Частное учреждение образования

«Колледж бизнеса и права»

ОБФУСКАТОР ИСХОДНОГО КОДА C#

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проекту по дисциплине

«Конструирование программ и языки программирования»

КП Т.691003

Руководитель проекта Е. В. Багласова

Выполнил В. В. Блинов

2019

Содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

КП Т.691003

Разраб.

*Блинов В.В.*

Провер.

Багласова Е.В.

Т. контр.

Н. контр.

Утверд.

Обфускатор исходного кода C# «SRCFU5C4T0R»

Лит.

Листов

36

КБП

[Введение 3](#_Toc11904744)

[1 Объектно-ориентированный анализ и проектирование системы 4](#_Toc11904745)

[1.1 Сущность задачи 4](#_Toc11904746)

[1.2 Проектирование модели 5](#_Toc11904747)

[2 Вычислительная система 9](#_Toc11904748)

[2.1 Требования к аппаратным и операционным ресурсам 9](#_Toc11904749)

[2.2 Инструменты разработки 9](#_Toc11904750)

[3 Проектирование задачи 11](#_Toc11904751)

[3.1 Требования к приложению 11](#_Toc11904752)

[3.2 Концептуальный прототип 11](#_Toc11904753)

[3.3 Организация данных 12](#_Toc11904754)

[3.4 Функции: логическая и физическая организация 13](#_Toc11904755)

[4 Описание программного средства 15](#_Toc11904756)

[4.1 Общие сведения 15](#_Toc11904757)

[4.2 Функциональное назначение 15](#_Toc11904758)

[4.3 Входные и выходные данные 15](#_Toc11904759)

[5 Методика испытаний 17](#_Toc11904760)

[5.1 Технические требования 17](#_Toc11904761)

[5.2 Функциональное тестирование 17](#_Toc11904762)

[6 Применение 19](#_Toc11904763)

[6.1 Назначение программы 19](#_Toc11904764)

[6.2 Условия применения 19](#_Toc11904765)

[Заключение 20](#_Toc11904766)

[Список информационных источников 21](#_Toc11904767)

[Приложение А](#_Toc11904768) [Текст программы 22](#_Toc11904770)

у

# Введение

Обфускацией программ называется такое эквивалентное преобразование программ, которое придает программе форму, затрудняющую понимание алгоритмов и структур данных, реализуемых программой, и препятствующую извлечению из текста программы определенной секретной информации, содержащейся в ней. Поскольку обфускация программ может найти широкое применение при решении многих задач криптографии и компьютерной безопасности, задаче оценки стойкости обфускации придается очень большое значение, начиная с самых первых работ в этой области. За последние два года написано свыше 70-ти статей по этой теме, что является показателем настоящий гонки между исследовательскими группами.

Давая пользователям доступ к установочным файлам программ, компании неизбежно раскрывают свои профессиональные секреты и наработки, и ничто не останавливает конкурентов от противоправного копирования и воровства чужих алгоритмов. Обратим внимание и на другой пример, это важные обновления (патчи), исправляющие ошибки в операционных системах. Почти мгновенно очередное обновление анализируется хакерами, они выявляют проблему, которую это обновление чинит, и атакуют пользователей, не успевших вовремя обновиться, пользователей.

Целью разработки проекта на тему «Обфускатор исходного кода C#» является создание программы, которая позволит пользователям (разработчикам) «запутывать» документы с исходным кодом, в проектах, написанных на Microsoft Visual Studio IDE.

Пояснительная записка к курсовому проекту состоит из шести разделов, содержащих необходимую информацию по организации и эксплуатации программного приложения.

В первом разделе «Объектно-ориентированный анализ и проектирование системы» раскрывается организационная сущность задачи, описывается предметная область и круг задач, которые должны быть автоматизированы. Описывается задача, перечисляются основные функции программы. Строится информационная модель, отражающая сущности задачи, их свойства и взаимосвязи.

Во втором разделе «Вычислительная система» перечисляются требования к аппаратному обеспечению и конфигурации компьютера, проводится характеристика операционной системы, обоснование выбранной среды для разработки приложения. Описываются новые возможности программы, а также ее отличия от предыдущих версий.

В третьем разделе «Проектирование задачи» описаны требования к приложению, строится концептуальный прототип системы меню, диалоговых окон и элементов управления.

В четвертом разделе «Описание программного средства» представлены общие сведения о программном средстве и его функциональном назначении.

В пятом разделе «Методика испытаний» описываются требования к техническим средствам для проведения испытаний, требования к характеристикам программы применительно к условиям эксплуатации, требования к информационной и программной совместимости. Представляются результаты функционального и полного тестирования.

Шестой раздел «Применение» предназначен для описания сведений о назначении программного средства и области его применения. В этом разделе приводится структура справочной системы, а также методика ее использования.

В заключении будет проанализировано созданное программное приложение, определена степень соответствия поставленной задачи и выполненной работы.

Приложение А будет содержать текст программы.

В графической части будут представлены диаграммы вариантов использования, классов, деятельности, последовательности и компонентов.

# 1 Объектно-ориентированный анализ и проектирование системы

## 1.1 Сущность задачи

Вероятно, задача обфускации была впервые упомянута (без явного употребления термина «обфускация») в 1976 году в основополагающей работе Диффи и Хеллмана. Желая проиллюстрировать концепцию шифрования с открытым ключом, они предложили следующую простую схему ее реализации. Выбирается произвольная криптосистема с секретным ключом, в процедуру шифрования вставляется секретный ключ, и затем инициализированная этим ключом программа шифрования запутывается так, чтобы извлечение из ее текста секретного ключа было очень трудной задачей. Таким образом, модифицированная процедура шифрования становится открытым ключом новой криптосистемы. Запутывание процедуры шифрования с целью предотвращения извлечения из ее текста секретного ключа является одним из возможных применений обфускации программ для решения некоторых задач криптографии и компьютерной безопасности.

В явном виде понятие обфускации программ было введено в 1997 году в работе Коллберга, Томборсона и Лоу. Авторы этой работы рассматривали обфускацию программ, в первую очередь, как средство защиты прав интеллектуальной собственности на алгоритмы, которые реализуются в программах с открытым кодом. Ими были предложены простейшие виды обфускирующих преобразований программ, проведена их систематическая классификация и прослежена взаимосвязь задачи обфускации программ с некоторыми известными задачами системного программирования.

Обфускация имеет под собой серьёзные теоретические основания. Важно не только выполнить основную задачу – запутать код, но и при этом не нанести серьёзного урона как скорости, так и функциональности приложения.

Цели обфускации:

* затруднение декомпиляции/отладки и изучения программ с целью обнаружения функциональности;
* затруднение декомпиляции проприетарных программ с целью предотвращения обратной разработки или обхода DRM и систем проверки лицензий;
* оптимизация программы с целью уменьшения размера работающего кода и (если используется некомпилируемый язык) ускорения работы. Демонстрация неочевидных возможностей языка и квалификации программиста (если производится вручную, а не инструментальными средствами).

Существует несколько разных видов обфускации в зависимости от того, какой именно код запутывается при ней. Для тех языков программирования, для которых программа представляется прямо в виде исходных текстов, используются специальные методы, делающие код нечитабельным для человека, но приемлемым для интерпретатора. Среди них - запись программы в одну строку, замена имён переменных и функций, вставка ничего не значащих комментариев. Для обфускации двоичного кода (как для исполняемых файлов, так и двоичного кода виртуальных машин .NET и Java) используются уже более серьёзные методы, в том числе и шифрование. Однако следует учитывать, что для двоичного кода обфускация способна серьёзно снизить скорость работы защищённых участков программы. Поэтому критичные ко времени выполнения куски кода, как правило, не имеют серьёзной криптографической защиты.

Обфускации может подвергаться не только программный код, но и другие виды данных: например, существуют специализированные методы обфускации HTML-страниц (особенно эффективно внедрение сценариев JavaScript'е, которые пишут текст динамически, и картинок с текстом). Обфускация HTML-страниц весьма полезна для спамеров, которые с её помощью обходят спам-фильтры.

Из аналогов можно выделить программу «Dotfuscator» – это инструмент, который выполняет сочетание обфускации, оптимизации, сжатия и усиления кода в приложениях .NET, Xamarin и Universal Windows Platform.

Исходя из сущности задачи можно выделить следующие методы, которые будут реализованы:

* обфускация проекта;
* выбор режима обфускации;
* выбор опции для обфускации;
* выбор файла для обфускации.

## 1.2 Проектирование модели

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций – диаграмм. В терминах языка UML определены следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма деятельности, диаграмма последовательности, диаграмма компонентов.

Перечень этих диаграмм представляет собой неотъемлемую часть графической нотации языка UML, сам процесс объектно-ориентированного программирования (ООП) неразрывно связан с процессом построения этих диаграмм. Совокупность построенных таким образом диаграмм содержит всю информацию, необходимую для реализации проекта сложной системы.

Диаграмма – граф специального вида, состоящий из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами.

Суть диаграммы вариантов использования состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы.

Актёр представляет собой внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует её функциональные возможности для достижения определённых целей или решения частных задач. При этом актёры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актёр может рассматриваться как некоторая отдельная роль относительно конкретного варианта использования.

К основным функциям разрабатываемой программы относятся следующие функции:

* обфускация проекта;
* выбор режима обфускации;
* выбор опции для обфускации;
* выбор файла для обфускации.

Диаграмма вариантов использования представлена на листе 1 графической части.

Центральное место в ООП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна. Схожая нотация применяется и для объектов — экземпляров класса, с тем различием, что к имени класса добавляется имя объекта, и вся надпись подчеркивается.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами кото­рого являются элементы типа «классификатор» и которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представлением таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

Класс в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линями на разделы или секции. Здесь указывается: имя класса, атрибуты и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельный класс может обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы классов дополняются атрибутами и операциями. Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Состав некоторых основных классов проекта представлен в таблице 2.1. Диаграмма классов представлена на листе 2 графической части.

Таблица 2.1 – Состав некоторых основных классов проекта

|  |  |  |
| --- | --- | --- |
| Имя класса | Имя поля | Методы |
| APIAnalyze | public SyntaxTree tree;  public CompilationUnitSyntax root;  static public ProjectId projId = ProjectId.CreateNewId();  int ecx = 0; | public Solution createSolution(string projName);  public SyntaxTree loadCode(string src);  public void printDeclareMethodsIdentifer();  public void printDeclareVariablesIdentifer();  public string[] getArrayOfDeclareMethods(string[] src);  public string[] getArrayOfDeclareMethodsIdentifer(string[] src);  public string[] getArrayOfDeclareVariables(string[] src);  public string[] getArrayOfDeclareVariablesIdentifer(string[] src);  public void printVarsInitAndOperations();  public async Task<string[]> getCallableMethods(string[] src, string pathToSolution);  public string[] getArrayOfVarsInitAndOperations(string[] src);  public string[] getListOfParameters(string[] src);  public string[] getDeclareMethodPrototype(string[] src); |
| Config | public static readonly string KEY\_LITE\_MODE = "--lite";  public static readonly string KEY\_NORMAL\_MODE = "--normal";  public static readonly string KEY\_RENAME\_VARS\_OPT = "--rv"; |  |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| Имя класса | Имя поля | Методы |
| Config | public static readonly string KEY\_RENAME\_NAMESPACES\_OPT = "--rn";  public static readonly string KEY\_RENAME\_CLASSES\_OPT = "--rc";  public static readonly string KEY\_RENAME\_METHODS\_OPT = "--rm";  public static string pathToOriginal;  public static string pathToObfuscated;  public static bool isRenameVars;  public static bool isRenameMethods;  public static bool isRenameNamespaces;  public static bool isRenameClasses;  public static bool isRenameMethodParams;  public static bool isEncryptStrings;  public static bool isMutate; |  |

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, переход в следующее состояние срабатывает только при завершении этой операции. Графически диаграмма деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому.

Диаграммы деятельности – частный случай диаграмм состояний. Основная цель использования таких диаграмм - визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Диаграмма деятельности представлена для метода переименования идентификаторов переменных на листе 3 графической части.

Для моделирования взаимодействия объектов в UML используется диаграмма последовательности. Диаграмма последовательности представлена на листе 4 графической части.

Все предыдущие диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления и оперировали понятиями, которые не имеют самостоятельного материального воплощения. Для описания реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели. Для физического представления модели используются диаграмма компонентов.

Диаграмма компонентов описывает объекты реального мира – компоненты программного обеспечения. Эта диаграмма позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

Диаграмма пакетов представлена на листе 5 графической части и содержит следующие компоненты:

* .NET 4.6 framework;
* решение «SRCFU5C4T0R»;
* документы примеров по использованию API ядра;
* проект «CLI.SRCFU5C4T0R»;
* проект содержащий юнит-тесты;
* библиотека «srclib»;
* файлы Kernel;
* файлы Obfuscation;
* файлы Renamers;
* файлы Helpers;
* файлы Utils;
* API ядра.

# 2 Вычислительная система

## 2.1 Требования к аппаратным и операционным ресурсам

Основными минимальными требованиями, выдвигаемыми к аппаратному обеспечению персонального компьютера, являются:

* процессор 1.4 МГц и выше;
* оперативная память 512 Мбайт и более;
* свободное место на диске 512 Мбайт;
* интегрированная видеокарта на 1024 Мбайт и более;
* монитор;
* мышь, клавиатура.

На компьютере должен быть установлен .NET Framework 4.6.1 или версией выше.

## 2.2 Инструменты разработки

Инструментами разработки для будущего программного приложения будут являться:

* операционная система Widows 10 PRO;
* программная среда разработки MS Visual Studio Community 2015;
* язык программирования C# 6.0;
* программа для построения UML диаграм Umlet;
* программа для построения диаграм Altova UModel 2019 (x64).

Windows – семейство [коммерческих](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D1%80%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (OC) корпорации [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), ориентированных на управление с помощью [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F). Изначально Windows была всего лишь графической программой-надстройкой, распространённой в 1980-х и 1990-х годах операционной системой [MS-DOS](https://ru.wikipedia.org/wiki/MS-DOS). Согласно данным ресурса [Net Applications](https://ru.wikipedia.org/wiki/Net_Applications), по состоянию на август 2014 года под управлением [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) семейства Windows работает около 88 % [персональных компьютеров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). Windows работает на [платформах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0) [x86](https://ru.wikipedia.org/wiki/X86), [x86-64](https://ru.wikipedia.org/wiki/X86-64), [IA-64](https://ru.wikipedia.org/wiki/IA-64) и [ARM](https://ru.wikipedia.org/wiki/ARM_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)). Существовали также версии для [DEC](https://ru.wikipedia.org/wiki/Digital_Equipment_Corporation) [Alpha](https://ru.wikipedia.org/wiki/DEC_Alpha), [MIPS](https://ru.wikipedia.org/wiki/MIPS_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)), [PowerPC](https://ru.wikipedia.org/wiki/PowerPC) и [SPARC](https://ru.wikipedia.org/wiki/SPARC). 10 [ ].

C# (произносится си шарп) – объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы, как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, WindowsPhone .NET Compact Framework и Silverlight. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual Source Safe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

# 3 Проектирование задачи

## 3.1 Требования к приложению

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* метод переименования пространств имен;
* метод переименования идентификаторов классов;
* метод переименования идентификаторов переменных;
* метод переименования идентификаторов методов;
* метод переименования параметров в методах;
* метод шифрования строковых констант;
* метод шифрования численных констант;
* метод перезаписи проекта на обфусцированный.

Требованием к организации входных данных является проект \*.csproj, содержащий информацию о документах.

Все компоненты системы должны иметь возможность резервирования, что повысит надёжность и позволит сохранять работоспособность системы и целостность данных при частичном выходе из строя программно-аппаратных средств.

Программное изделие должно иметь маркировку с обозначением товарного знака компании-разработчика, типа (наименования), номера версии, порядкового номера, даты изготовления и номера сертификата соответствия Госстандарта Беларуси (если таковой имеется).

Требованием к организации выходных данных программы является перезапись проекта \*.csproj вместе с обфусцированными исходными файлами проекта.

## 3.2 Концептуальный прототип

Концептуальный прототип состоит из описания внешнего командного интерфейса, а именно, входных аргументов командной строки. Структура команд управления представлена в таблице 3.1.

Таблица 3.1. – Структура команд управления

|  |  |
| --- | --- |
| Аргумент командной строки | Описание |
| --lite | Легкий режим обфускации |
| --normal | Средний режим обфускации |
| --rv | Опция переименование идентификаторов переменных |
| --rn | Опция переименование идентификаторов пространств имен |
| --rm | Опция переименования идентификаторов методов |
| --rp | Опция переименования идентификаторов параметров методов |
| --rc | Опция переименования идентификаторов классов |

## 3.3 Организация данных

Для удобства в программе был использован анализатор Roslyn [ ]. Структура данных для анализа представлена на рисунке 3.1.

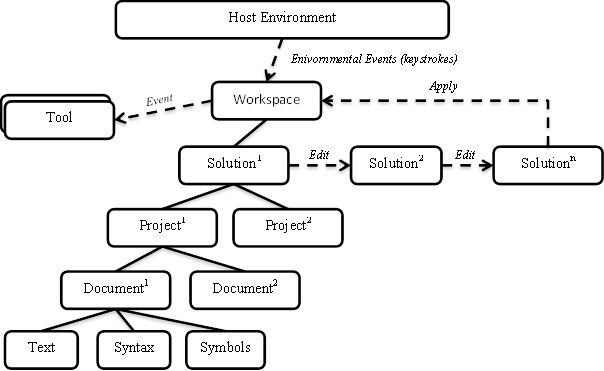


Рисунок 3.1 – Необходимая данные для анализа

Синтаксическое дерево является базовым элементом, необходимым для анализа кода. Именно по нему происходит перемещение в ходе анализа. Дерево строится на основе кода, приведённого в файле, из чего следует вывод, что каждый файл имеет своё синтаксическое дерево. Помимо этого, стоит учитывать тот факт, что синтаксическое дерево является неизменяемым. Нет, изменить его, конечно, можно, вызвав соответствующий метод, но результатом его работы будет новое синтаксическое дерево, а не изменённое старое.  
Например, для следующего кода:

class C

{

void M()

{ }

}

Синтаксическое дерево кода представлено на рисунке 3.2.

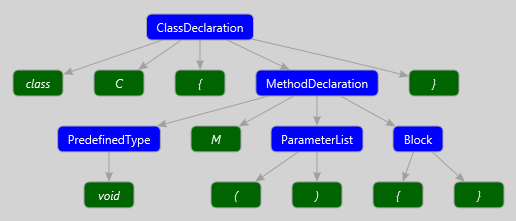


Рисунок 3.2 – Пример синтаксического дерева

Здесь синим цветом обозначены узлы дерева (Syntax nodes), а зелёным – лексемы (Syntax tokens). В синтаксическом дереве, которое строит Roslyn на основе программного кода, можно выделить 3 элемента:

* syntax nodes;
* syntax tokens;
* syntax trivia.

## 3.4 Функции: логическая и физическая организация

На основании диаграммы вариантов использования в программном приложении «SRCF5C4T0R» были реализованы следующие методы:

* метод переименования пространств имен;
* метод переименования идентификаторов классов;
* метод переименования идентификаторов переменных;
* метод переименования идентификаторов методов;
* метод переименования параметров в методах;
* метод шифрования строковых констант;
* метод шифрования численных констант;
* метод перезаписи проекта на обфусцированный.

Рассмотрим основные методы программы. При выборе опции переименования идентификаторов переменных, будет вызван соответствующий метод с именем «renameVarsIdentifier». Программный код метода переименования идентификаторов переменных представлен ниже:

public Solution renameVarsIdentifier(Solution solution) {

var projId = APIAnalyze.projId;

MSBuildWorkspace wspLoading = MSBuildWorkspace.Create();

var prjLoading = wspLoading.OpenProjectAsync(Config.pathToOriginal).Result;

Solution slnLoading = wspLoading.CurrentSolution;

foreach(var prjId in slnLoading.GetProjectDependencyGraph().GetTopologicallySortedProjects()) {

var prj = slnLoading.GetProject(prjId);

foreach(var doc in prj.Documents)

solution = solution.AddDocument(DocumentId.CreateNewId(projId), doc.Name, doc.GetTextAsync().Result);

}

var documents = solution.Projects.SelectMany(x => x.Documents).Select(x => x.Id).ToList();

Console.WriteLine("-------------------------------------------------------------------------------------");

foreach(var documentId in documents) {

List<VariableDeclarationSyntax> vars;

int i;

do {

var doc = solution.GetDocument(documentId);

var model = doc.GetSemanticModelAsync().Result;

var syntax = doc.GetSyntaxRootAsync().Result;

vars = syntax.DescendantNodes()

.OfType<VariableDeclarationSyntax>()

.Where(x => x.Variables.Count(z => !z.Identifier.ValueText.StartsWith("\_\_0x")) > 0)

.ToList();

for(i = 0; i < vars.Count; i++) {

foreach(var vr in vars[i].Variables) {

var symbol = model.GetDeclaredSymbol(vr);

var newName = "\_\_0x" + Utils.String.to\_Tiny(Utils.String.to\_SHA1(vr.Identifier.ValueText));

Console.WriteLine("Renaming variable: " + vr.Identifier.ValueText + " to " + newName + $" {vr.Kind()}");

solution = Renamer.RenameSymbolAsync(solution, symbol, newName, null).Result;

break;

}

break;

}

} while(i < vars.Count);

}

Console.WriteLine("-------------------------------------------------------------------------------------");

return (solution);

}

При выборе опции переименования идентификаторов классов будет вызван соответствующий метод с именем «renameClasses». Программный код метода переименования идентификаторов классов представлен ниже:

public Solution renameClasses(Solution solution) {

MSBuildWorkspace workspace = MSBuildWorkspace.Create();

var projId = APIAnalyze.projId;

MSBuildWorkspace wspLoading = MSBuildWorkspace.Create();

var prjLoading = wspLoading.OpenProjectAsync(Config.pathToOriginal).Result;

Solution slnLoading = wspLoading.CurrentSolution;

foreach(var prjId in slnLoading.GetProjectDependencyGraph().GetTopologicallySortedProjects()) {

var prj = slnLoading.GetProject(prjId);

foreach(var doc in prj.Documents)

solution = solution.AddDocument(DocumentId.CreateNewId(projId),doc.Name,doc.GetTextAsync().Result);

}

var documents = solution.Projects.SelectMany(x => x.Documents).Select(x => x.Id).ToList();

Console.WriteLine("-------------------------------------------------------------------------------------");

foreach(var documentId in documents) {

while(true) {

var doc = solution.GetDocument(documentId);

var model = doc.GetSemanticModelAsync().Result;

var syntax = doc.GetSyntaxRootAsync().Result;

var classes = syntax.DescendantNodes()

.OfType<ClassDeclarationSyntax>()

.Where(x => !x.Identifier.ValueText.StartsWith("\_\_0x"))

.ToList();

var cl = classes.FirstOrDefault();

if(cl == null)

break;

var symbol = model.GetDeclaredSymbol(cl);

var newName = "\_\_0x" + Utils.String.to\_Tiny(Utils.String.to\_SHA1(cl.Identifier.ValueText));

Console.WriteLine("Renaming class: " + cl.Identifier.ValueText + " to " + newName);

solution = Renamer.RenameSymbolAsync(solution, symbol, newName, null).Result;

}

}

Console.WriteLine("-------------------------------------------------------------------------------------");

return solution;

}

Весь программный код представлен в Приложении А.

# 4 Описание программного средства

## 4.1 Общие сведения

Данное приложение именуется как «SRCFU5C4T0R». После установки папка с приложением будет занимать 19.6 Мбайт дисковой памяти.

Состав программы:

* «SRCFU5C4T0R.CLI.exe» – главный файл CLI приложения;
* «BlackLizardPlanetAPI.dll» – библиотека для шифрования;
* «Microsoft.CodeAnalysis.CSharp.dll» – библиотека roslyn;
* «Microsoft.CodeAnalysis.CSharp.Workspaces.dll » – библиотека roslyn;
* «Microsoft.CodeAnalysis.dll » – библиотека roslyn;
* «Microsoft.CodeAnalysis.Workspaces.Desktop.dll» – библиотека roslyn;
* «Microsoft.CodeAnalysis.Workspaces.dll» – библиотека roslyn;
* «SRCFU5C4T0R.CLI.exe.config» – конфигурационный файл;
* «SRCFU5C4T0R.Core.dll» – библиотека ядра «SRCFU5C4T0R»;
* «srclib.dll» – вспомогательная библиотека для расшифровки;
* «System.Collections.Immutable.dll» – библиотека для неизменяемых коллекций;
* «System.Composition.AttributedModel.dll» – библиотека предоставляет рефлексию данных для поддержки конвенционных моделей;
* «System.Composition.Convention.dll» – библиотека для Convention class;
* «System.Composition.Hosting.dll» – библиотека для контейнера композиций;
* «System.Composition.Runtime.dll» – библиотека для рантайма контейнера композиций;
* «System.Composition.TypedParts.dll» – библиотека для контейнера композиций;
* «System.Reflection.Metadata.dll» – библиотека для метаданных.

Разработанное приложение может работать на любом компьютере, где установлена ОС MS Windows с .NET Framework 4.6.1.

Структура проекта содержит все файлы и папки, необходимые компиляции объектного файла.

## 4.2 Функциональное назначение

Созданное приложение позволит пользователям «запутывать» документы проектов \*.csproj с целью скрытия важных участков кода, DRM, систем лицензирования и т.п.

Для реализации данного проекта необходимо определить следующие методы и объекты:

* метод переименования пространств имен;
* метод переименования идентификаторов классов;
* метод переименования идентификаторов переменных;
* метод переименования идентификаторов методов;
* метод переименования параметров в методах;
* метод шифрования строковых констант;
* метод шифрования численных констант;
* метод перезаписи проекта на обфусцированный.

## 4.3 Входные и выходные данные

Входными данными являются проект MS Visual Studio C# (\*.csproj) и режим или опция обфускации для выбранного проекта.

Выходными данными являются проект MS Visual Studio C# (\*.csproj) с обфусцированными документами исходного кода C# (\*.cs).

# 5 Методика испытаний

## 5.1 Технические требования

Для полноценного функционирования разработанного программного приложения, качественного выполнения всех поставленных задач, требуется ПК следующей конфигурации:

Таблица 5.1– Конфигурация ПК

|  |  |
| --- | --- |
| Процессор | AMD/Intel2.2 Гц+ |
| Оперативная память | 512 Мб + |
| Дисковой накопитель | 512 Мб |
| Операционная система | OS Windows |

## 5.2 Функциональное тестирование

В процессе написания программного продукта необходимо производить тестирование на правильность работы приложения. Одной из основных задач тестирования является устранение ошибок, происходящих при вводе данных.

В таблице 5.1 представлены тест-кейсы, подготовленные для проведения функционального тестирования.

Таблица 5.1– Тест-кейсы для проведения функционального тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № тест-  кейса | Модуль/  Функция | Шаги воспроизведения | Результат |
| 001 | Легкий режим | 1. Запустить приложение указав входными аргументами командной строки:  * путь к проекту; * легкий режим обфускации. | Ожидаемый результат: измененный проект с обфусцированными документами C# |
| Фактический результат: изменный проект с обфусцированнымиданными. Пример обфусцированного документа:  namespace \_\_0x9ᵃ26848ᵇ02ᵇᶠ69ᵉ29ᶠ465ᵉᵉ7816ᵈᵇ02ᵃ8747ᶜᶜ2ᶠᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ {  class \_\_0xᶠ10ᵉ2821ᵇᵇᵇᵉᵃ527ᵉᵃ02200352313ᵇᶜ059445190ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ {  public int \_\_0xᵈᵃ23614ᵉ02469ᵃ0ᵈ7ᶜ7ᵇᵈ1ᵇᵈᵃᵇ5ᶜ9ᶜ474ᵇ1904ᵈᶜᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ = 5;  public static void Main() {  int \_\_0x86ᶠ7ᵉ437ᶠᵃᵃ5ᵃ7ᶠᶜᵉ15ᵈ1ᵈᵈᶜᵇ9ᵉᵃᵉᵃᵉᵃ377667ᵇ8ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ = 5;  int \_\_0xᵉ9ᵈ71ᶠ5ᵉᵉ7ᶜ92ᵈ6ᵈᶜ9ᵉ92ᶠᶠᵈᵃᵈ17ᵇ8ᵇᵈ49418ᶠ98ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ = 3; |

Продолжение таблицы 5.1.

|  |  |  |  |
| --- | --- | --- | --- |
| № тест-  кейса | Модуль/  Функция | Шаги воспроизведения | Результат |
|  |  |  | int \_\_0x84ᵃ516841ᵇᵃ77ᵃ5ᵇ4648ᵈᵉ2ᶜᵈ0ᵈᶠᶜᵇ30ᵉᵃ46ᵈᵇᵇ4ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ = \_\_0x86ᶠ7ᵉ437ᶠᵃᵃ5ᵃ7ᶠᶜᵉ15ᵈ1ᵈᵈᶜᵇ9ᵉᵃᵉᵃᵉᵃ377667ᵇ8ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ + \_\_0xᵉ9ᵈ71ᶠ5ᵉᵉ7ᶜ92ᵈ6ᵈᶜ9ᵉ92ᶠᶠᵈᵃᵈ17ᵇ8ᵇᵈ49418ᶠ98ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ;  }  }  } |
| 002 | Средний режим обфускации | 1. Запустить приложение указав входными аргументами командной строки:  * путь к проекту; * средний режим обфускации. | Ожидаемый результат: измененный проект с обфусцированными документами C# |
| Фактический результат: изменный проект с обфусцированными данными. |
| 003 | Опция переименования идентификаторов переменных | 1. Запустить приложение указав входными аргументами командной строки:  * путь к проекту; * опцию переименования идентификаторов переменных. | Ожидаемый результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы переменных |
| Фактический результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы переменных |
| 004 | Опция переименования идентификаторов методов | 1. Запустить приложение указав входными аргументами командной строки:  * путь к проекту; * опцию переименования идентификаторов методов. | Ожидаемый результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы методов |
| Фактический результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы методов |
| 005 | Опция переименования идентификаторов пространств имен | 1. Запустить приложение указав входными аргументами командной строки:  * путь к проекту; * опцию переименования идентификаторов пространств имен. | Ожидаемый результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы пространств имен |
| Фактический результат: измененный проект с обфусцированными документами в которых «запутаны» идентификаторы пространств имен |

# 6 Применение

## 6.1 Назначение программы

Созданное приложение позволит пользователям «запутывать» документы проектов \*.csproj с целью скрытия важных участков кода, DRM, систем лицензирования и т.п.

Главная цель процесса обфускации заключается в том, чтобы запутать программный код и скрыть в нем логические связи, т.е. изменить его так, чтобы он был сложен для изучения посторонними лицами. Однако одна лишь обфускация не обеспечивает эффективной защиты программ, так как она не предотвращает незаконного использования программного продукта. Поэтому обфускацию принято использовать совместно с другими методами защиты.

## 6.2 Условия применения

Для применения данного программного средства необходимы следующие технические требования:

* процессор Intel Core 2 Duo или выше;
* минимальный объем оперативной памяти — 512 Мбайт;
* операционная система MS Windows Vista (Windows NT 6.0) и выше;
* рекомендуется монитор типа HDMI;
* устройство для чтения дисков;
* клавиатура;
* мышь.

# Заключение

В рамках курсового проектирования по предмету «Конструирование программ и языки программирования» было разработано программное приложение «SRCFU5C4T0R.exe», позволяющее «запутывать» проекты MS Visual Studio C# (\*csproj).

Программное средство имеет ряд достоинств: простой интерфейс, небольшой объем памяти, занимаемый приложением на различных носителях информации, удобно для быстрой обработки информации.

В программе реализованы такие задачи, как:

* обфускация проекта;
* выбор режима обфускации;
* выбор опции для обфускации;
* выбор файла для обфускации.

В процессе курсового проектирования программное приложение прошло функциональное и полное тестирование, которые доказали корректность выполнения всех определенных функций, отсутствие сбоев и неадекватных реакций со стороны программы.

В процессе разработки программы использовался в большом объеме тематический материал и исследована предметная область разработки аналогичных программных средств.

Также использовался материал по программированию по разделам «записи», «файлы», «массивы», что способствовало закреплению наработанных навыков и умений в этих интересных областях знаний

# Список информационных источников

1. Багласова, Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2017. – 30 c.
2. Багласова, Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2017. – 30 c.
3. Введение в Roslyn. Использование для разработки инструментов статического анализа [Электронный ресурс] : статья. – Режим доступа https://habr.com/ru/company/pvs-studio/blog/301204/. – Дата доступа : 21.05.2019.
4. Что такое обфускатор и обфускация? [Электронный ресурс]. – Режим доступа : https://www.kv.by/archive/index2008071108.htm. – Дата доступа : 21.05.2019.
5. Обфускация и деобфускация программ [Электронный ресурс]. – Режим доступа : <https://studme.org/128267/informatika/obfuskatsiya_deobfuskatsiya_programm>. – Дата доступа : 21.05.2019

# Приложение А

# (обязательное)

# Текст программы

//SRCFU5C4T0R.Core/Kernel/API/APIIAnalyze.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.CodeAnalysis;

using Microsoft.CodeAnalysis.CSharp;

using Microsoft.CodeAnalysis.CSharp.Syntax;

using Microsoft.CodeAnalysis.MSBuild;

namespace SRCFU5C4T0R.Core.Kernel.API {

/// <summary>

/// Class that implements the interface IAnalyze

/// </summary>

internal class APIAnalyze : IAnalyze {

public SyntaxTree tree;

public CompilationUnitSyntax root;

static public ProjectId projId = ProjectId.CreateNewId();

//TODO: UI

int ecx = 0; //HACK:

/// <summary>

/// Implementation of method for create solution in workspace

/// </summary>

/// <param name="projName">Project name</param>

/// <returns>Created solution</returns>

public Solution createSolution(string projName) {

MSBuildWorkspace workspace = MSBuildWorkspace.Create();

//var projId = ProjectId.CreateNewId();

var versionStamp = VersionStamp.Create();

var projectInfo = ProjectInfo.Create(projId, versionStamp, projName, projName, LanguageNames.CSharp);

return workspace.CurrentSolution.AddProject(projectInfo);

}

/// <summary>

/// Implementation of method for load C# code from string variable

/// </summary>

/// <param name="src"></param>

/// <returns>Loaded tree</returns>

public SyntaxTree loadCode(string src) {

tree = CSharpSyntaxTree.ParseText(src);

return (tree);

}

/// <summary>

/// Print all names of methods in loaded source code

/// </summary>

public void printDeclareMethodsIdentifer() {

Console.WriteLine("Name of declare methods: ");

IEnumerable<MethodDeclarationSyntax> dst = tree.GetRoot().DescendantNodes().OfType<MethodDeclarationSyntax>();

foreach (MethodDeclarationSyntax method in dst) {

SyntaxToken id = method.Identifier;

Console.WriteLine("{0}", id);

}

}

/// <summary>

/// Print all names of variables in loaded source code

/// </summary>

public void printDeclareVariablesIdentifer() {

root = (CompilationUnitSyntax)tree.GetRoot();

var nameOfVars = root.DescendantNodes().OfType<VariableDeclarationSyntax>();

foreach (var vars in nameOfVars)

Console.WriteLine(vars.Variables.First().Identifier.Value);

Console.Write("\r\n");

}

/// <summary>

/// Get 1-rank string array of methods (one method = one element of array)

/// </summary>

/// <param name="src">Input string array which will be modified</param>

/// <returns>Array of methods</returns>

public string[] getArrayOfDeclareMethods(string[] src) {

IEnumerable<MethodDeclarationSyntax> dst = tree.GetRoot().DescendantNodes().OfType<MethodDeclarationSyntax>();

for (UInt16 i = 0; i < dst.Count(); ++i)

src[i] = dst.ElementAt(i).ToString();

return (src);

}

/// <summary>

/// Write all methods in 1-rank string array

/// </summary>

/// <param name="src">Input 1-rank string array</param>

/// <returns>1-rank string array with all name of methods</returns>

public string[] getArrayOfDeclareMethodsIdentifer(string[] src) {

ecx = 0;

IEnumerable<MethodDeclarationSyntax> dst = tree.GetRoot().DescendantNodes().OfType<MethodDeclarationSyntax>();

foreach(MethodDeclarationSyntax method in dst) {

SyntaxToken id = method.Identifier;

src[ecx] = id.ToString();

ecx++;

}

return (src);

}

/// <summary>

/// Method which write all name of variables in 1-rank string array

/// </summary>

/// <param name="src">Input 1-rank string array, which will be modified and store name of declare variables</param>

/// <returns>1-rank modified string array</returns>

public string[] getArrayOfDeclareVariables(string[] src) {

root = (CompilationUnitSyntax)tree.GetRoot();

var varDeclorations = root.DescendantNodes().OfType<VariableDeclarationSyntax>();

for (UInt16 i = 0; i < varDeclorations.Count(); ++i)

src[i] = varDeclorations.ElementAt(i).ToString();

return (src);

}

/// <summary>

/// Method which write all declare variables identificators in 1-rank string array

/// </summary>

/// <param name="src">Input 1-rank string array, which will be modified and store name of declare variables</param>

/// <returns>1-rank modified string array</returns>

public string[] getArrayOfDeclareVariablesIdentifer(string[] src) {

ecx = 0;

root = (CompilationUnitSyntax)tree.GetRoot(); //HACK: f1x th15, 0fc m4ny 1n1t14l

var nameOfVars = root.DescendantNodes().OfType<VariableDeclarationSyntax>();

foreach (var vars in nameOfVars) {

src[ecx] = vars.Variables.First().Identifier.Value.ToString();

ecx++;

}

return (src);

}

/// <summary>

/// Print all variables initializers and right hand side of operations in source code

/// </summary>

public void printVarsInitAndOperations() {

root = (CompilationUnitSyntax)tree.GetRoot();

var declareVarsExpr = root.DescendantNodes().OfType<VariableDeclarationSyntax>();

var assignmentVarsExpr = root.DescendantNodes().OfType<AssignmentExpressionSyntax>();

foreach (var expr in declareVarsExpr)

Console.WriteLine(expr.Variables.Last().Initializer.Value);

foreach(var expr in assignmentVarsExpr)

Console.WriteLine(expr.Right);

}

/// <summary>

/// Write to array all methods which will be called

/// </summary>

/// <param name="src">Array which will be storing all callable methods</param>

/// <param name="pathToSolution">Path to your solution in which you want to get callable methods</param>

/// <returns>String array which contains all callable methods in source code</returns>

public async Task<string[]> getCallableMethods(string[] src, string pathToSolution) {

var workspace = MSBuildWorkspace.Create();

int ecx = 0;

var solution = await workspace.OpenSolutionAsync(pathToSolution);

var project = solution.Projects.Single();

var compilation = await project.GetCompilationAsync();

foreach(var syntaxTree in compilation.SyntaxTrees) {

var root = await syntaxTree.GetRootAsync();

var model = compilation.GetSemanticModel(syntaxTree);

var invocations = root.DescendantNodes().OfType<InvocationExpressionSyntax>();

foreach(var invocation in invocations) {

var expression = invocation.Expression;

src[ecx] += invocation + "\n";

ecx++;

}

}

return src;

}

/// <summary>

/// Write to array all variables initializers and operations

/// </summary>

/// <param name="src">Array which will be modified</param>

/// <returns>String array which contains all variables initializers and right hand side of operation in source code</returns>

public string[] getArrayOfVarsInitAndOperations(string[] src) {

ecx = 0;

root = (CompilationUnitSyntax)tree.GetRoot();

var declareVarsExpr = root.DescendantNodes().OfType<VariableDeclarationSyntax>();

var assignmentVarsExpr = root.DescendantNodes().OfType<AssignmentExpressionSyntax>();

foreach (var vars in declareVarsExpr) {

if (vars.Variables.Last().Initializer == null) continue;

src[ecx] = vars.Variables.Last().Initializer.Value.ToString();

ecx++;

}

foreach(var expr in assignmentVarsExpr) {

src[ecx] = expr.Right.ToString();

ecx++;

}

return (src);

}

/// <summary>

/// Get all entry point parameters

/// </summary>

/// <param name="src">Array which you want to write parameters of entry point</param>

/// <returns>Modified string array</returns>

public string[] getListOfParameters(string[] src) {

ecx = 0;

root = (CompilationUnitSyntax)tree.GetRoot();

IEnumerable<MethodDeclarationSyntax> dst = tree.GetRoot().DescendantNodes().OfType<MethodDeclarationSyntax>();

foreach (MethodDeclarationSyntax method in dst) {

ParameterListSyntax id = method.ParameterList;

src[ecx] = id.ToString();

ecx++;

}

return (src);

}

public string[] getDeclareMethodPrototype(string[] src) {

root = (CompilationUnitSyntax)tree.GetRoot();

IEnumerable<MethodDeclarationSyntax> ty = tree.GetRoot().DescendantNodes().OfType<MethodDeclarationSyntax>();

foreach (MethodDeclarationSyntax method in ty) {

// SyntaxTokenList id = method.;

// Console.WriteLine("{0} ", id);

}

//TODO

return (src);

}

}

}

//SRCFU5C4T0R.Core/Kernel/API/IAnalyze.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using Microsoft.CodeAnalysis;

using Microsoft.CodeAnalysis.CSharp;

using Microsoft.CodeAnalysis.CSharp.Syntax;

using Microsoft.CodeAnalysis.FindSymbols;

using Microsoft.CodeAnalysis.Rename;

using System.Threading.Tasks;

namespace SRCFU5C4T0R.Core.Kernel.API {

/// <summary>

/// Interface for load and analyze C# code

/// </summary>

interface IAnalyze {

/// <summary>

/// Create solution in workspace

/// </summary>

/// <param name="projName">Project name</param>

/// <returns>Created solution</returns>

Solution createSolution(string projName);

/// <summary>

/// Method for load C# code from string variable

/// </summary>

/// <returns>Loaded SyntaxTree</returns>

//HACK: Load Code ;(

SyntaxTree loadCode(string src); //TODO: add API with documents/solutions/workspace/MSBuild/Adhoc/

/// <summary>

/// Print all names of methods in loaded source code

/// </summary>

void printDeclareMethodsIdentifer();

/// <summary>

/// Print all name of declare variables in loaded sorce code

/// </summary>

void printDeclareVariablesIdentifer();

/// <summary>

/// Print all variables expressions

/// </summary>

void printVarsInitAndOperations();

/// <summary>

/// Get all callable methods

/// <param name="src">Array which will be storing all callable methods</param>

/// <param name="pathToSolution">Path to your solution in which you want to get callable methods</param>

/// </summary>

Task<string[]> getCallableMethods(string[] src, string pathToSolution);

/// <summary>

/// Get 1 rank string-array with name of declare methods

/// </summary>

/// <returns>1 rank string-array</returns>

string[] getArrayOfDeclareMethods(string[] src);

/// <summary>

/// Method which write in your input array name of methods

/// </summary>

/// <param name="src">Input string array</param>

/// <returns>String array</returns>

string[] getArrayOfDeclareMethodsIdentifer(string[] src);

/// <summary>

/// Write in your input array all names of declare variables

/// </summary>

/// <param name="src">Array which you want to write names of declare variables</param>

/// <returns>Modified string array</returns>

string[] getArrayOfDeclareVariables(string[] src);

/// <summary>

/// Get all declare variables identificators

/// </summary>

/// <param name="src">Array which you want to write all identificators</param>

/// <returns>String array with identidicators</returns>

string[] getArrayOfDeclareVariablesIdentifer(string[] src);

/// <summary>

/// Get all variables expressions

/// </summary>

/// <param name="src">Array which you want to modifie</param>

/// <returns>Modified array</returns>

string[] getArrayOfVarsInitAndOperations(string[] src);

/// <summary>

/// Get all ecntry point parameters

/// </summary>

/// <param name="src">Array which you want to write parameters of entry point</param>

/// <returns>Modified string array</returns>

string[] getListOfParameters(string[] src);

/// <summary>

/// Get prototype of decalre method

/// </summary>

/// <param name="src">Input array for writing output</param>

/// <returns>1-rank string array</returns>

string[] getDeclareMethodPrototype(string[] src);

}

}

//SRCFU5C3T0R.Core/Obfuscation/EncryptStrings.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.CodeAnalysis;

using Microsoft.CodeAnalysis.CSharp;

using Microsoft.CodeAnalysis.CSharp.Syntax;

using Microsoft.CodeAnalysis.Editing;

using Microsoft.CodeAnalysis.MSBuild;

using Microsoft.CodeAnalysis.Rename;

using SRCFU5C4T0R.Core.Kernel.API;

using SRCFU5C4T0R.Core.Obfuscation;

using srclib.Helpers;

namespace SRCFU5C4T0R.Core.Obfuscation {

/// <summary>

/// This class provide to encrypt string

/// </summary>

class EncryptStrings {

public Solution encryptStrings(Solution solution) {

var projId = APIAnalyze.projId;

MSBuildWorkspace wspLoading = MSBuildWorkspace.Create();

var prjLoading = wspLoading.OpenProjectAsync(Config.pathToOriginal).Result;

Solution slnLoading = wspLoading.CurrentSolution;

foreach(var prjId in slnLoading.GetProjectDependencyGraph().GetTopologicallySortedProjects()) {

var prj = slnLoading.GetProject(prjId);

foreach(var doc in prj.Documents)

solution = solution.AddDocument(DocumentId.CreateNewId(projId), doc.Name, doc.GetTextAsync().Result);

}

var documents = solution.Projects.SelectMany(x => x.Documents).Select(x => x.Id).ToList();

Console.WriteLine("-------------------------------------------------------------------------------------");

foreach(var documentId in documents) {

List<LocalDeclarationStatementSyntax> vars;

List<InvocationExpressionSyntax> callable;

int i, z; // !< counters

do {

var doc = solution.GetDocument(documentId);

var editor = DocumentEditor.CreateAsync(doc).Result;

var model = doc.GetSemanticModelAsync().Result;

var syntax = doc.GetSyntaxRootAsync().Result;

vars = syntax.DescendantNodes()

.OfType<LocalDeclarationStatementSyntax>()

.ToList();

callable = syntax.DescendantNodes()

.OfType<InvocationExpressionSyntax>()

.ToList();

i = 0;

foreach (var localDeclarationStatementSyntax in vars) {

foreach (VariableDeclaratorSyntax variable in localDeclarationStatementSyntax.Declaration.Variables) {

if (variable.Initializer == null) continue;

var stringKind = variable.Initializer.Value.Kind();

if (stringKind == SyntaxKind.StringLiteralExpression) {

string formattedInitializer = Utils.String.delete\_quotes(variable.Initializer.Value.ToString()); // delete quotes from Initializer e.g. "value" to value

var newVariable = SyntaxFactory.ParseStatement($"string {variable.Identifier.ValueText} = a1f95c1bb38e78c13308.\_\_0xᵉ7ᵉ371ᶠ3ᶠ8ᵃᵃ56895ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ362ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ\_(\"{a1f95c1bb38e78c13308.\_\_0x3442496ᵇ96ᵈᵈ01591ᵃ8ᶜᵈ44ᵇ1ᵉᵉᶜ1368ᵃᵇ728ᵃᵇᵃᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ\_(formattedInitializer)}\");");

newVariable.NormalizeWhitespace();

editor.ReplaceNode(variable, newVariable);

Console.WriteLine($"Key: {variable.Identifier.Value} Value:{variable.Initializer.Value}");

}

}

doc = editor.GetChangedDocument();

Console.WriteLine(doc.GetSyntaxTreeAsync().Result);

i++;

}

z = 0;

foreach (var invocationExpressionSyntax in callable) {

if (invocationExpressionSyntax.ArgumentList.Arguments.Any(x =>

x.Expression.Kind() == SyntaxKind.StringLiteralExpression)) {

var stringList = new List<string>();

for (int x = 0; x < invocationExpressionSyntax.ArgumentList.Arguments.Count(); x++)

if (invocationExpressionSyntax.ArgumentList.Arguments[x].Expression.Kind() == SyntaxKind.StringLiteralExpression)

stringList.Add("a1f95c1bb38e78c13308.\_\_0xᵉ7ᵉ371ᶠ3ᶠ8ᵃᵃ56895ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ362ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ\_(\"" + a1f95c1bb38e78c13308.\_\_0x3442496ᵇ96ᵈᵈ01591ᵃ8ᶜᵈ44ᵇ1ᵉᵉᶜ1368ᵃᵇ728ᵃᵇᵃᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ\_(invocationExpressionSyntax.ArgumentList.Arguments[x].Expression.GetFirstToken().ValueText) + "\")");

else

stringList.Add(invocationExpressionSyntax.ArgumentList.Arguments[x].Expression

.GetFirstToken().ValueText);

SeparatedSyntaxList<ArgumentSyntax> arguments = new SeparatedSyntaxList<ArgumentSyntax>().AddRange

(new ArgumentSyntax[]

{

SyntaxFactory.Argument(SyntaxFactory.ParseExpression($"{string.Join(",", stringList)}")),

}

);

var newMethodWithStringObfuscation =

SyntaxFactory.InvocationExpression(SyntaxFactory.IdentifierName(invocationExpressionSyntax.Expression.ToString()))

.WithArgumentList(SyntaxFactory.ArgumentList().WithOpenParenToken(SyntaxFactory.Token(SyntaxKind.OpenParenToken))

.WithArguments(arguments).WithCloseParenToken(SyntaxFactory.Token(SyntaxKind.CloseParenToken)));

Console.WriteLine($"Replacing values for method {invocationExpressionSyntax.Expression.GetFirstToken().ValueText}");

editor.ReplaceNode(invocationExpressionSyntax, newMethodWithStringObfuscation);

}

z++;

}

File.WriteAllText(Config.pathToObfuscated + doc.Name, editor.GetChangedRoot().SyntaxTree.ToString()); // TODO:

} while(i < vars.Count && z < callable.Count);

}

Console.WriteLine("-------------------------------------------------------------------------------------");

return solution;

}

}

}

//SRCFU5C4T0R.Core/Obfuscation/NumProtection.cs

using System;

using System.CodeDom;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.CodeAnalysis;

using Microsoft.CodeAnalysis.MSBuild;

using Microsoft.CodeAnalysis.CSharp;

using Microsoft.CodeAnalysis.CSharp.Syntax;

using Microsoft.CodeAnalysis.Editing;

using SRCFU5C4T0R.Core.Kernel.API;

using srclib.Helpers;

namespace SRCFU5C4T0R.Core.Obfuscation {

/// <summary>

/// This class provide to mutation expressions

/// </summary>

class NumProtection {

private FieldDeclarationSyntax createNewField(FieldDeclarationSyntax field, VariableDeclaratorSyntax fld) {

double src\_initializer = Convert.ToDouble(field.Declaration.Variables.Last().Initializer.Value.ToString());

string encrypted\_arg =

Convert.ToString(a1f95c1bb38e78c13308

.\_\_0xᵉ7ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ362ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ(

src\_initializer));

FieldDeclarationSyntax tmp = SyntaxFactory.FieldDeclaration(SyntaxFactory.VariableDeclaration(

SyntaxFactory.ParseTypeName(field.Declaration.Type.ToString()),

SyntaxFactory.SeparatedList(new[] {

SyntaxFactory.VariableDeclarator(

SyntaxFactory.Identifier(fld.Identifier.ValueText), null,

SyntaxFactory.EqualsValueClause(SyntaxFactory.ParseExpression( (src\_initializer % 1 > 0) ?

$"a1f95c1bb38e78c13308.\_\_0xᵉ1ᵉ333ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ364ᵈᵉᵃᵈᶜᵒᵈᵉᵈ" +

$"ᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ({ encrypted\_arg })"

: "a1f95c1bb38e78c13308.\_\_0xᵉ1488ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ("

+ encrypted\_arg + ")")

)

)

})

));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Adding modifiers (private, readonly, etc.) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (field.Modifiers.Count > 0)

foreach (var modifer in field.Modifiers)

tmp = tmp.AddModifiers(modifer);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return tmp;

}

public Solution confuseExpressions(Solution sln) {

var projId = APIAnalyze.projId;

MSBuildWorkspace wspLoading = MSBuildWorkspace.Create();

var prjLoading = wspLoading.OpenProjectAsync(Config.pathToOriginal).Result;

Solution slnLoading = wspLoading.CurrentSolution;

foreach (var prjId in slnLoading.GetProjectDependencyGraph().GetTopologicallySortedProjects()) {

var prj = slnLoading.GetProject(prjId);

foreach (var doc in prj.Documents)

sln = sln.AddDocument(DocumentId.CreateNewId(projId), doc.Name, doc.GetTextAsync().Result);

}

var documents = sln.Projects.SelectMany(x => x.Documents).Select(x => x.Id).ToList();

Console.WriteLine($"Count of documents founded is: { documents.Count }");

foreach (var documentId in documents) {

List<LocalDeclarationStatementSyntax> vars;

List<FieldDeclarationSyntax> fields;

List<AssignmentExpressionSyntax> expr; //TODO: expressions

Console.WriteLine($"Current document name is: { sln.GetDocument(documentId).Name }");

var current\_doc = sln.GetDocument(documentId);

var editor = DocumentEditor.CreateAsync(current\_doc).Result;

var syntax = current\_doc.GetSyntaxRootAsync().Result;

vars = syntax.DescendantNodes().OfType<LocalDeclarationStatementSyntax>()

.Where(x => x.Declaration.Variables.Last().Initializer != null &&

x.Declaration.Variables.Last().Initializer.Value.Kind() == SyntaxKind.NumericLiteralExpression)

.ToList();

fields = syntax.DescendantNodes().OfType<FieldDeclarationSyntax>()

.Where(x => x.Declaration.Variables.Last().Initializer != null &&

x.Declaration.Variables.Last().Initializer.Value.Kind() == SyntaxKind.NumericLiteralExpression)

.ToList();

expr = syntax.DescendantNodes().OfType<AssignmentExpressionSyntax>().ToList();

if (expr.Count > 0) {

Console.WriteLine($"[MUTATION] : Count of founded expressions is: {expr.Count}");

foreach (var current\_expr in expr) {

foreach (var expr\_node in current\_expr.Right.ChildNodes()) {

if (expr\_node.Kind() == SyntaxKind.NumericLiteralExpression) {

double src\_token = Convert.ToDouble(expr\_node.ToString());

double encrypted\_arg = a1f95c1bb38e78c13308

.\_\_0xᵉ7ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ362ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ(src\_token);

string formatted\_node = (src\_token % 1 > 0)

? "a1f95c1bb38e78c13308.\_\_0xᵉ1ᵉ333ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ364ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ("

+ encrypted\_arg + ")"

: "a1f95c1bb38e78c13308.\_\_0xᵉ1488ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ("

+ encrypted\_arg + ")";

var new\_node = SyntaxFactory.ParseExpression($"{formatted\_node}");

editor.ReplaceNode(expr\_node, new\_node);

}

}

}

}

foreach(var field in fields) {

foreach(var fld in field.Declaration.Variables) {

Console.WriteLine($"[MUTATION]: founded field: { field }");

FieldDeclarationSyntax newField = createNewField(field, fld);

Console.WriteLine($"[MUTATION]: new field is: { newField }");

editor.ReplaceNode(fld, newField);

}

current\_doc = editor.GetChangedDocument();

}

foreach(var var\_declorator in vars) {

foreach(var vr in var\_declorator.Declaration.Variables) {

if(vr.Initializer == null) continue;

double src\_initializer = Convert.ToDouble(vr.Initializer.Value.ToString());

string encrypted\_arg =

Convert.ToString(a1f95c1bb38e78c13308

.\_\_0xᵉ7ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ362ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ(

src\_initializer));

string formattedInitializer = (src\_initializer % 1 > 0) ?

"a1f95c1bb38e78c13308.\_\_0xᵉ1ᵉ333ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃ364ᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ("

+ encrypted\_arg + ");"

: "a1f95c1bb38e78c13308.\_\_0xᵉ1488ᵉ371ᶠ3ᶠ8ᵃᵃ13212ᵉ9ᵃᵈ4790ᵃ53432ᶠ2ᵈᵇᶜᵃᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉᵈᵉᵃᵈᶜᵒᵈᵉ("

+ encrypted\_arg + ");";

var newVariable = SyntaxFactory.ParseStatement($"{ var\_declorator.Declaration.Type } { vr.Identifier } = { formattedInitializer };");

newVariable.NormalizeWhitespace();

editor.ReplaceNode(vr, newVariable);

Console.WriteLine($"[MUTATION] Key: { vr.Identifier } replaced with value: { formattedInitializer }");

}

current\_doc = editor.GetChangedDocument();

}

sln = sln.RemoveDocument(documentId);

sln = sln.AddDocument(DocumentId.CreateNewId(projId), current\_doc.Name, current\_doc.GetTextAsync().Result);

}

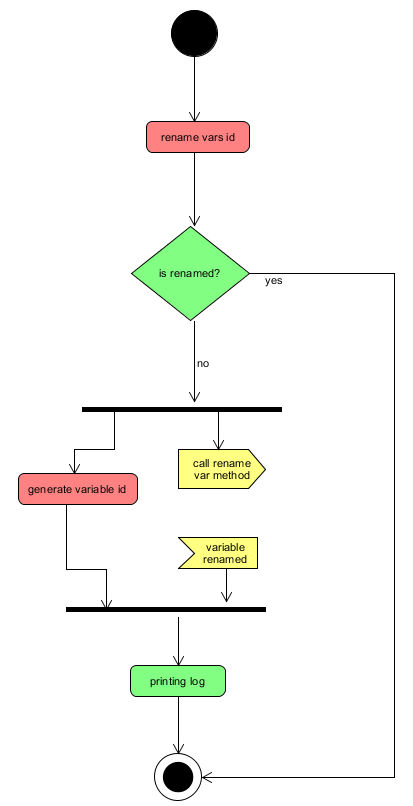
return sln;

}

}

}





Масса

Лит.

Масштаб

Обфускатор исходного кода C# «SRCFU5C4T0R»

Диаграмма деятельности метода переименования переменных

№ докум.

Подпись

Дата

Изм.

Лист

КП Т.691003.401 ГЧ

Багласова Е. В.

Лист 3

Реценз.

Т. Контр.

Утверд.

Н. Контр.

Блинов В. В.

Разраб.

Провер.

Инв.№подл.

Листов 5

КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.

Подп. и дата

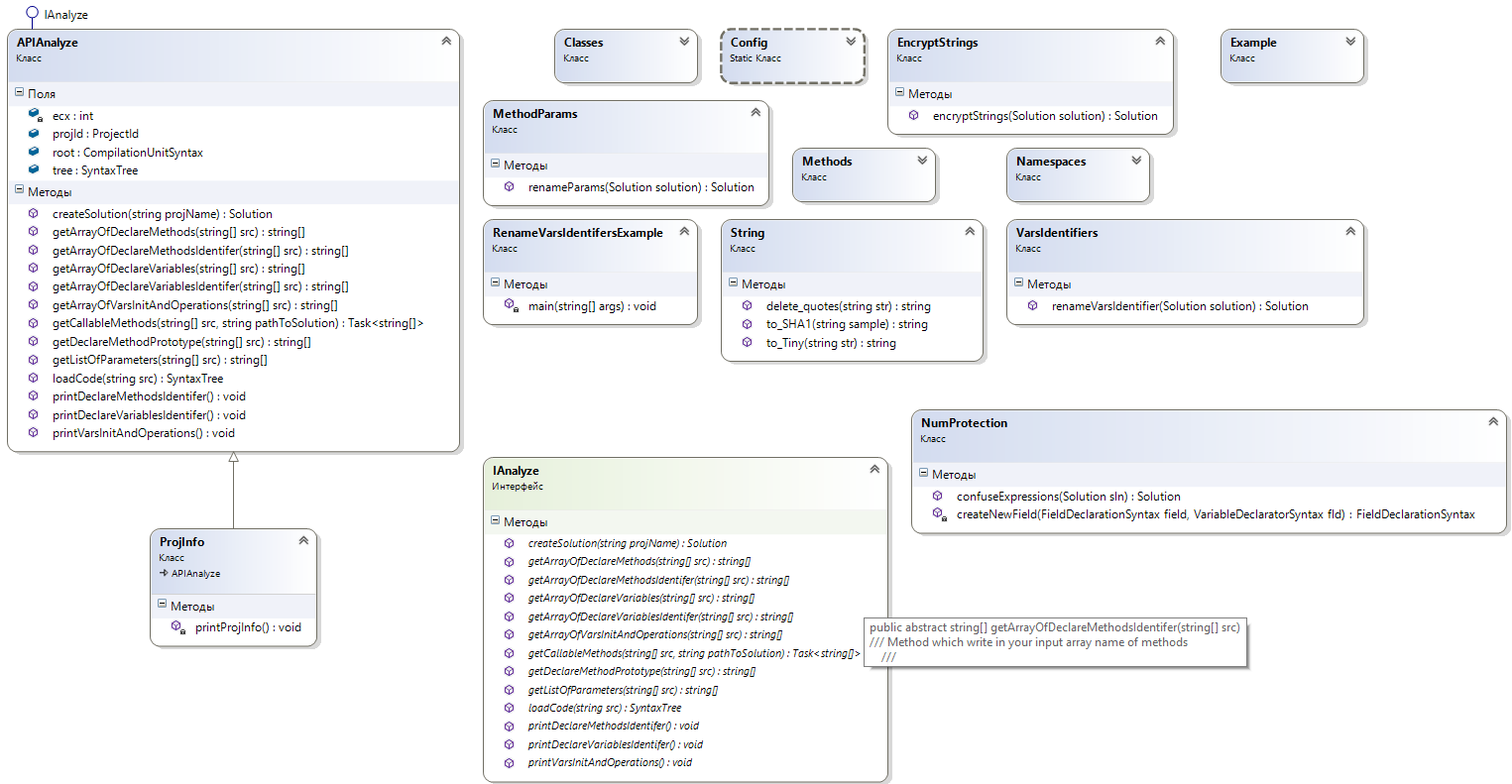
У



Масса

Лит.

Масштаб



КП Т.691003.401 ГЧ

Багласова Е. В.

Лист 2

Реценз.

Т. Контр.

Обфускатор исходного кода C# «SRCFU5C4T0R»

Диаграмма классов

Утверд.

Н. Контр.

Блинов В. В.

Разраб.

Провер.

Инв.№подл.

Листов 5

КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.

Подп. и дата

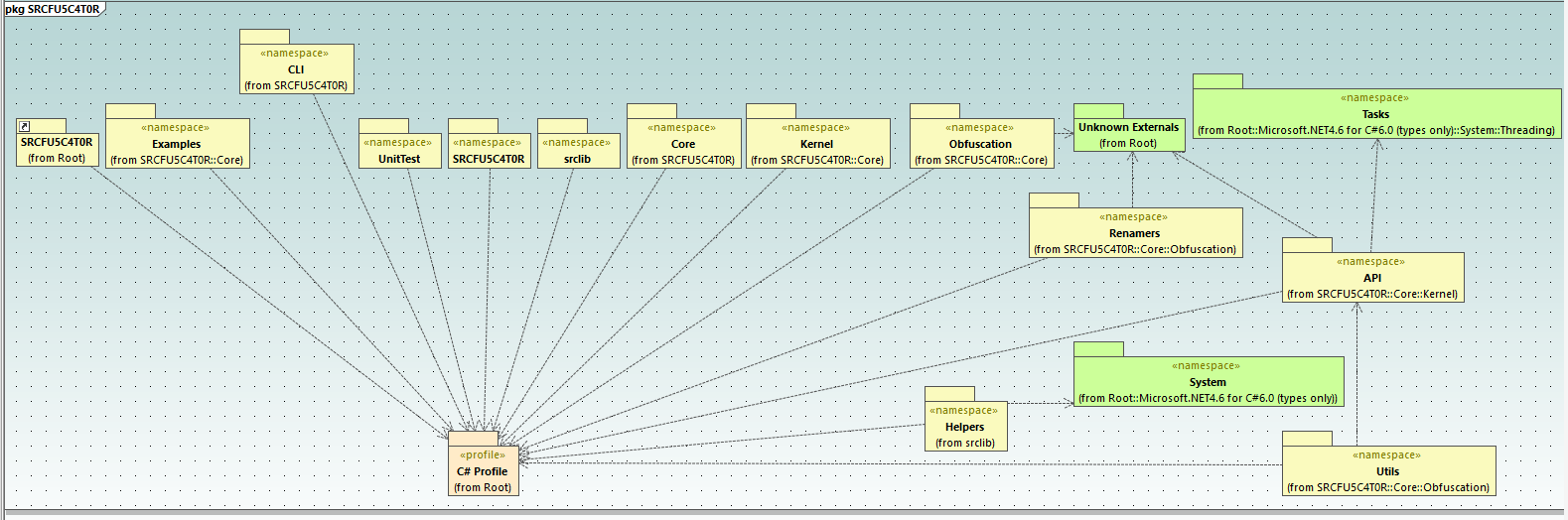
У



Масса

Лит.

Масштаб



КП Т.691003.401 ГЧ

Багласова Е. В.

Лист 3

Реценз.

Т. Контр.

Обфускатор исходного кода C# «SRCFU5C4T0R»

Диаграмма пакетов

Утверд.

Н. Контр.

Блинов В. В.

Разраб.

Провер.

Инв.№подл.

Листов 5

КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.

Подп. и дата

У



Масса

Лит.

Масштаб

КП Т.691003.401 ГЧ

Багласова Е. В.

Лист 4

Реценз.

Т. Контр.

Обфускатор исходного кода C# «SRCFU5C4T0R»

Диаграмма последовательности

Утверд.

Н. Контр.

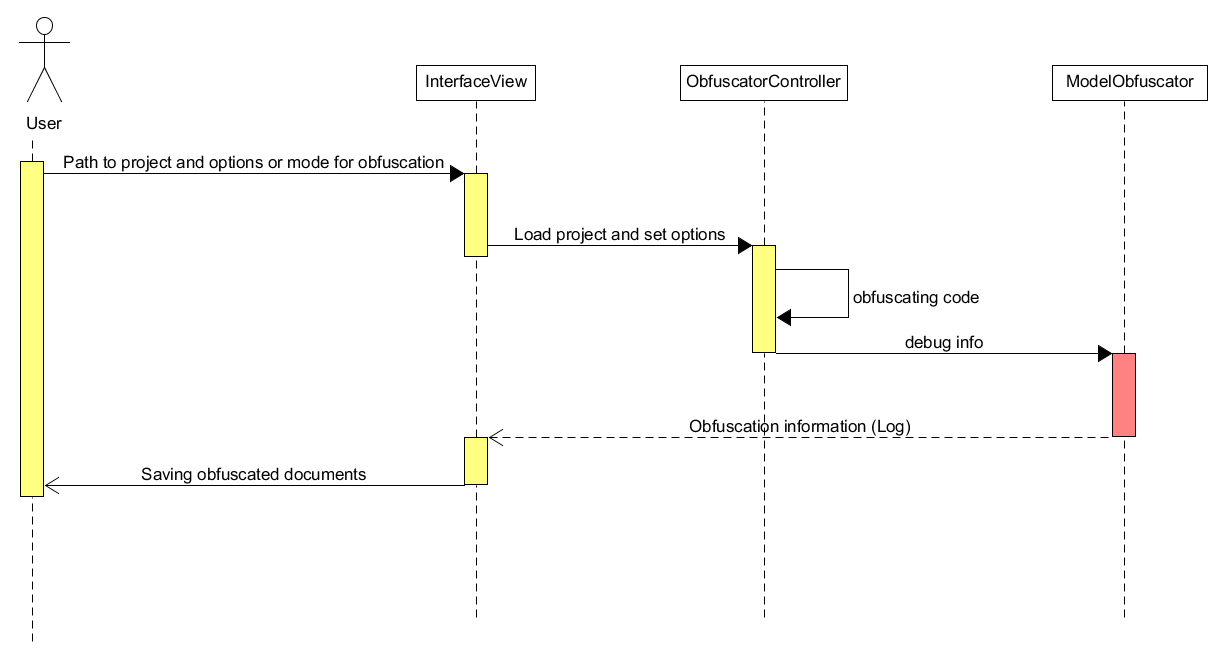
Блинов В. В.

Разраб.

Провер.

Инв.№подл.

Листов 5



КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.

Подп. и дата

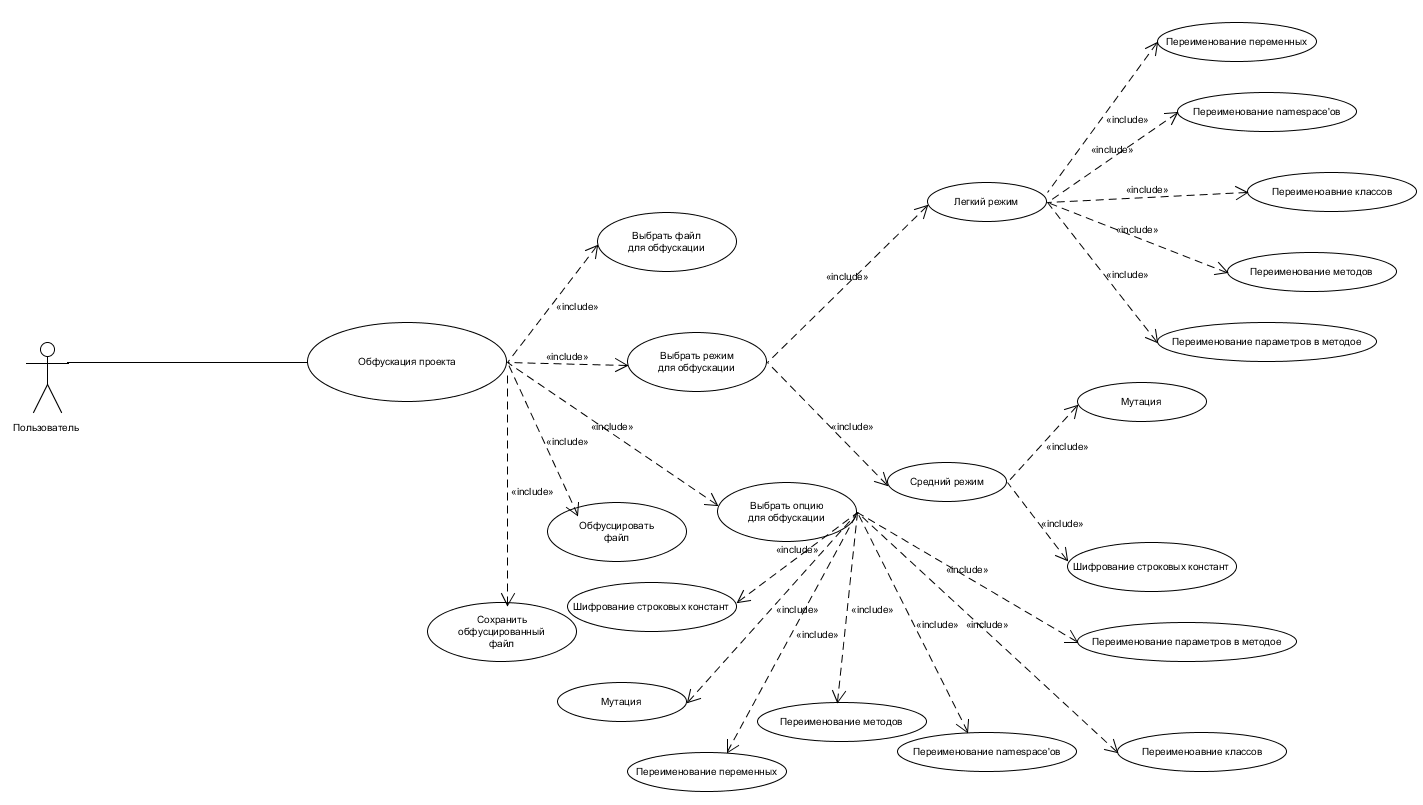
У



Масса

Лит.

Масштаб



КП Т.691003.401 ГЧ

Багласова Е. В.

Лист 5

Реценз.

Т. Контр.

Обфускатор исходного кода C# «SRCFU5C4T0R»

Диаграмма вариантов использования

Утверд.

Н. Контр.

Блинов В. В.

Разраб.

Провер.

Инв.№подл.

Листов 5

КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.

Подп. и дата

У