

GC.Collect();

ZipFile.ExtractToDirectory(tempFile, tempDir);

File.Delete(tempFile);

var pathToMainExe = string.Empty;

foreach(var file in Directory.GetFiles(tempDir))

{

if(Path.GetExtension(file) == ".exe")

{

pathToMainExe = file;

break;

}

}

var moduleInfo = CreateModuleInfo(GetAssemblyInfo(pathToMainExe));

moduleInfo.Path = Path.Combine(pathToFolder, moduleInfo.Title, Path.GetFileName(pathToMainExe));

if(Directory.Exists(Path.Combine(pathToFolder, moduleInfo.Title)))

{

var dir = new DirectoryInfo(Path.Combine(pathToFolder, moduleInfo.Title));

dir.Delete(true);

}

Directory.Move(tempDir, Path.Combine(pathToFolder, moduleInfo.Title));

var description = ModulesDescription;

description[moduleInfo.Title] = moduleInfo;

ModulesDescription = description;

Данный алгоритм выполняет чтение сборки модуля, регистрации его в файле конфигурации и установки его для дальнейшего использования. Для переустановки модуля или для обновления версии достаточно выполнить команду установки для уже установленного модуля, что выполнит его переустановку.

В случае, если необходимо удалить модуль с клиентской машины необходимо отправить команду DeleteModule. После получения данной команды, клиент удаляет модуль, указанный в поле данных запроса и удаляет запись о модуле из файла конфигурации. Данные действия выполняет метод bool DeleteModule(DescriptionModule module), алгоритм которого описан далее:

var desc = ModulesDescription;

var res = desc.Remove(module.Title);

if (res)

{

if(Directory.Exists(Path.GetDirectoryName(module.Path)))

{

var dir = new DirectoryInfo(Path.GetDirectoryName(module.Path));

dir.Delete(true);

}

ModulesDescription = desc;

}

return res;

### 4.2.3 Конфигурирование модуля и получение данных модуля

Если модуль реализует протокол взаимодействия со службой, то возможно его конфигурирование и получение результатов его работы. Конфигурирование модулей заключается в отправке параметров конфигурации в поле RequestArgs запроса с указанным модулем для конфигурации.

Для получения информации о работе модуля сервер отправляет команду GetModuleWorkResult, имя модуля для получения информации и аргументы запуска модуля. После получения всех необходимых данных клиент выполняет метод object GetModuleWorkResult(DescriptionModule module, string args), после окончания работы метода он возвращает объект, содержащий результат выполнения модуля. Алгоритм данного метода представлен ниже:

var moduleExecuter = new ModuleExecuter(module, args);

return moduleExecuter.ProseccResult;

Данный метод выполняет работу по созданию объекта класса ModuleExecuter, и возвращению результата работы модуля, который получает от объекта класса ModuleExecuter.

Класс ModuleExecuter является абстракцией над исполнением модуля. Данный класс инкапсулирует запуск модуля, передачу аргументов запуска, настройку и переопределение параметров запуска, получение данных выполнения модуля, ожидания окончания выполнения работы модуля, конкретное завершение и контроль за ошибками выполнения и запуска модуля. Класс ModuleExecuter описан ниже:

public class ModuleExecuter : IDisposable

{

private ProcessStartInfo processStartInfo = null;

private Process process = null;

private bool disposedValue = false;

private bool isDataReady = false;

private Thread processThread = null;

private ManualResetEventSlim dataReadyEvent = null;

private Exception processExeption = null;

private string processData = null;

public bool IsDataReady { get => isDataReady; }

public object ProseccResult

{

get

{

dataReadyEvent.Wait();

if (processExeption != null)

throw processExeption;

return processData as string;

}

}

public ModuleExecuter(DescriptionModule module, string args)

{

SetStartUpArgs(module, args);

process = new Process();

process.StartInfo = processStartInfo;

dataReadyEvent = new ManualResetEventSlim(false);

process.Start();

processThread = new Thread(WaitProcessData);

processThread.IsBackground = false;

processThread.Start();

}

private void SetStartUpArgs(DescriptionModule module, string args)

{

processStartInfo = new ProcessStartInfo();

switch(module.Title)

{

case "FileParsing":

processStartInfo.FileName = module.Path;

processStartInfo.Arguments = args;

processStartInfo.UseShellExecute = false;

processStartInfo.RedirectStandardOutput = true;

processStartInfo.StandardOutputEncoding = System.Text.Encoding.UTF8;

processStartInfo.CreateNoWindow = true;

processStartInfo.WorkingDirectory = Path.GetDirectoryName(module.Path);

break;

}

}

private void WaitProcessData()

{

try

{

var reader = process.StandardOutput;

processData = reader.ReadToEnd();

dataReadyEvent.Set();

isDataReady = true;

reader.Close();

process.WaitForExit();

}

catch(Exception e)

{

processExeption = e;

}

}

private ModuleExecuter()

{ }

private ModuleExecuter(ModuleExecuter moduleExecuter)

{ }

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (processThread.IsAlive)

if(!processThread.Join(TimeSpan.FromSeconds(20)))

{

processThread.Abort();

process.Kill();

}

processStartInfo = null;

process = null;

processThread = null;

dataReadyEvent = null;

processExeption = null;

disposedValue = true;

}

}

~ModuleExecuter()

{

Dispose(disposing: false);

}

public void Dispose()

{

Dispose(disposing: true);

GC.SuppressFinalize(this);

}

}

### 4.2.4 Отправка запросов и получение ответов на запросы

Для выполнения всех выше перечисленных действия необходим функционал отправки запросов и получения ответов на запросы. Для отправки запросов и получения ответов сервер использует два метода TaskModel SendTaskToClient(Client client, TaskType taskType, object request, int recieveTimeOut = 0) и TaskModel SendTaskToClient(Client client, TaskType taskType, object request, int recieveTimeOut = 0). Эти методы описаны ниже:

TaskModel SendTaskToClient(Client client, TaskType taskType, object request, int recieveTimeOut = 0)

{

var packet = new Packet()

{

GUID = serverGuid,

HostName = serverHostName,

Task = new TaskModel(taskType, request)

};

return dataTransfer.SendTaskToHost(client.IP, defaultClientPort, packet, recieveTimeOut).Task;

}

TaskModel SendTaskToClient(Client client, TaskModel taskModel, int recieveTimeOut = 0)

{

var packet = new Packet()

{

GUID = serverGuid,

HostName = serverHostName,

Task = taskModel

};

return dataTransfer.SendTaskToHost(client.IP, defaultClientPort, packet, recieveTimeOut).Task;

}

Клиент для отправки запросов и получения ответов использует метод TaskModel SendTaskToServer(TaskType type, object request)

TaskModel SendTaskToServer(TaskType type, object request)

{

var packet = new Packet()

{

HostName = dataTransfer.LocalHostName,

GUID = serviceGUID,

Task = new TaskModel(type, request)

};

return dataTransfer.SendTaskToHost(serverName, serverPort, packet).Task;

}

Выше перечисленные методы конструируют пакет из переданных данных методу и передают его методу Packet SendTaskToHost(string remoteIp, int remotePort, Packet packet, int recieveTimeOutMillisec = 0), который в свою очередь отправляет пакет на указанный IP-адрес и порт удаленной машины.

## 4.3 Обработка входящих запросов

### 4.3.1 Обработка запросов со стороны сервера

Извлечение входящих подключений из очереди и дальнейшая передача их обработчику запросов происходит в методе void ProcessNewConnection(). Данный метод выполняется в отдельном потоке и ожидает два события: addedNewConnEvent – оповещает поток о том, что в очереди обработки присутствуют активные подключения и closeEvent – оповещает поток о завершении работы системы. Алгоритм метода ProcessNewConnection()описан далее:

void ProcessNewConnection()

{

var handles = new WaitHandle[]

{

closeEvent.WaitHandle,

addedNewConnEvent.WaitHandle

};

do

{

switch (WaitHandle.WaitAny(handles))

{

case 0:

lock (requests)

foreach (var request in requests)

{

request?.Client.Shutdown(SocketShutdown.Both);

request?.Client.Close();

}

return;

case 1:

Socket client = null;

lock(requests)

{

client = requests.Dequeue().Client;

addedNewConnEvent.Reset();

}

ProcessClient(client);

client?.Close();

break;

}

} while (true);

}

Данный метод отправляет подключения обработчику запросов по мере их поступления, либо, в случае завершения работы системы, закрывает все активные подключения, находящиеся в очереди, и завершает свою работу.

Все входящие подключения записываются в очередь обработки Queue<TcpClient> requests через метод bool AddNewConnection(TcpClient client), который использует блок непосредственной передачи данных для добавления нового подключения в очередь обработки запросов. Алгоритм метода AddNewConnection(TcpClient client)описан далее:

bool AddNewConnection(TcpClient client)

{

lock (requests)

{

requests.Enqueue(client);

addedNewConnEvent.Set();

}

return true;

}

После передачи подключения обработчику запросов, выполняется метод ProcessClient(Socket socket). Данный метод принимает пакет от удаленного хоста, извлекает имя машины хоста из пакета, получает IP-адрес и DNS-имя хоста, вызывает метод CheckAndGetClient(packet.GUID, packet.HostName, dns, ip)для идентификации клиента, обрабатывает запрос и затем отправляет сформированный пакет с ответом хосту. Данный алгоритм, с указанной выше последовательностью, описан ниже:

void ProcessClient(Socket socket)

{

Packet packet;

try

{

packet = JsonController.DeserializeFromBson<Packet>(Connection.RecieveData(socket));

}

catch(Exception e)

{

FileController.WriteLogInfo(pathToLogFile, e);

return;

}

string ip = (socket.RemoteEndPoint as IPEndPoint).Address.ToString();

string dns = null;

try

{

dns = Dns.GetHostEntry(ip).HostName;

}

catch (Exception e)

{

FileController.WriteLogInfo(pathToLogFile, e);

dns = packet.HostName;

}

var client = CheckAndGetClient(packet.GUID, packet.HostName, dns, ip);

switch(packet.Task.TaskType)

{

case TaskType.InitClient:

packet.Task.Answer = null;

break;

}

packet.GUID = serverGuid;

packet.HostName = serverHostName;

try

{

Connection.SendData(socket, JsonController.SerializeToBson(packet));

}

catch(Exception e)

{

FileController.WriteLogInfo(pathToLogFile, e);

}

}

После обработки запроса о отправки ответа метод AddNewConnection(TcpClient client)закрывает соединение и переходит к обработки следующего события.

### 4.3.2 Обработка запросов со стороны клиента

Алгоритм предшествующий непосредственно обработки запроса со стороны клиента идентичен серверному, по этой причине следует описать только метод void ProcessServer(Socket server). Алгоритм обработки запроса описан ниже:

try

{

DescriptionModule moduleInfo;

switch (packet.Task.TaskType)

{

case TaskType.GetModules:

packet.Task.Answer = modulesController.Modules;

break;

case TaskType.InstallModule:

modulesController.StoreModule(packet.Task.Request as byte[]);

packet.Task.Answer = modulesController.Modules;

break;

case TaskType.DeleteModule:

moduleInfo = packet.Task.GetRequest<DescriptionModule>();

modulesController.DeleteModule(moduleInfo);

packet.Task.Answer = modulesController.Modules;

break;

case TaskType.GetModuleWorkResult:

moduleInfo = packet.Task.GetRequest<DescriptionModule>();

var args = packet.Task.GetRequestArgs<string>();

packet.Task.Answer = modulesController.GetModuleWorkResult(moduleInfo, args);

break;

case TaskType.GetAllDrives:

var drives = new HashSet<string>();

foreach (var drive in System.IO.DriveInfo.GetDrives())

if (drive.IsReady)

drives.Add(drive.Name);

packet.Task.Answer = drives;

break;

}

}

catch(Exception e)

{

FileController.WriteLogInfo(pathToLogFile, e);

packet.Task.Answer = null;

packet.Task.AnswerException = e;

}

В зависимости от типа запроса выполняется своя часть алгоритма, после чего записывается результат обработки запроса в пакет. В случае возникновения ошибки обработки запроса в пакет помещается информация о ней, эту информацию может использовать администратор для устранения данной ошибки.

После обработки запроса формируется пакет со всей необходимой информацией и отправляется серверу. Данная часть алгоритма описана далее:

try

{

packet.HostName = dataTransfer.LocalHostName;

packet.GUID = serviceGUID;

Connection.SendData(server, JsonController.SerializeToBson(packet));

server?.Close();

return;

}

catch(Exception e)

{

FileController.WriteLogInfo(pathToLogFile, e);

server?.Close();

return;

}

## 4.4 Передача данных по сети

Передача данных основывается на протоколе TCP. А ее исполнение выполняет класс DataTransfer. Функционал данного класса описан далее.

Конструктор данного класса определяет IP-адрес машины, свободный порт. В случае, если эти данные переданы конструктору, то происходит проверка на их корректность . Далее создается прослушивающий сокет по указанным IP-адресу и порту. После запускается асинхронный метод прослушивания входящих подключений BeginAcceptTcpClient(ListenerCallback, listener). При наличии входящего подключения асинхронно вызывается метод void ListenerCallback(IAsyncResult result), алгоритм которого описан далее:

void ListenerCallback(IAsyncResult result)

{

TcpListener listener = result.AsyncState as TcpListener;

try

{

var client = listener.EndAcceptTcpClient(result);

AddNewConnection(client);

listener.BeginAcceptTcpClient(ListenerCallback, listener);

}

catch (ObjectDisposedException)

{

return;

}

catch (Exception e)

{

throw new Exception("Ошибка прослушивания соединений LCB", e);

}

}

Конструктор класса DataTransfer описан ниже:

public DataTransfer(string localIp, int localPort, Func<TcpClient, bool> AddNewConnMethod)

{

foreach(var ip in Dns.GetHostEntry(

Dns.GetHostName()).AddressList)

if (ip.AddressFamily == AddressFamily.InterNetwork)

{

defaultLocalEndPointIP = ip.ToString();

break;

}

defaultLocalEndPointPort = Convert.ToUInt16(

GetAvailablePort(50000));

localEndPointIP = localIp == null ?

defaultLocalEndPointIP : localIp;

localEndPointPort = localPort <= 0 ?

defaultLocalEndPointPort : localPort;

localHostName = Dns.GetHostName();

AddNewConnection = AddNewConnMethod;

try

{

listener = Connection.StartListener(

localEndPointIP, localEndPointPort);

}

catch(Exception)

{

Listener = Connection.StartListener(

defaultLocalEndPointIP, defaultLocalEndPointPort);

}

listener.BeginAcceptTcpClient(

ListenerCallback, listener);

}

При необходимости отправки запроса данный класс предоставляет метод Packet SendTaskToHost(string remoteIp, int remotePort, Packet packet, int recieveTimeOutMillisec = 0). Алгоритм данного метода представлен далее:

Packet SendTaskToHost(string remoteIp, int remotePort, Packet packet, int recieveTimeOutMillisec = 0)

{

var host = Connection.ConnectToHost(

remoteIp, remotePort).Client;

if (recieveTimeOutMillisec != 0)

host.ReceiveTimeout =

recieveTimeOutMillisec;

Connection.SendData(host, JsonController.SerializeToBson(packet));

var answer = JsonController.DeserializeFromBson<Packet>

(Connection.RecieveData(host));

host.Close();

return answer;

}

# 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Неотъемлемой частью разработки любого программного обеспечения является его тестирование. Тестирование является процессом исследования компонентов программного обеспечения и определение качества реализации программного продукта. Разработчик не в силах предусмотреть каждую неисправность, возникшую в процессе разработки, поэтому для предотвращения различного рода неполадок используется тестирование. Кроме того, ошибки в работе могут возникать при внесении какой-либо дополнительной функциональности в исходный код программы, несмотря на то, что код уже отлажен.

Целью тестирования является обеспечение наиболее высокого качества работы разрабатываемого программного продукта. Под качеством понимается совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять предполагаемым и установленным потребностям пользователя, согласно стандарта ISO/IEC 12207-2003.

Производить оценку качества программного продукта можно по многим критериям, одним из которых является надежность. Под надежностью понимается вероятность работы программного продукта без отказов в течение длительного периода времени, рассчитанная с учетом стоимости каждого отказа для конечного пользователя.

Тестирование производилось в два этапа:

1. Тестирование отдельно клиентской и серверной частей в процессе написания программного кода;
2. Тестирование системы защиты от утечек информации в совокупности с разработанным модулем парсинга специальных слов после окончания написания программного кода.

Оба этапа являются необходимыми, исключение одного из них из процесса тестирования невозможно. К примеру, если не проводить модульного тестирования при анализе работы системы в целом, будет происходить множественное количество сбоев в работе, локализировать которые может быть весьма сложной задачей, в то время как при работе модуля в отдельности такие сбои достаточно очевидны.

Тестирование модуля межсетевого взаимодействия производилось с помощью юнит-тестов, идея которых состоит в том, чтобы проверить на корректность небольшие участки довольно крупных и сложных систем, производя валидацию работы системы частями на некотором наборе данных.

Для проведения юнит-тестирования в большинстве языков программирования имеются стандартные инструменты для создания различного рода тестов. Как правило, такие инструменты содержатся в отдельных модулях, которые подключаются разработчиками, и содержат в себе обширную функциональность для проведения всестороннего тестирования систем различного уровня сложности, а также предоставляют инструменты для автоматизированного запуска тестов, производимых специальными командами из консоли.

Среда разработки Microsoft Visual Studio имеет поддержку нескольких пакетов модульного тестирования, однако наиболее популярным из них является MSTest – открытый фреймворк, включенный в состав Visual Studio для тестирования приложений, создаваемых для платформы .NET, создаваемый автоматические тесты. Преимуществами данного фреймворка являются:

* инструменты для создания модульных тестов и организации их в иерархические наборы;
* наличие интерфейса командной строки для реализации тестирования;
* тестирование исключений;
* наличие генераторов отчетов о тестировании;
* поддержка тестирования программного кода, работающего с базой данных;
* высокое качество работы фреймворка;
* низкая стоимость;
* безопасность регрессии сети.

Первым шагом при создании теста в Visual Studio является создание проекта юнит-тестирования, для чего в окне Обозревателя решений создается новый проект, названный EssentialTools.Tests. В разделе шаблонов C# выбирается элемент Test, затем выбирается вариант Unit Test Project. Далее проекту необходимо предоставить ссылку на проект приложения, чтобы получить доступ к исходному коду для выполнения тестирования. Модульные тесты добавляются в файл UnitTest.cs внутри проекта тестирования.

Следующим шагом является внесение следующих изменений:

namespace EssentialTools.Tests

{

[TestClass]

public class UnitTest1

{

private ClassName getTestObject()

{

return new TestedClassName();

}

[TestMethod]

public void TestingMethod()

{

ClassName target = getTestObject();

decimal total = 200;

var temp1 = target.ApplyCount(total);

Assert.AreEqual(total \* 0.5M, temp1);

}

}

}

Для примера приведен фрагмент исходного кода, написанный для иллюстрации модульного тестирования. В данном фрагменте был добавлен одиночный модульный тест. Класс, содержащий тесты, определен атрибутом TestClass, а отдельные тесты представляют собой методы, которые определены атрибутом TestMethod. Метод getTestObject() не является модульным тестом, поскольку не содержит нужного атрибута.

В начале тестового метода, показанного в листинге, вызывается метод getTestObject(), который создает экземпляр объекта для тестирования. Также определяется значение total, для которого непосредственно производится тестирование. Данный этап тестирования называется организацией модульного теста.

На этапе действия теста вызывается метод target.ApplyCount(total), и возвращаемый им результат присваивается переменной temp1. Конечным этапом является утверждение, где применяется метод Assert.AreEqual, который осуществляет проверку того, что полученное значение из метода ApplyCount(total) составляет 50% от изначального значения total.

Класс Assert находится в пространстве имен Microsoft.VisualStudio.TestTools.UnitTesting и содержит набор методов, которые используются в MSTest при тестировании. Данные методы и их краткое описание представлены в таблице 5.1.

Таблица 5.1 – Методы класса Assert фреймворка MSUnit

|  |  |
| --- | --- |
| Метод | Описание |
| 1 | 2 |
| AreEqual<T>(T,T)  AreNotEqual<T>(T,T) | Метод утверждает, что два объекта типа Т имеют одинаковое/различное значение. |
| AreSame<T>(T,T)  AreNotSame<T>(T,T) | Метод утверждает, что две переменные типа Т ссылаются на один/разные объекты. |
| Fail() | Метод указывает на отрицательный результат утверждения, то есть никакие условия не проверены. |
| Inconclusive() | Метод показывает, что результат юнит-теста не может быть установлен однозначно. |
| IsTrue()  IsFalse() | Метод утверждает, что логическое значение равно true/false. Используется для оценки |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| 1 | 2 |
|  | выражения, возвращающего логический результат. |
| IsNull(object)  IsNotNull(object) | Метод утверждает, что переменная не присвоена/присвоена ссылке на объект. |
| IsInstanceOfType(object,  type)  IsNotInstanceOfType  (object,type) | Метод утверждает, что объект относится/не относится к указанному типу или является/не является производным для указанного типа. |

Методы, представленные в таблице 5.1, являются удобными инструментами при создании множества тестов, значительно ускоряющими время их написания и избавляя разработчика от дополнительной реализации данной функциональности. Практически все представленные методы использовались при написании тестов для модуля парсинга специальных слов, так как они являются наиболее простыми и удобными инструментами тестирования аспектов приложения для среды .NET.

Каждый метод класса Assert позволяет проверить какой-либо аспект юнит-теста, и, если проверка не проходит, они генерируют исключение. Для прохождения теста все утверждения должны успешно завершиться.

Не менее важной составляющей пространства имен Microsoft.VisualStudio.TestTools.UnitTesting является атрибут ExpectedException. Он представляет собой утверждение, которое дает положительный результат в случае, если юнит-тест генерирует исключение с типом, указанным в параметре ExceptionType. Данный атрибут является надежным способом для обеспечения генерации исключений и избавляет от создания блоков try…catch внутри кода юнит-теста.

Создание тестов можно производить на любом этапе разработки. Основной задачей тестирования является покрытие необходимой функциональности итогового продукта. Существует несколько различных принципов тестирования, в зависимости от момента времени, в который тестирование внедряется в разработку программного обеспечения.

Одним из часто встречающихся принципов является Test-driven Development. Суть данного принципа заключается в том, что система и ее компоненты должны быть описаны при помощи тестов в самом начале процесса разработки программного обеспечения, то есть каждая существенная функция в программе должна быть заключена в test-case, а затем происходит реализация описанных ранее функций, покрывая таким образом уже написанные тесты. Такой подход к тестированию позволяет достичь максимальной интеграции тестирования, но вместе с тем возникает дополнительная нагрузка на разработчика, который, помимо проектирования системы, должен грамотно подобрать правильный набор тестов, которые должна удовлетворить разрабатываемая система.

Отдельным элементом является тестирование реализуемых компонентов на доступность, особенно для систем, использующих компьютерные сети, в частности сеть Интернет. Поэтому при разработке веб-приложений и подобных им систем уделяется особое внимание аспектам работы, связанным с использованием сетей для передачи данных как для доступа к системе, так и для обмена данными между сервисами, сходящими в состав программного обеспечения.

После написания юнит-тестов для некоторых отдельно взятых элементов программного продукта, их выполнения и исправления недостатков в работе модуля, следует произвести тестирование системы защиты от утечек информации в целом, используя интеграционное тестирование.

Интеграционное тестирование предназначено для оценки компонентов программного продукта на более высоком уровне, чем юнит-тесты. Такие тесты позволяют убедиться, что два или более компонента программного продукта работают совместно и предоставляют пользователю ожидаемый результат, включая все компоненты, необходимые для полноценной обработки запросов. Интеграционные тесты широко используются для тестирования инфраструктуры программного обеспечения и включает такие компоненты, как база данных, сетевые устройства, файловая система, а также конвейер «запрос-ответ». В отличие от модульных, данные тесты используют фактические компоненты, используемые приложением в рабочей среде, требуют большего объема кода и выполняются более длительный промежуток времени. Поэтому использование интеграционных тестов рекомендуется только для наиболее важных сценариев инфраструктуры.

Данный вид тестирования может производиться в двух аспектах: требования и бизнес-процессы. Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу дизайна тестовых случаев. В данном случае разрабатывается список компонентов, которые необходимо протестировать, а также расставить приоритеты требованиям на основе возможных рисков и, на основе этого, приоритезировать тестовые сценарии. Такой подход позволяет акцентировать внимание на наиболее важные элементы программного продукта.

Тестирование в перспективе «бизнес-процессы» использует знания этих бизнес-процессов, которые описывают сценарии ежедневного использования программного продукта. Тестовые сценарии в данном случае основываются на случаях использования программы.

Интеграционное тестирование является значимым дополнением модульных тестов. В то время, как модульные тесты выполняют проверку отдельных более значимых компонентов на сервере, интеграционные тесты ориентированы на клиентскую службу, воссоздавая действия пользователя. В результате вскрываются проблемы, возникающие при взаимодействии между компонентами программы. Данный вид тестирования не лишен и недостатков, одним из которых является длительность создания и выполнения тестов. К тому же, такие тесты являются достаточно хрупкими, изменение одного атрибута может привести к тому, что тест не будет пройден.

Учитывая тот факт, что интеграционное тестирование требует дополнительного времени и усилий, такой вид тестирования применяется к наиболее важным компонентам программного продукта и на более важных этапах разработки, например, после еженедельного контроля исходного кода или после завершения разработки основных функциональных блоков. Тем не менее, интеграционные тесты не менее важны по сравнению с юнит-тестами, и они позволяют выявлять проблемы, недоступные модульным тестам. К тому же, время, затрачиваемое на интеграционное тестирование, полностью окупается, и его использование в процессе разработки программного обеспечения необходимо.

Тестирование системы защиты от утечек информации в совокупности с модулем парсинга специальных слов производилось двумя пользователями на разных компьютерах с разной производительностью и операционными системами: Windows 7 и Windows 10.

Проводимые над программным продуктом тесты представлены в таблице 5.2.

Таблица 5.2 – Функциональные тесты

|  |  |  |
| --- | --- | --- |
| Содержание теста | Ожидаемый результат | Тест пройден |
| 1 | 2 | 3 |
| Установка службы через командную строку Windows | Успешная регистрация службы в реестре и ее установка | Да |
| Конфигурирование службы и ее дальнейший запуск | Успешный парсинг аргументов запуска службы и ее дальнейший | Да |
| Подключение службы к удаленному серверу | Успешное подключение к серверу и запись результата подключения в файл логирования | Да |
| Остановка службы | Успешная остановка службы с сохранением состояния | Да |
| Перезапуск службы | Успешный перезапуск службы | Да |
| Удаление службы через командную строку Windows | Успешное удаление службы и записей в реестре | Да |
| Установка типа запуска | Успешное изменение типа запуска | Да |

Продолжение таблицы 5.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Установка сервера через установщик | Успешная установка сервера на компьютер | Да |
| Удаление сервера через деинсталятор | Успешное удаление сервера с компьютера | Да |
| Конфигурирование сервера через установку параметров сети | Успешное изменение IP-адреса и порта сервера | Да |
| Запуск сервера | Успешный запуск сервера | Да |
| Завершение работы сервера | Успешное завершение работы сервера и сохранение всех конфигураций | Да |
| Поиск файлов, используя пустой словарь | Сообщение о необходимости заполнения словаря ключевыми словами | Да |
| Добавление, удаление и перезапись словоформ из словаря через консоль | Успешное добавление, удаление и перезапись словоформ в JSON-файле словаря | Да |
| Вывод словаря и доступных расширений в консоль | Успешный вывод словаря и доступных расширений | Да |
| Получение запроса от клиентской службы | Успешное получение запроса | Да |
| Запуск модуля из диалогового окна сервера администрирования | Открытие диалогового окна модуля | Да |
| Парсинг файлов без указания атрибутов поиска | Сообщение о необходимости выбора атрибутов поиска | Да |
| Добавление, удаление и перезапись словоформ из словаря через диалоговое окно модуля | Успешное выполнение операций добавления, удаления и перезаписи словоформ | Да |
| Начало работы парсинга с использованием словаря | Успешное выполнение работы и вывод результата в диалоговом окне | Да |
| Начало работы парсинга с использованием словоформ | Успешное выполнение работы и вывод результата в диалоговом окне | Да |
| Вывод результата парсинга в диалоговом окне | Вывод результата в виде списка файлов с указанием пути и найденных совпадений | Да |

Продолжение таблицы 5.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Ответ на запрос клиентской службы | Отправка результата клиентской службе | Да |
| Экспорт результата парсинга в файл | Формирование файлов с расширениями .txt, .doc и .pdf, запись в них результата парсинга | Да |
| Загрузка модуля на сервер | Успешное считывание, регистрация и загрузка модуля на сервер | Да |
| Удаление модуля с сервера | Успешное удаление файла конфигурации модуля и самого модуля | Да |
| Подключение службы к серверу | Успешное подключение службы и ее регистрация в системе | Да |
| Получение конфигурации клиента | Успешная загрузка конфигурации клиента | Да |
| Установка модуля на клиентскую машину | Успешная установка и регистрация модуля на клиенте | Да |
| Удаление модуля с клиентской машины | Успешное удаление модуля с клиентской машины и удаление записи из файла конфигурации | Да |
| Запрос результат выполнения модуля | Успешное получение сервером результата выполнения модуля на клиентской машине | Да |

Тестирование модуля проводилось как в ходе написания исходного кода, так и после реализации функциональности в совокупности с модулем парсинга специальных слов. По результатам тестирования, описанным в таблице 5.2, можно сделать вывод о том, что созданные функциональные блоки работают корректно и предоставляют пользователю достоверную информацию о работе.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Результатом реализации дипломного проекта является модуль межсетевого взаимодействия DLP-системы, в который поддерживает интеграцию модулей контроля каналов утечки информации.

## 6.1 Требования к программному и аппаратному обеспечению

Минимальные требования к программному и аппаратному обеспечению:

* 1 гигабайт оперативной памяти для сервера;
* 1 гигабайт оперативной памяти для клиента;
* наличие доступа к локальной сети;
* 120 мегабайт свободного места на жестком диске для сервера;
* 100 мегабайт свободного места на жестком диске для клиента;
* наличие установленного фреймворка .NET Framework 4.7.2. и новее.

## 6.2 Руководство по установке клиентской службы

Для установки службы достаточно запустить инсталятор SokolServiceSetup(x64).exe для 64-разрядной системы и SokolServiceSetup(x86).exe для 32-разрядной системы. Данный инсталятор устанавливает службу, регистрирует ее в реестре системы и производит все необходимые инициализации, необходимые для успешной установки службы, поэтому никаких лишних действия от пользователя не требуется.

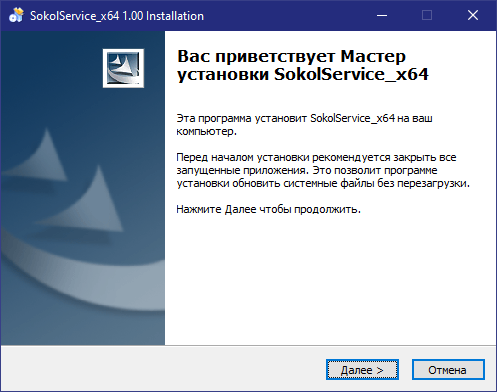


Рисунок 6.1 – Окно приветствия установщика службы

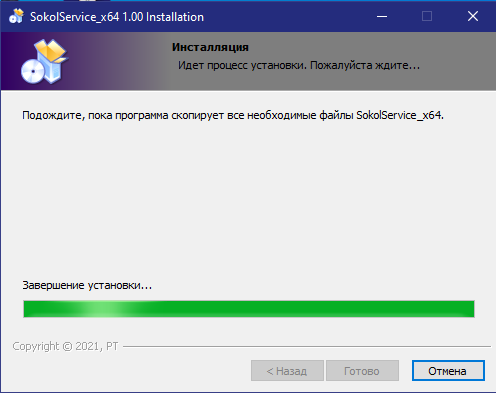


Рисунок 6.2 – Окончание установки

Этапы установки службы указаны на рисунках 6.1 – 6.2. Для удаления службы достаточно запустить деинсталятор Uninstall.exe, который удалит службы, очистит реестр от записей службы, рисунок 6.3:

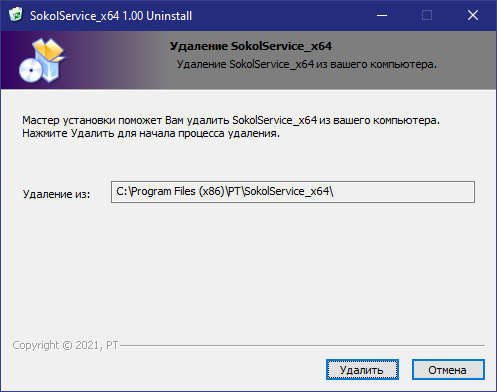


Рисунок 6.3 – Удаление службы

## 6.3 Руководство по установке сервера

Для установки сервера достаточно запустить инсталятор сервера SokolServerSetup.exe, который предложит выбрать каталог установки и создание ярлыков.

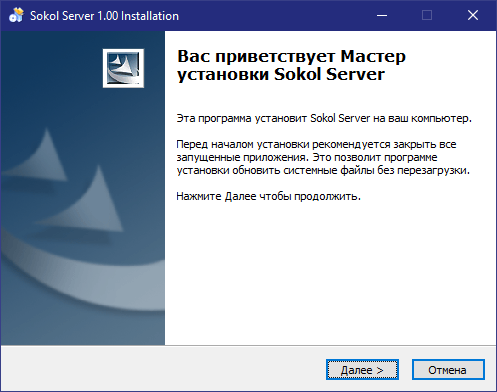


Рисунок 6.4 – Окно приветствия установщика сервера

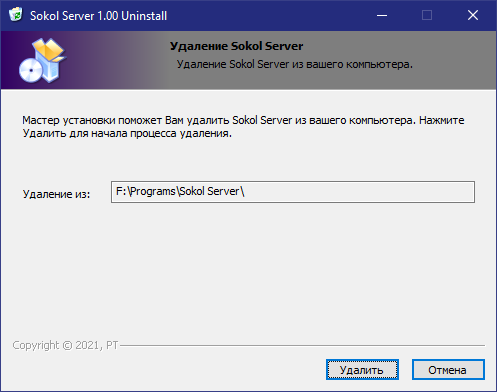


Рисунок 6.5 – Удаление сервера

На рисунке 6.4 показано окно приветствия инсталятора сервера.

Для удаления сервера достаточно запустить деисталятор, который удалит сервер со всеми каталогами созданными сервером, рисунок 6.5

## 6.4 Руководство по управлению сервером

При первом запуске сервера пользователь увидит главное окно не сконфигурированного сервера, рисунок 6.6



Рисунок 6.6 – Главное окно сервера

Справа находится колонка, в которой будут отображаться подконтрольные клиенты, слева указаны модули загруженные на сервер, по центру конфигурация клиента, сверху находиться панель инструментов.

На панели инструментов находятся кнопки управления сервером и раскрывающиеся меню, рисунок 6.7:

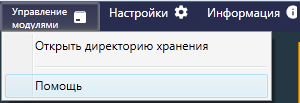


Рисунок 6.7 – Панель инструментов сервера

При выборе «Открыть директорию хранения» в меню «Управление модулями» откроется директория хранения модулей на сервере. Если необходимо настроить сервер, то нужно выбрать «Настройки» после чего откроется окно конфигурирования сервера, рисунок 6.8, в котором можно сконфигурировать службу, после чего сохранить, либо отменить изменения.

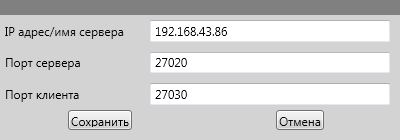


Рисунок 6.8 – Окно конфигурирования сервера

В последнем меню можно получить информацию о сервере: открыть файл логирования, рисунок 6.9; получить информацию о программном продукте, рисунок 6.10.

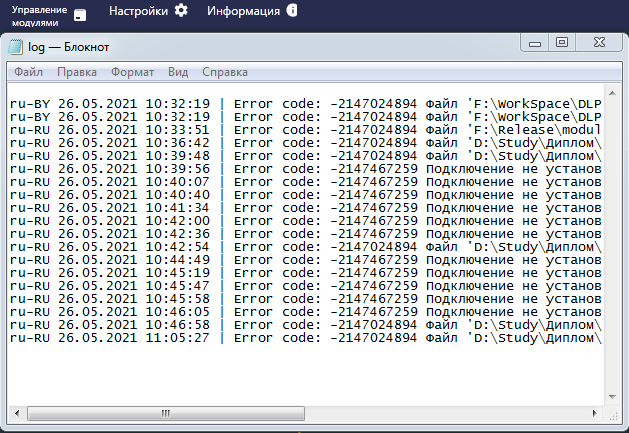


Рисунок 6.9 – Окно конфигурирования сервера

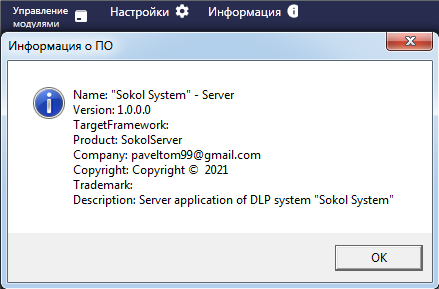


Рисунок 6.10 – Окно конфигурирования сервера

Для конфигурирования клиентов перед этим необходимо загрузить модули на сервер. Это выполняется по нажатию правой кнопки мыши по колонке модулей и выбору «Загрузить модуль на сервер», рисунок 6.11:

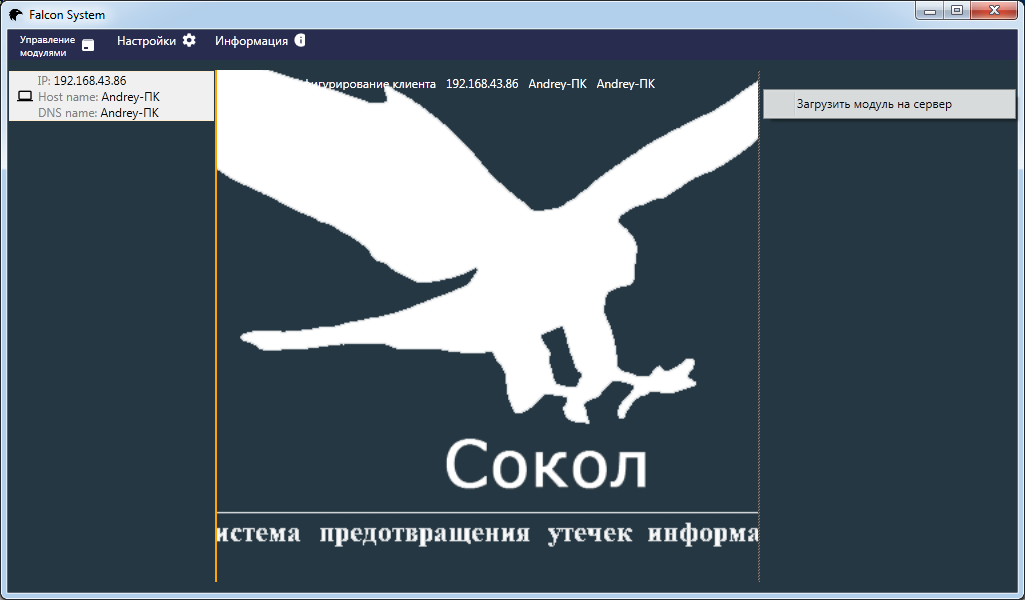


Рисунок 6.11 – Загрузка модуля на сервер

После этого откроется диалоговое окно, в котором необходимо указать загружаемый модуль.

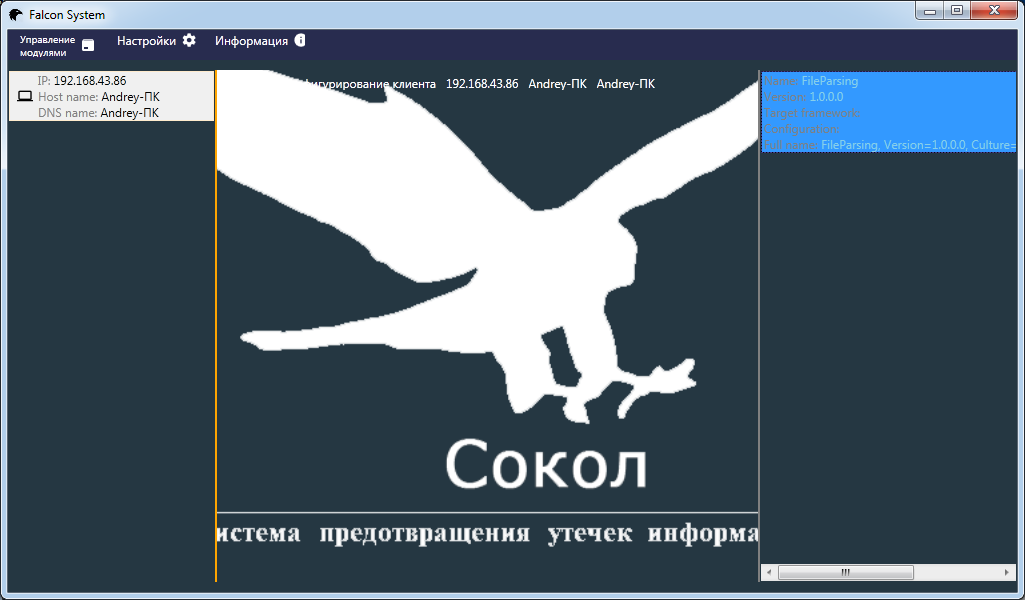


Рисунок 6.12 – Успешная загрузка модуля

После успешной загрузки модуля он отобразиться в колонке модулей, рисунок 6.12.

Перед началом конфигурирования клиента необходимо запросить конфигурацию клиента. Это производиться по нажатию на кнопку «Просмотреть установленные модули», после сервер запросит конфигурацию клиента и отобразит ее, рисунок 6.13:

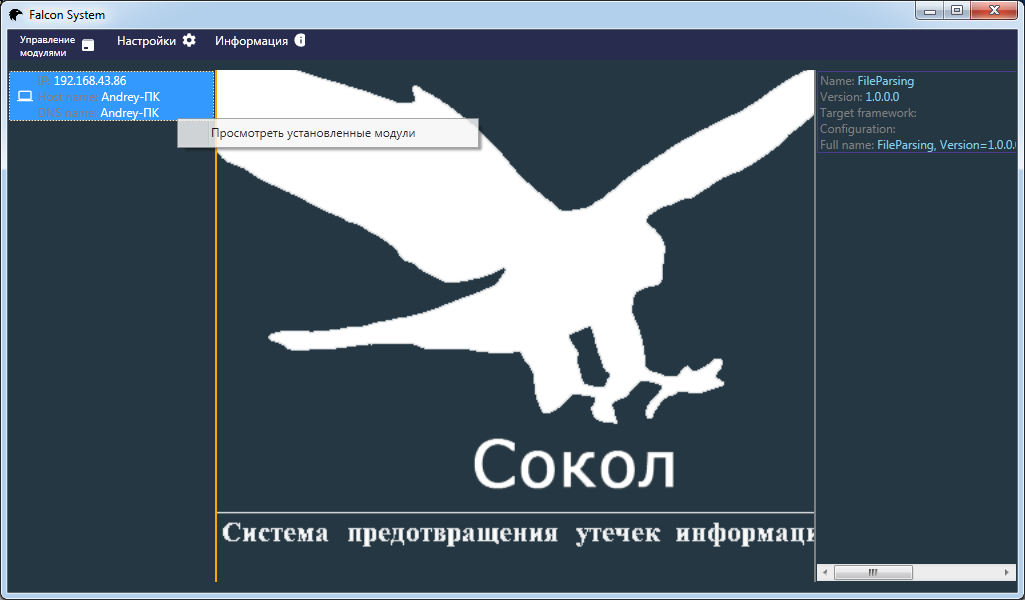


Рисунок 6.13 – Запрос конфигурации клиента

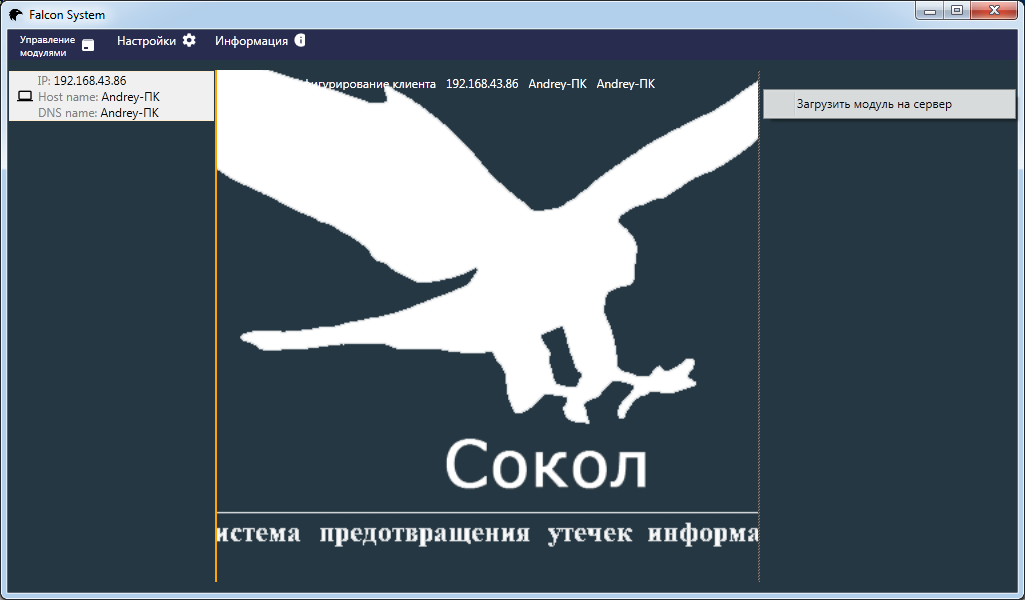


Рисунок 6.14 – Успешная загрузка конфигурации клиента

После успешной загрузки конфигурации клиента, рисунок 6.14, вверху главного окна появиться подсказка, о том что возможно конфигурирование клиента. Установка и использование модуля возможны не только на компьютерах с ОС Windows, но и на компьютерах с любой другой операционной системой. Для того, чтобы эксплуатировать модуль на определенной ОС, необходимо установить .NET Framework версии 4.7.2 и поддерживаемую им общеязыковую среду выполнения. В качестве примера показана установка модуля на компьютер с ОС Windows.

При установке модуля на сервер администрирования указывается путь к директории модуля, запакованной в ZIP-архив. После установки директория с исполняемым файлом модуля находится в папке Modules сервера администрирования. В той же директории находится дескриптор модуля, в котором содержится информация о названии модуля, версии и авторе.

Загруженный на сервер модуль может устанавливаться пользователем на клиентские компьютеры следующим образом, рисунки 6.15 – 6.16 . Если клиент не был ранее конфигурирован, то конфигурация будет пустой, для выполнения ее необходимо установить нужные модули на клиентскую машину.

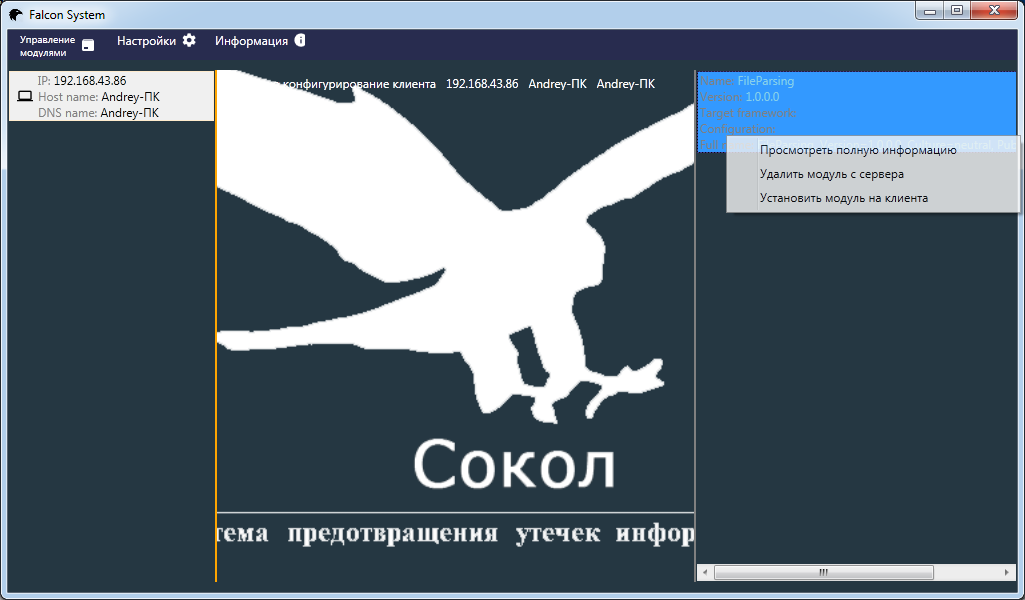


Рисунок 6.15 – Выбор необходимого модуля для установки на клиентскую машину

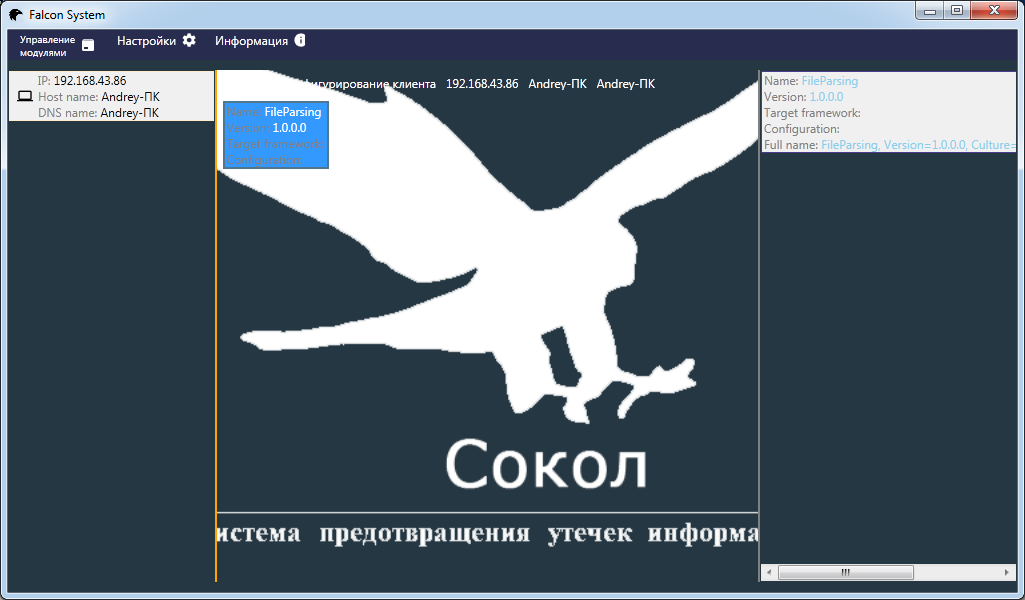


Рисунок 6.16 – Успешная установка модуля

После успешной конфигурации клиента можно просмотреть информацию о модуле, рисунок 6.17, запустить его или настроить для определенных задач, входящих в его спектр работы. Все эти действия рассмотрим на примере модуля парсинга специальных слов. Общий вид диалогового окна модуля представлен на рисунке 6.18.

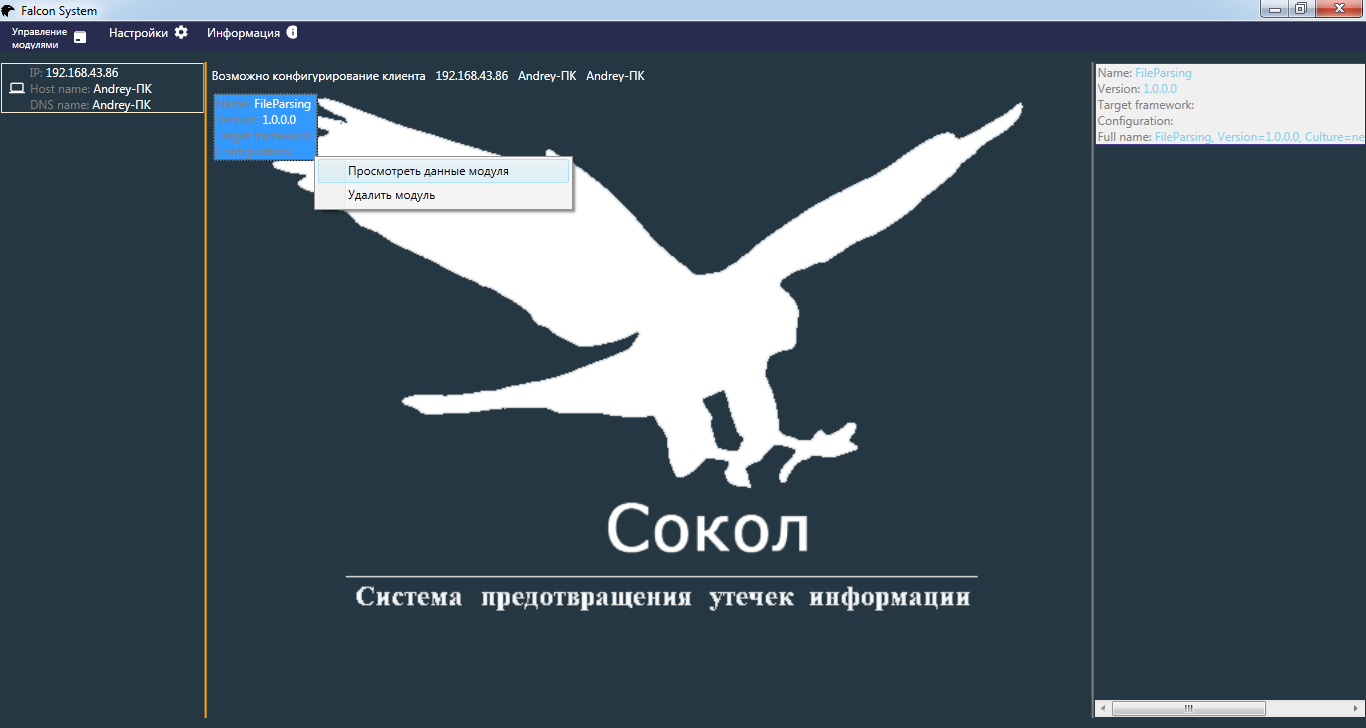


Рисунок 6.17 – Просмотр данных модуля

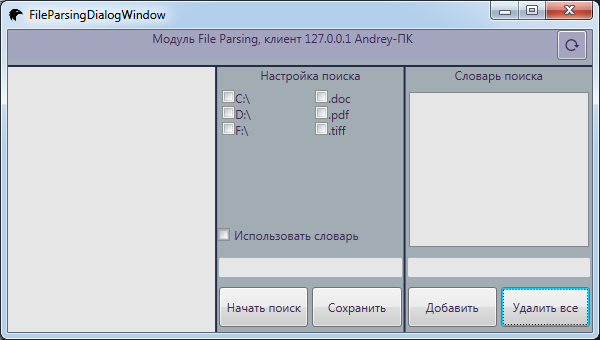


Рисунок 6.18 – Общий вид диалогового окна модуля

Для активизации модуля необходимо нажать кнопку «Начать поиск». В случае нажатия кнопки без указания каких-либо атрибутов пользователь получит сообщение о необходимости выбора того или иного атрибута. Пример сообщения о необходимости выбора логического диска проиллюстрирован на рисунке 6.19:

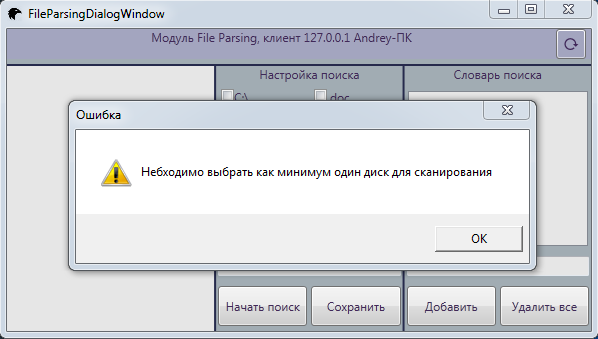


Рисунок 6.19 – Сообщение о необходимости выбора атрибута

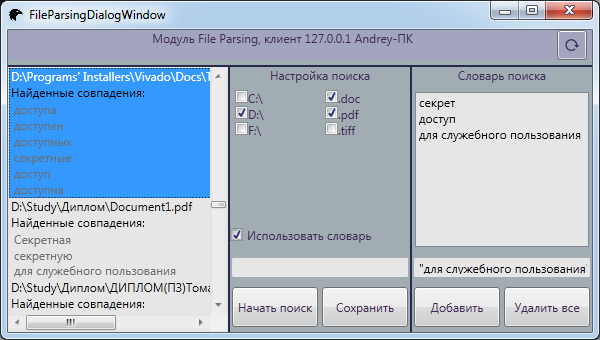


Рисунок 6.20 – Результат работы модуля

Выбор атрибутов для проведения парсинга происходит путем постановки галочек напротив нужных пользователю атрибутов

Результат работы модуля с использованием словаря ключевых слов представлен пользователю в виде списка файлов с указанием пути и обнаруженными ключевыми словами, рисунок 6.20.

Пользователь может сохранить результат выполненной работы парсинга, нажав на кнопку «Сохранить».

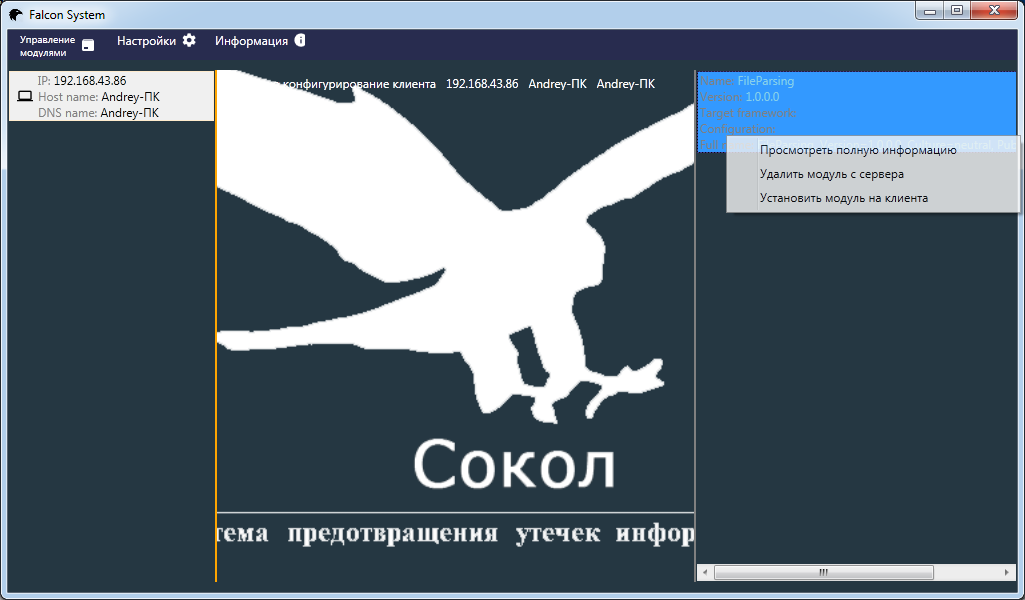


Рисунок 6.21 – Удаление модуля с клиентского компьютера

Модуль, установленный на клиентский компьютер, привязан к клиентской службе, получая от нее запросы и формируя ответы в виде результата работы. Чтобы удалить модуль с клиентского компьютера, администратору системы достаточно нажать на установленный модуль правой кнопкой мыши и нажать кнопку «Удалить модуль», рисунок 6.21.

# 7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ МОДУЛЯ МЕЖСЕТЕВОГО ВЗАИМОДЕЙСТВИЯ DLP-СИСТЕМЫ

## 7.1 Характеристика разработанного программного средства

Целью данного дипломного проекта является разработка и реализация модуля межсетевого взаимодействия для DLP-системы, на основе универсального протокола взаимодействия. Данный модуль предназначен для обеспечения взаимодействия клиентской службы с сервером администрирования и интегрирования новых модулей в систему.

Данная система предоставляет следующие функции:

* конфигурирование работы клиентов сервером администрирования;
* интегрирование модулей, которые будут контролировать определённые каналы утечки информации;
* запуск модулей клиентом и контроль их работы;
* обновление модулей клиентами;
* управление работой модулей клиентами;
* автоматический запуск клиентского модуля вместе с Windows;
* установка связи клиентской службы с сервером при наличии соединения;
* сохранение информации о результатах работы модулей системы на клиентской ЭВМ и дублирование ее на сервере при наличии соединения.

## 7.2. Расчет инвестиций в разработку программного средства

Для организации-заказчика инвестициями в разработку программного средства является цена программного средства, которая может определяться следующими альтернативными способами:

- на основе полных затрат на разработку программного средства организацией-разработчиком

- в процессе переговоров между разработчиком и заказчиком (;

- на основе средних рыночных цен на программные средства .

Таблица 7.1 – Формирование цены программного средства на основе затрат

|  |  |  |
| --- | --- | --- |
| Наименование статьи затрат | Расчет по формуле | Значение, р. |
| 1 | 2 | 3 |
| 1. Основная заработная плата разработчиков | Формула (7.1), см табл. 7.2 | 7335 |

Продолжение таблицы 7.1.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 1. Дополнительная заработная плата разработчиков |  | 880,2 |
| 1. Отчисления на социальные нужды |  | 2842,4 |
| 1. Прочие расходы |  | 2567,2 |
| 1. Общая сумма инвестиция (затрат) на разработку |  | 13624,8 |
| 1. Плановая прибыль, включаемая в цену программного средства |  | 4087,4 |
| 1. Отпускная цена программного средства |  | 17712,2 |

Расчет основной заработной платы разработчиков рассчитывается по формуле

где Кпр – коэффициент премий;

n – категории исполнителей;

Зчi – часовая заработная плата исполнителя i-й категории;

ti – трудоемкость работ.

Таблица 7.2 – Расчет затрат на основную заработную команды разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячная заработная плата, р. | Часовая заработная плата, р. | Трудоемкость работ, ч. | Итого, р. |
| 1 | 2 | 3 | 4 | 5 |

Продолжение таблицы 7.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Инженер-Программист | 1800 | 10.7 | 180 | 1926 |
| Ведущий Инженер-программист | 3300 | 19.8 | 170 | 3366 |
| Дизайнер | 1050 | 6.2 | 150 | 937 |
| Тестировщик | 1100 | 6.5 | 169 | 1106 |
| Всего затраты на основную заработную плату разработчиков | | | | 7335 |

## 7.3 Расчет результата от разработки и использования программного средства

### 7.3.1 Экономический эффект от разработки программного средства для организации-разработчика

Экономический эффект от разработки программного средства по индивидуальному заказу рассчитан на основе цены, установленной в процессе переговоров между разработчиком и заказчиком, т. к. создаётся под нужды и требования конкретного заказчика и является уникальным, то экономический эффект у организации-разработчика будет определяться по формуле

где ‒ цена, установленная на основе переговоров между заказчиком и организацией-разработчиком (без НДС), р.; ‒ затраты на разработку программного средства, р.

### 7.3.2 Экономический эффект от использования программного средства для организации-заказчика

Экономическим эффектом в результате использования программного средства является прирост чистой прибыли, полученный за счет:

* экономии затрат на заработную плату с начислениями на заработную плату служащих в связи с сокращением их численности;
* экономии материальных затрат, электроэнергии, затрат на оплату труда, технологических потерь;
* снижение материальных затрат на производство продукции.

Наиболее подходящий экономический эффект для данного дипломного проекта является экономия на заработной плате и начислениях на заработную плату в результате сокращения численности работников, который рассчитывается по формуле

где Кпр – коэффициент премий;

– численность работников i-й категории, высвобожденных после внедрения программного средства;

Зi – годовая заработная плата высвобождаемых работников.

Экономическим эффектом при использовании программного средства является прирост чистой прибыли, полученный за счет экономии на текущих затратах предприятия, который рассчитывается по формуле

где Этек – экономия на текущих затратах при использовании программного средства;

– прирост текущих затрат связанных с использованием программного средства;

Нп – ставка налога на прибыль согласно действующему законодательству.

– затраты на совершенствование программного средства, является основной заработной платой р.(см таб. 7.1)

## 7.4 Расчет показателей экономической эффективности разработки и использования программного средства

### 7.4.1 Оценка экономической эффективности для организации-разработчика

Оценка экономической эффективности разработки осуществляется с помощью расчета простой нормы прибыли по формуле

где – прирост чистой прибыли, полученный от разработки программного средства организацией-разработчиком по индивидуальному заказу, р.;

Зр – затраты на разработку программного средства организацией-разработчиком, р.

### 7.4.2 Расчет экономической эффективности для организации-заказчика

Так как сумма инвестиций меньше суммы годового экономического эффекта, т.е. инвестиции окупятся менее чем за год, оценка экономической эффективности инвестиций будет рассчитываться по следующей формуле

где – прирост чистой прибыли, полученный от использования разработанного программного средства по индивидуальному заказу, р.;

Цпс – цена программного средства, рассчитанная на основе затрат (см. табл.7.1), р.

В данном разделе была рассчитан экономическая эффективность, срок окупаемости инвестиций, экономический эффект от использования программного средства. По данным расчетам инвестиции окупятся меньше чем за один год использования данного программного средства.

В результате расчетов полная себестоимость готового продукта составляет 17712,2.

# ЗАКЛЮЧЕНИЕ

Проводя сравнительный анализ систем предотвращения утечек информации, можно сделать вывод, что на данный момент в сети Интернет существует малое количество аналогов, позволяющих конфигурировать клиентов, в зависимости от необходимых требований. Реализация архитектуры модуля межсетевого взаимодействия системы предотвращения утечек информации предоставляющей дальнейшую интеграцию модулей контроля каналов утечки информации и расширения всей системы позволяет дальнейшую поддержку разработанного программного продукта. В ходе работы над дипломным проектом был разработан модуль межсетевого взаимодействия системы предотвращения утечки информации. Модуль позволяет администратору локальной сети удаленно конфигурировать клиентов и производить контроль компьютеров и предпринимать необходимые меры по защите информационной безопасности. Для эксплуатации модуля создано клиентсерверное приложение, в котором клиентская часть реализована в виде службы, в которую интегрируются разработанные модули контроля каналов утечки информации. Сервер представлен десктопным приложением, который является центром конфигурирования подконтрольных клиентов. Для удобства использования системы реализован интерфейс пользователя, позволяющий наглядно увидеть возможности системы и способы дальнейшего ее улучшения. В дальнейшем возможно реализовать расширенные возможности конфигурации сервера, улучшение алгоритмов обработки запросов для увеличения быстроты их выполнения, возможности настройки автономных действий службы в случае отключения сервера, также реализации интерфейса, удаленно подключаемого к серверу, для управления системой и алгоритма аутентификации и авторизации администратор системы. Реализованный модуль также не лишен недостатков. Первый недостаток заключается в том, что сервер интегрирован с интерфейсом пользователя, но данный недостаток решается путем реализации функционала удаленного взаимодействия этих частей системы. Вторым недостатком является малое количество функций конфигурации как клиентов так и сервера, однако использование при разработке фреймворка .NET и языка C# предусматривает расширяемость модуля путем внесения несложных дополнений в исходный код и использованием уже готовых библиотек для работы с файлами различного рода

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Системы DLP – Who? What? Where? How? [Электронный ресурс].–Электронные данные. – Режим доступа: http://www.topsbi.ru/default.asp?artID=1675[. –](https://www.domo.com/learn/data-never-sleeps-7.%E2%80%93) Дата доступа: 30.03.2021.
2. Защита от утечек информации [Электронный ресурс]. –Электронные данные. – Режим доступа: https://efsol.ru/promo/data-protect.html[. –](https://www.domo.com/learn/data-never-sleeps-7.%E2%80%93) Дата доступа: 30.03.2021.
3. DLP-система Zecurion для комплексной защиты от учетек конфиденциальной информации [Электронный ресурс] / Zecurion. – Электронные данные. – Режим доступа: https://www.zecurion.ru/solution/dlp/data-loss-prevention/[. –](https://www.domo.com/learn/data-never-sleeps-7.%E2%80%93) Дата доступа: 31.03.2021.
4. DLP-система InfoWatch Traffic Monitor [Электронный ресурс] – Электронные данные. – Режим доступа: https://www.infowatch.ru/products/traffic-monitor. – Дата доступа: 31.03.2021.
5. Обеспечение информационной безопасности - FalconGaze [Электронный ресурс] / FalconGaze. – Электронные данные. – Режим доступа: https://falcongaze.com/ru/. – Дата доступа: 01.04.2021.
6. СёрчИнформ КИБ [Электронный ресурс] / SearchInform. – Электронные данные. – Режим доступа: https://searchinform.ru/products/kib/. – Дата доступа: 02.04.2021.
7. Камер, Д. Сети TCP/IP, Том III. Разработка приложений типа клиент/сервер / Д. Э. Камер, Д. Л. Стивенс. – М.: Издательский дом «Вильямс», 2002. – 592 с.
8. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework на языке C# / Дж. Рихтер. – 4-е изд. – СПб.: Издательство «Питер», 2013. – 896 с.

# 

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Листинг кода

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Ведомость документов