Санкт-Петербургский Государственный Политехнический университет

Институт информационных технологий и управления

Факультет технической кибернетики

Кафедра измерительных информационных технологий

**Отчет по преддипломной практике**

Тема: **Внедрение элементов защиты в .NET приложения  
без необходимости модификации исходного кода**

Направление: 090900 — Информационная безопасность

Специальность: 090104 — Комплексная защита объектов информатизации

Выполнил студент гр. 53505/2 С.А. Гладышев

Руководитель В.Ю. Сальников

Санкт-Петербург   
2014

**Оглавление**

[Введение 5](#_Toc385178662)

[Техническое задание 8](#_Toc385178663)

[1. Обзор технологий защиты 11](#_Toc385178671)

[1.1. Стеганография и водяные знаки 11](#_Toc385178672)

[1.2. Обфускация 13](#_Toc385178673)

[1.3. Упаковка 15](#_Toc385178674)

[1.4. Системы лицензирования 16](#_Toc385178675)

[1.5. Защита от модификаций 16](#_Toc385178676)

[1.6. Сокрытие данных 17](#_Toc385178677)

[1.7. Комплексный подход 18](#_Toc385178678)

[1.8. Обзор имеющихся решений 18](#_Toc385178679)

[2. Теоретическая разработка 21](#_Toc385178680)

[3. Реализация 22](#_Toc385178681)

[4. Тестирование 23](#_Toc385178682)

[Список литературы 24](#_Toc385178683)

**Перечень терминов и условных обозначений**

**CLR** (Common Language Runtime) **–** общеязыковая исполняющая среда. Исполняющая среда, интерпретирующая код на языке IL в байт-код, в который компилируются программы, написанные, в частности, на .NET-совместимых языках программирования.

**DLL** (Dynamic Link Library) **–** динамически подключаемая библиотека. Динамическая библиотека, позволяющая многократное использование различными программными приложениями.

**Easter Egg** («пасхально яйцо») **–** разновидность секрета, оставляемого в программном обеспечении создателями, особенность которого состоит в том, что его содержание, как правило, не вписывается в общую концепцию.

**IL** (CIL, MSIL, Common Language Runtime) **–** промежуточный язык, разработанный фирмой Microsoft для платформы .NET Framework.

**ILASM/ILDASM** (IL assembler/IL disassembler) – служебные программы, ассемблер и дизассемблер, поставляемые совместно с VisualStudio.

**JIT-компиляция** (Just-in-time) – динамическая компиляция, технология увеличения производительности программных систем, использующих байт-код, путём компиляции его в машинный код непосредственно во время работы программы.

**MSBuild** – платформа сборки проекта, разработанная Microsoft и позволяющая автоматизировать сборку и назначить дополнительные задачи.

**WinApi-приложение** (Windows application programming interfaces) – приложения, использующие набор базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows.

**ПО** – программное обеспечение.

**Сборка** (англ. assembly) – двоичный файл, содержащий управляемый код (метаданные и инструкции языка IL), получаемый на выходе компилятора.

# Введение

В наши дни информационные технологии являются основной движущей силой прогресса и захватывают все большие области современной жизни. Информационные технологии – это совокупность программно-технических средств вычислительной техники и способов их применения для обработки информации в различных областях. В связи с таким широким их распространением, серьезно встает вопрос защиты обрабатываемых и хранимых данных. Ведь если раньше, вы были уверены в том, что никто не получит доступ к вашим документам, лежащим в сейфе, без ключа или пароля. Теперь же большая часть информации хранится в электронном виде и необходимы уже другие способы защиты.

Этот принцип действует и для разрабатываемых программ. Если раньше, программы были по большей части низкоуровневыми и хранились в виде машинных кодов конкретной архитектуры, то с течением времени появляется все больше различных высокоуровневых инструментов для облегчения разработки, развиваются и усложняются сами языки программирования.

Чтобы снять с разработчика обязанности прямой работы с памятью, контроля типов, а также сделать программу более гибкой для различных платформ и языков, приложения выполняются специальной исполняющей средой (виртуальной машиной). Примеры таких виртуальных машин: JVM (виртуальная машина Java), CLR (общеязыковая исполняющая среда), Dalvik Virtual Machine (часть мобильной платформы Android). Эти средства значительно упрощают разработку, но они разработаны так, что требуют трансляции приложения в промежуточный байт-код (промежуточный язык, язык высокоуровневого ассемблера), который затем в момент выполнения компилируется в машинный ход JIT-компиляторами (компиляция «на лету»). Вследствие чего, приложения хранятся на диске в виде инструкций промежуточного языка, который легко декомпилируется обратно в исходные коды с помощью различных инструментов, таких как .NET Reflector, FernFlower (декомпилятор Java) и другие.

Соответственно существуют простые способы обхода лицензирования, получения исходных кодов и модификации под свои нужды незащищенных приложений написанных на языках .Net (и других подобных языков программирования, таких как Java). При этом для них не существует абсолютной защиты, как и не существует такой защиты для низкоуровневых приложений (например Win-32, написанных на С++), отсюда исходит основной принцип защиты программных продуктов:

*Защита приложения лишь увеличивает время его взлома и в идеале, защита должна окупить себя, т.е. время, затраченное на защиту приложения должно быть сопоставимо со временем, затраченным на взлом этой защиты.*

Существуют несколько основных технологий или методик защиты .Net приложений. Их можно разделить на две части: те, которые работают с кодом программы и те, которые работают с памятью.

Для сокрытия кода используются различные обфускаторы и упаковщики. Обфускаторы используют для защиты запутывание кода (англ. control flow), переименование методов и классов, шифрование ресурсов, добавление невалидных конструкций и др. Упаковщики модифицируют сборку, и на выходе получается WinApi-приложение, содержащие в себе код сборки и модуль распаковки. Упаковщики и обфускаторы могут применяться одновременно.

В дополнение к защите кода может применяться защита данных, хранящихся в памяти во время выполнения программы. Здесь возможно множества решений, которые зависят от конкретного приложения и его архитектуры.

Данная работа направлена на рассмотрение существующих решений, выявления преимуществ и недостатков и разработку нового решения, основанного одновременно на нескольких методах и технологиях, как защиты, так и упрощения ее внедрения в готовый проект.

# Техническое задание

## Введение

Данный программный продукт предназначен для защиты данных, используемых в приложениях, написанных с использованием .NET Framework. Программа модифицирует целевое приложение и тем самым защищает данные, находящиеся в памяти во время его работы.

## Основания для разработки

Разработка решения ведется на основании обзора и анализа существующих решений в сфере защиты программного обеспечения в рамках текущего дипломного проекта.

## Назначение разработки

Программа представляет собой отдельное приложение, позволяющее модифицировать .NET сборки путем внедрения кода, обеспечивающего определенную защиту. Внесенные изменения должны предоставлять эффективную защиту данных приложения, хранящихся в памяти и по возможности препятствовать их модификации.

## Требования к программному продукту

Требования к функциональным характеристикам.

Программа должна уметь работать с запускаемыми exe файлами и библиотеками функций DLL. Цель модификации – внедрение конструкций, отвечающих за выполнение логики по защите данных.

Логика защиты данных описана в отдельной DLL библиотеке, которая должна быть подключена к модифицируемому приложению и поставляться вместе с ним конечному пользователю.

Входными данными для приложения является валидная .NET сборка, транслированная в промежуточный язык IL. На выходе необходимо получить приложение с тем же функционалом и не нарушенной логикой работы.

Должна быть предусмотрена возможность модификации приложения в ходе сборки проекта с использованием платформы MSBuild.

Требования к надежности

При некорректном завершении работы приложения по каким-либо причинам, состояние модифицируемой сборки должно быть восстановлено до исходного.

Требования к составу и параметрам технических средств

Минимальные технические характеристики:

* Windows XP SP1 и выше;
* Microsoft Visual Studio 2005 и выше;
* .NET Framework 2.0 и выше.

Требования к информационной и программной совместимости

Модифицируемые сборки должны быть написаны на платформе .NET Framework версии 2.0 и выше на следующих языках программирования: C#, F#, Visual Basic.

Специальные требования

Специальных требований к характеристикам программы не предъявляется.

## Технико-экономические показатели

Разрабатываемый продукт является бесплатным для распространения.

Время, затраченное на разработку, составит 20 человеко-часов.

## Стадии и этапы разработки

Эскизный проект – 1 час

Разрабатывается структура программы, алгоритм модификации сборок и алгоритм защиты данных. Определяется система взаимодействия с модифицируемыми приложениями.

Тестовый проект – 6 часов

Создание рабочего решения, позволяющего защищать приложения с помощью простого алгоритма. Отсутствие проверок и восстановления после ошибок.

Технический проект – 4 часа

Написание эффективного алгоритма по защите данных, добавление обработки ошибок и восстановления после некорректного завершения работы.

Рабочий проект – 4 часа

Реализация автоматизации процесса, интеграция с процессом сборки модифицируемых решений. Подготовка контрольно-отладочного примера.

Тестирование – 5 часов

Тестирование работоспособности решения. Попытка противодействовать используемой защите.

## Порядок контроля и приемки

Контроль и приемка разработки осуществляются на основе проведенных испытаний контрольно-отладочных примеров. При этом проверяется выполнение всех функций программы.

# Обзор технологий защиты

В данной главе будут рассмотрены основные технологии защиты программного обеспечения, исполняемого в среде .Net и приведены популярные существующие решения для этих областей. По завершению этой главы будет проведен анализ преимуществ и недостатков рассмотренных средств защиты.

## Стеганография и водяные знаки

Стеганография – это метод защиты программного обеспечения (ПО) путем встраивания секретной информации в приложение (код или данные). Это делается для идентификации владельца ПО и позволяет избежать или детектировать создание нелицензионных копий приложений. Подобное внедрение водяных знаков (англ. watermarks) позволяет владельцам лицензионного ПО определить подлинность копии путем извлечения водяных знаков.

По методу встраивания водяные знаки можно разделить на две категории: статические (англ. static) и динамические (англ. dynamic). К статическим относятся: модификация данных (строк) и кода. Для детектирования таких водяных знаков не обязательно запускать приложение, но они очень легко снимаются. К динамическим относятся: «пасхальные яйца» (англ. dynamic Easter Egg watermarks) – изменения в незаметном или редко используемом месте программы, изменения алгоритма работы программы (англ. dynamic execution trace watermarks) или изменение динамических данных, доступных из вне (англ. dynamic data structure watermarks), например, глобальных переменных или областей памяти.

Чтобы понять эффективность тех или иных мер защиты, необходимо понимать потенциальные направления атак. Рассмотрим возможные виды атак на водяные знаки:

* Удаление водяных знаков (англ. subtractive attack). Заключается в нахождение измененного места и вырезания защиты.
* Искажение водяных знаков (англ. distortive attack). Цель такой атаки – исказить значение так, чтобы владелец не смог идентифицировать свое приложение.
* Добавление водяных знаков (англ. additive attack). Добавление новых водяных знаков затрудняет доказательство владельцем подлинности продукта.

Любая из указанных атак считается эффективной, если не нарушена работоспособность приложения и в случае добавления или искажения, механизм создания новых водяных знаков должен использовать алгоритм владельца.

Исходя из особенностей атак на водяные знаки мы можем определить стратегию защиты ПО с использованием данного метода:

* Водяные знаки должны быть скрыты от обнаружения;
* Водяные знаки должны обладать устойчивостью к модификации приложения (обфускации, деобфускации);
* Чем больше используется различных способов внедрения водяных знаков и их количество, тем эффективнее защита;
* Внедрение защиты не должно сказываться на производительности приложения;
* Водяные знаки следует генерировать с использованием закрытого алгоритма или ключей (ассиметричное шифрование), что позволит избавиться от проблемы добавления новых водяных знаков злоумышленником.

Как пример эффективного решения можно привести следующий алгоритм: в программе создается несколько статических массивов и, чтобы они не были «мертвыми», нам необходимо вовлечь их в какие-либо вычисления. Генерировать значения водяных знаков будем с помощью ассиметричного алгоритма шифрования (например, AES) и будем использовать хеширование для точной идентификации водяных знаков в памяти. Секретный ключ для генерации хранится у разработчика. Проверять приложения на подлинность будем путем анализа памяти и нахождения значений наших водяных знаков.

Подобные методы защиты могут быть реализованы как в отдельных продуктах, так и в комплекте с обфускатором.

## Обфускация

Обфусцированной (англ. obfuscated, запутанной) называется программа, которая на всех допустимых для исходной программы входных данных выдаёт тот же самый результат, что и оригинальная программа, но более трудна для анализа, понимания и модификации [3].

Соответственно, *обфускация* – это процесс применения к исходной программе запутывающих преобразований, затрудняющих дальнейший ее анализ. Обфускация может осуществляться на разных уровнях, соответствующих типам информации, на которые она нацелена, так, основываясь на таксономии запутывающих преобразований [4], можно выделить следующие уровни обфускации:

1. **Запутывание форматирования исходного текста** (англ. layout obfuscation).

Сюда относятся изменение имен методов и классов, объединение классов и пространств имен, удаление форматирования, комментариев, изменение метаданных. Данные преобразования могут применяться как к исходному тексту программы на языке высокого уровня, так и к скомпилированной сборке. Особо не влияют на сложность анализа.

1. **Запутывание данных** (англ. data obfuscation).

Изменения могут затрагивать размещение данных в памяти (имеется ввиду перенос переменных в глобальную область, объединение нескольких переменных в одну, изменение порядка в массиве), кодирование, агрегацию и перемешивания строк и ресурсов.

1. **Запутывание потока управления** (англ. control flow obfuscation).

В отличие от запутывания данных, данный метод способен достаточно сильно увеличить сложность программы. Можно выделить следующие способы преобразований:

* Преобразования агрегации. Вставка и переплетение функций, раскрутка циклов, клонирование кода;
* Преобразования вычислений. Вставка избыточного кода, и усложнение вычислительных преобразований;
* Использование непрозрачных предикатов. Непрозрачный предикат – предикат, значение которого известно в момент запутывания, но трудноустанавливаемо после [3], это такие выражение, которые не зависимо от значений аргументов, всегда истины, либо ложны.
* Табличная интерпретация. Добавление таблиц переходов (например, больших CASE структур), сильно увеличивающих ветвление кода. Добавление интерпретатора (виртуальной машины) – свойственно упаковщикам.

1. **Превентивные трансформации** (англ. preventive transformation).

Такие трансформации ориентированы на отладчики, декомпиляторы, деобфускаторы. Могут быть как врожденными (затрудняющие распутывание существующими методами) и целевыми (использующие известные проблемы конкретных решений).

На рынке существует множество различных обфускаторов, отличающихся своим набором возможностей для обфускации, рассмотренных выше, возможностями противодействия деобфускации, цена и зависимостью от версий сборок и платформы. В общем, обфускаторы можно разделить на две группы по используемым средствам воздействия на обфусцируемую сборку [6]:

* Stand-alone обфускаторы, использующие свои интерфейсы для доступа к метаданным и генерации обфусцированной сборки;
* Зависимые от сервисов .Net (ILASM/ILDASM). Функционал таких обфускаторов заведомо ограничен тем набором инструментов, которые предоставляют сервисы.

## Упаковка

*Упаковка* – способ модификации сборки, при котором на выходе получается WinApi-приложение, содержащее в себе упакованный (не обфусцированный) IL код и распаковочный модуль.

Такой способ вносит некоторые ограничения:

* Сборка остается рабочей только под операционной системой Windows;
* Из-за этого теряется преимущество JIT-компиляции среды CLR, дающее гибкость для х64 и х86 платформ (генерация различных инструкций, в зависимости от платформы);
* Упаковка немодифицированных сборок не дает серьезной защиты и легко обходится;
* Упаковщики могут перехватывать вызовы виртуальной машиной функций библиотеки mscoree.dll, что предоставляет дополнительную защиту от его снятия. Но на работу таких программ может влиять антивирус и они сильно зависимы от версии .Net Framework.

Упаковщики помимо своей основной функции обычно содержат множество средств дополнительной защиты, основными из которых являются:

* Отслеживание отладки. Позволяет запретить или детектировать выполнение приложения под отладчиком. В продвинутом варианте, оболочка (упаковщик) приложения может передать сигнал на нижний уровень, где, например, может произойти затирание отлаживаемого кода в памяти.
* Обнаружение виртуальной машины. Запрещает выполнение приложения под виртуальной машиной
* Защита памяти и проверка целостности данных. Отслеживание и выявление обращений к памяти и ее модификации.
* Защита импорта и создание DLL оберток. Позволяет либо скрыть информацию об используемых библиотеках, либо исключить их хранение на диске путем включение DLL в состав исполняемого файла.

## Системы лицензирования

Данный метод защиты используется для защиты лицензионного ПО от копирования. Основная суть заключается в выдачи вместе с продуктом уникального ключа, который однозначно идентифицирует данную копию продукта. Отслеживание подлинности продуктов может проводится с использованием сервера лицензирования и активации, это значительно повышает устойчивость к взлому. Сервер активации предполагает, что при установке приложения, оно должно быть активировано с помощью ключа активации, использование которого строго отслеживается, таким образом, избегается возможность повторной установки.

Часто такие системы входят в состав продуктов, называемых протекторами. *Протекторы* (англ. native processor wrapper) – приложения, совмещающие в себе упаковщик и обфускатор (обычно довольно слабый). Протекторы являются комплексным решением.

При генерации упакованной сборки, протектор внедряет в нее блок активации. Многие протекторы, имеющие систему лицензирования предлагают различные дополнительные сервисы:

* Сервер активации;
* Генератор ключей. Возможность генерировать ключи на сервере в автоматическом режиме;
* Деактивация ключей. Позволяет пользователю перенести программу на другой компьютер и активировать ее заново;
* Блокировка ключей. В приложение встраивается база скомпрометированных ключей активации;
* Привязка к аппаратному обеспечению.

## Защита от модификаций

Суть данного метода заключается в обнаружении нарушения целостности оригинального ПО. Может применяться совместно с обфускацией приложения и защищать его от попыток деобфусцирования.

Самый простой способ защиты от модификаций (англ. tamper-resistant**)**– это подсчет хеш-суммы всего приложения или его модулей (в случае использования упаковщика – хеш-суммы упакованного кода). Полученное значение сравнивается с оригинальным значением, рассчитанным для данной сборки. Минус статического расчета хеш-суммы в возможности модификации приложения уже после его запуска.

Так же существуют методы динамической самопроверки (англ. integrity veriﬁcation). В отличие от проверки статического образа приложения (файл на диске), программа сравнивает значения в памяти. Могут сравниваться значения переменных в памяти на выходе из функций, хеш-сумма последних выполненных инструкций или значений в стеке [8].

Более продвинутые метод заключается в шифровании исходного кода в памяти. Секции кода, незадействованные в текущих вычислениях могут быть зашифрованы в любое время и расшифрованы при необходимости их выполнения. Таким образом, исключается модификация кода без знания алгоритма шифрования. Данный способ можно усложнить еще больше, транслировав весь код в код виртуальной машины [9].

## Сокрытие данных

Рассмотренные ранее методы защиты помогают защитить код от кражи и модификации, позволяют отслеживать подлинность используемого ПО или противодействовать отладке, но они не помогают защитить данные приложения. Запущенное приложение в какой-то момент хранит все свои данные в оперативной памяти, откуда злоумышленник может их получить. От этого может спасти модификация данных, перед записыванием их в память. Возможных реализаций этого способа защиты существует множество, и они будут рассмотрены позже.

В ходе анализа не было найдено продуктов, предоставляющих подобную защиту, но она, несомненно, используется в готовых приложениях.

## Комплексный подход

Как уже было сказано ранее, нельзя полностью защитить приложение от взлома, можно лишь увеличить необходимые на это затраты. Сокрытие данных не придаст приложению дополнительной защиты без использования других методов, т.к. такую защиту легко снять, то же самое относится к системе лицензирования, защите от модификаций, да и любого рода проверкам.

Для эффективной защиты необходимо применять комплексный подход, который и реализован в готовых предлагаемых решениях, которые будут рассмотрены далее.

## Обзор имеющихся решений

В основном на рынке инструменты для защиты приложений представлены в виде двух групп: обфускаторы и протекторы.

На основании обзоров и анализов обфускаторов был выбран Spices.Net Obfuscator как один из лучших в данной группе, благодаря следующим возможностям:

* Неограниченная версия для некоммерческих проектов;
* Интеграция в MSBuild и VisualStudio;
* Оперативная поддержка на русском языке;
* Реализация основных методов защиты.

Из множества протекторов, работающих с .Net сборками можно выделить следующие: Obsidium, Enigma Protector, Oreans Themida, .NET Reactor. Все они имеют примерно одинаковую цену ($150-200) и за исключением .NET Reactor’a схожие возможности. .NET Reactor больше ориентирован на .Net приложения и включает в себя довольно мощный обфускатор, что достаточно важно для .Net приложений с открытым исходным кодом. В нем так же предусмотрена достаточно гибкая система лицензирования.

В качестве протектора для сравнения был выбран Enigma Protector, благодаря более богатому набору функционала для защиты.

Лучшие отобранные решения будут сравниваться по описанным методам защиты приложений. Если данный метод реализован, напротив него ставится «+», если нет – «–», если информация не предоставлена – «?».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможности | Spices.Net Obfuscator | .NET Reactor | Enigma Protector | Мое решение |
| Обфускация кода |  |  |  |  |
| Защита кода | + | + | + | +/– |
| Защита от дизасемблирования | + | + | – | – |
| Запутывание потока управления | + | + | – | – |
| Упаковка | – | + | + | – |
| Система лицензирования | – | + | + | – |
| **Tamper-resistant** | + | – | + | – |
| Watermarks | + | – | + | – |
| Сокрытие данных | – | – | – | + |
| Поддержка различных версий и языков .Net | + | + | ? | + |
| Цена | $400 | $180 | $150 | $0 |

Исходя из проведенного обзора, можно сделать некоторые выводы и рекомендации. Enigma Protector предоставляет множество различных методов защиты, но можно предположить, что из-за достаточно простой обфускации, эту защиту достаточно легко снять. Spices.Net Obfuscator представляет собой очень мощный инструмент для обфускации приложений и если не нужна система лицензирования, это, пожалуй, лучший вариант. .NET Reactor – что-то среднее между двумя описанными решениями.

Но ни одно из этих решений не предоставляет инструмента для защиты данных, что так же является довольно важным элементом защиты. Этот метод защиты и будет представлен в моем решении, которое может быть использовано как часть комплексной подхода к защите приложения.

# Теоретическая разработка

# Реализация

# Тестирование

# Список литературы

|  |  |
| --- | --- |
| [1] | F. Z. William, «Concepts and Techniques in Software Watermarking and Obfuscation,» The Department of Computer Sciences, The University of Auckland, 08 2007. [В Интернете]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.8892&rep=rep1&type=pdf. [Дата обращения: 12 04 2014]. |
| [2] | C. Collberg и C. Thomborson, «Software Watermarking: Models and Dynamic Embeddings,» Department of Computer Science, The University of Auckland, 1999. [В Интернете]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.8088&rep=rep1&type=pdf. [Дата обращения: 12 04 2014]. |
| [3] | А. В. Чернов, «Анализ запутывающих преобразований программ,» Института Системного программирования РАН, 2003. [В Интернете]. Available: http://citforum.ru/security/articles/analysis/. [Дата обращения: 11 04 2014]. |
| [4] | С. Collberg, С. Thomborson и D. Low, «A Taxonomy of Obfuscating Transformations,» Department of Computer Science, The University of Auckland, 1997. [В Интернете]. Available: http://www.labri.fr/perso/fleury/courses/SS/download/papers/obfuscation-survey-collberg.pdf. [Дата обращения: 11 04 2014]. |
| [5] | И. Ледовских, «Метрики сложности кода,» 27 01 2012. [В Интернете]. Available: http://www.ispras.ru/ru/preprints/docs/prep\_25\_2013.pdf. [Дата обращения: 11 04 2014]. |
| [6] | eRaider, «Защита .NET приложений — всё же, во что заворачивать селёдку?,» 16 10 2010. [В Интернете]. Available: http://habrahabr.ru/post/106262/. [Дата обращения: 11 04 2014]. |
| [7] | А. Трамвон, «Обзор систем защиты ПО для Windows от нелегального использования,» 15 03 2014. [В Интернете]. Available: http://habrahabr.ru/post/215553/. [Дата обращения: 11 04 2014]. |
| [8] | P. Wang, «Tamper Resistance for Software,» Information and Communications University, 2005. [В Интернете]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.8556&rep=rep1&type=pdf. [Дата обращения: 12 04 2014]. |
| [9] | G. Wurster, «A generic attack on hashing-based software tamper resistance,» Ottawa-Carleton Institute for Computer Science, 04 2005. [В Интернете]. Available: http://people.scs.carleton.ca/~gwurster/publications/Thesis-2005.pdf. [Дата обращения: 12 04 2014]. |
| [10] | Antelle, «Обзор обфускаторов для .NET,» 22 06 2010. [В Интернете]. Available: http://habrahabr.ru/post/97062/. [Дата обращения: 11 04 2014]. |