К.С. Сарин, Л.Д. Сеитбекова

БЕЗОПАСНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к практическим работам

для студентов специальностей и направлений

10.03.01 – «Информационная безопасность»,

10.05.03 – «Информационная безопасность автоматизированных систем»,

10.05.04 – «Информационно-аналитические системы безопасности»

В-Спектр
Томск, 2018

УДК 004.056

ББК 32.973.26-018.2

К 64

К 64 **Сарин К.С., Сеитбекова Л. Д.** Безопасность программного обеспечения. Ч. 1: методические указания к лабораторному практикуму. – Томск: В-Спектр, 2018. – 57 с.
ISBN 978-5-91191

Практикум по дисциплине «Безопасность программного обеспечения» для специальностей 10.05.03 – «Информационная безопасность автоматизированных систем», 10.05.04 – «Информационно-аналитические системы безопасности» и направления 10.03.01 – «Информационная безопасность», содержит задания, методические указания по выполнению практических занятий, требования по представлению отчётности, вопросы для самоконтроля.

Содержание

Введение

Методические рекомендации к практическим занятиям подготовлены с целью обучения студентов специальностей 10.03.01 – «Информационная безопасность», 10.05.03 – «Информационная безопасность автоматизированных систем», 10.05.04 – «Информационно-аналитические системы безопасности». На занятиях студенты приобретут умения и навыки защиты программного обеспечения от исследования.

На занятиях студенты в интерактивной форме освоят профессиональные компетенции, обозначенные в основной образовательной программе по данной дисциплине. Работая с дизассемблером, изучая ассемблерный код, преодолевая защиту исполняемого модуля программы, а также осуществляя защиту посредством ConfuserEx? студенты специальности 10.03.01 – «Информационная безопасность», смогут освоить компетенцию ПК-2 – способность применять программные средства системного, прикладного и специального назначения, инструментальные средства, языки и системы программирования для решения профессиональных задач.

Обучающиеся по направлению 10.05.03 – «Информационная безопасность автоматизированных систем» получат способность участвовать в разработке защищенных автоматизированных систем в сфере профессиональной деятельности, что является профессиональной компетенцией ПК-9, а также способность проводить инструментальный мониторинг защищенности информации в автоматизированной системе и выявлять каналы утечки информации, что является профессиональной компетенцией ПК-17.

В свою очередь, студенты направления 10.05.04 – «Информационно-аналитические системы безопасности» освоят компетенцию ПК-10 – способность осуществлять выбор технологии, инструментальных средств, средств вычислительной техники и средств обеспечения информационной безопасности создаваемых специальных ИЛС.

Практическая работа № 1

Исследование приложений .NET Framework с помощью декомпиляции

Цель работы – изучить особенности .Net приложений, ознакомиться с возможностями декомпилятора JetBrains dotPeek, преодолеть защиту приложения при помощи данного декомпилятора.

Краткие теоретические сведения

Основным инструментом Microsoft для разработки программного обеспечения на данный момент является .NET Framework, представляющим собой программную платформу, основой которой является общеязыковая среда исполнения Common Language Runtime (CLR).

Языки программирования, использующие среду CLR, могут использовать все ее функциональные возможности, это значит, что

исключение, созданное методом, написанном на C++, может быть перехвачено и обработано в Delphi, а служба, написанная на языке C\#, может обратиться к методу класса, написанной на языке Delphi.

Visual Studio является компилируемой средой разработки, что подразумевает наличие по итогам компиляции исполняемого модуля программы, того самого, что подлежит распространению (использует пользователь) и защиту которого необходимо произвести.

Процесс разработки и выполнения приложений .Net, состоит из следующих основных этапов (рис.1):

– создание программы на любом подходящем языке .NET;

– создание байт-кода, т.е. компиляция исходного кода в модуль на промежуточном языке (IL - Intermediate Language);

– объединение управляемых модулей и создание сборки;

– развертывание или установка сборки на предназначенную платформу;

– обращение к среде CLR (CLR - Common Language Runtime), которая загружает, компилирует байт-код в базовый и выполняет его.

Среда CLR работает с промежуточным языком низкого уровня, наподобие ассемблера MSIL (Microsoft Intermediate Language) или IL или, как его еще называют байт-код. При загрузке приложения на выполнение CLR загружается в память первой, выполняет службы, необходимые операции, включая компиляцию, распределение памяти и управление кодом обратившегося к ней приложения.

IL код компилируется непосредственно перед запуском программы. Такая технология получила название Just-in-time compilation (JIT, компиляция

«на лету»). JIT-компилятор осуществляет динамическую компиляцию, преобразуя байт-код в машинный код, таким образом реализуя технологию увеличения производительности программ, за счет достижения высокой скорости выполнения.

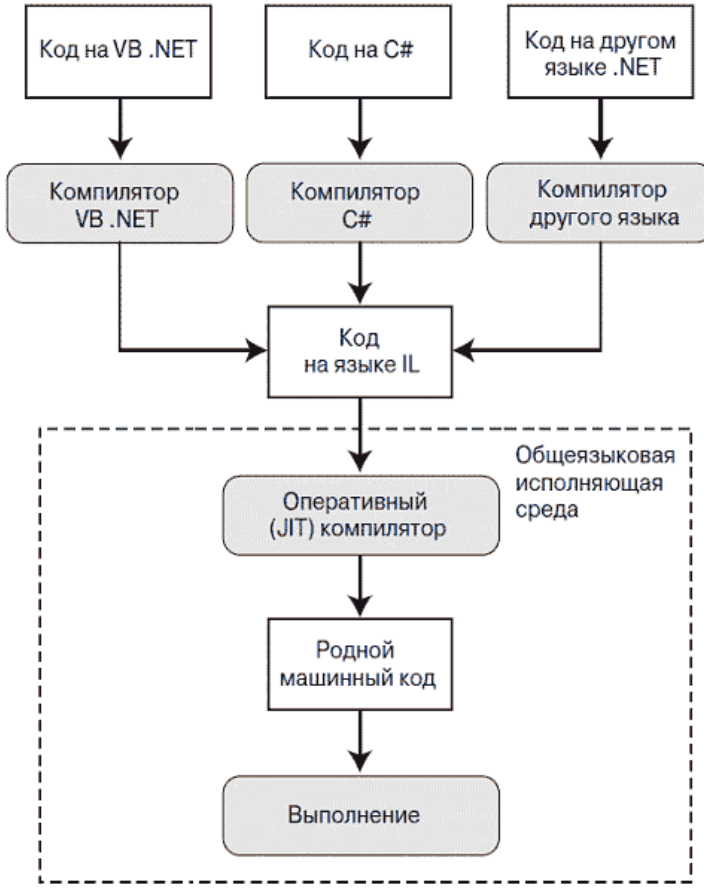


Рис. 1. Реализация .Net программы

Сборки предоставляют развертывание и контроль версий, помогают улучшить доступ к коду, а также его защиту.

Метаданные необходимы для описания свойств других данных: информация о интерфейсе экспортируемых классов, каждом типе данных: его имя, типы и имена его полей, описание свойств и методов со всеми их

параметрами и возвращаемыми значениями, о доступности всех членов класса и об их атрибутах, отражаются такие детали реализации, как описания защищённых методов и других компонентов, структура защищённых полей.

Именно из-за сборки в байт-код и наличия метаданных, приложения .Net легко поддаются декомпиляции.

Декомпилятор – это программа, способная на основании исполняемого модуля, воссоздать исходный код на языке высокого уровня.

Декомпиляция – процесс воссоздания декомпилятором исходного кода программ.

Из бесплатно распространяемых декомпиляторов выделяется JetBrains dotPeek, так как обладает рядом отличительных особенностей:

– для предоставления декомпозированного кода использует большинство привычных пользователям Microsoft Visual Studio функций: нумерация строк, подсветка синтаксиса, открытие декомпилированных файлов в отдельных вкладках;

– позволяет не только исследовать сборку, но и вносить в нее изменения;

– позволяет сохранять результат полученных изменений в проект Microsoft Visual Studio, тем самым давая возможность продолжить работу с исходным кодом сборки.

Для работы с dotPeek выделяются следующие системные требования:

– процессор: мин. Intel Core 2 Duo 2 ГГц;

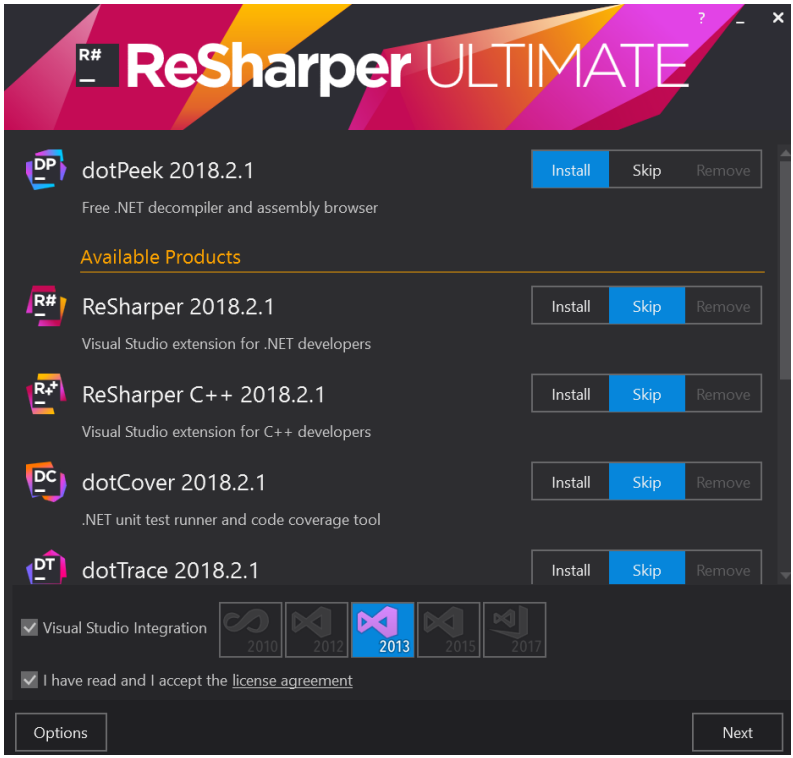
– оперативная память: для интеграции Visual Studio: мин. 4 ГБ, рекомендуется 6 ГБ;

– диск: минимальное пространство: 400 Мб;

– операционная система (рекомендуется 64-разрядная ОС для оптимальной работы): Microsoft Windows 7/ Server 2008 R2/ 8/ Server 2012/ 8.1/ Server 2012 R2/ 10.

Ход работы

Для начала работы с JetBrains dotPeek, его необходимо скачать с данного ресурса https://www.jetbrains.com/decompiler/?fromMenu. После запуска установочного файла, из предложенных приложений стоит выбрать только dotPeek, а так же интеграцию с той версией Visual Studio, что установлена на вашем компьютере, нажать кнопку «Next», начнется установка (рис.).



После запуска программы, необходимо добавить исследуемый файл «Курсовая» (рис. 2).

После чего добавленный файл отразится во вкладке «Assembly Explorer», в которой можно увидеть все декомпилированные модули программы (рис. 3).

Так как никакой защиты от обфускации на исполняемом файле не было применено, то можно легко найти модуль, отвечающий за реализацию парольной защиты приложения (рис. 4). Как видно из рисунка пароль введенный пользователем сохраняется в переменной «textBox1.Text». Эталонным является пароль «ТУСУР2018!», при нажатии кнопки «button1_Click», веденный пароль сравнивается с эталонным, при успешном варианте пользователю открывается доступ к функциям программы, в противном случае требуется повторный ввод пароля. Благодаря декомпилированному коду можно узнать эталонный пароль, либо вовсе убрать условие проверки пароля.

Для поиска и навигации в связанных и загруженных сборках, включая сборки платформы .NET Framework, предоставляет широкий набор функций.

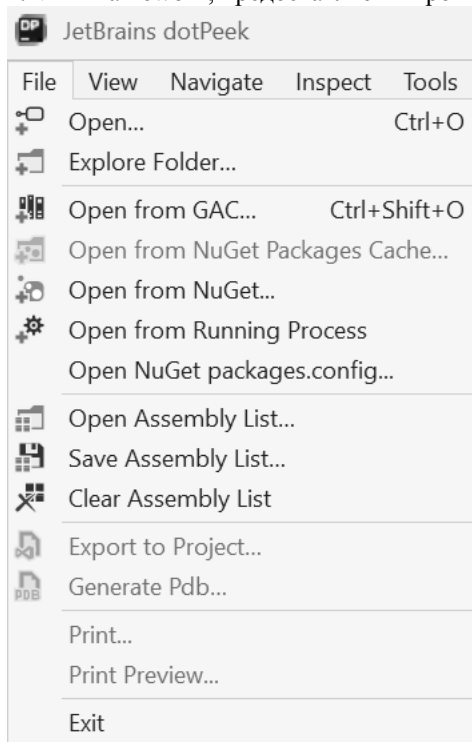


Рис. 2. Выбор файла

Для использования навигации, необходимо щелкнуть правой кнопкой мыши по искомому элементу в коде, в открывшемся окне нажать «Navigate» (рис. 5). Используя средства навигации осуществляется анализ кода.

Для возможности редактирования исходного кода, необходимо сохранить декомпилировании файл в проект Visual Studio, для этого на панели «Assembly Explorer» необходимо нажать на кнопку со знаком среды Visual Studio. В открывшемся окне в поле «Destination Folder» необходимо выбрать путь сохранения проекта (рис. 6.).

Следует уточнить что созданный проект не является полным аналогом реального проекта Visual Studio, формы Windows и все их элементы придется создавать заново, а, следовательно, и корректировать декомпилированный код. Сравнение изначального проекта Visual Studio и созданного посредством dotPeek представлено на рис. 7.

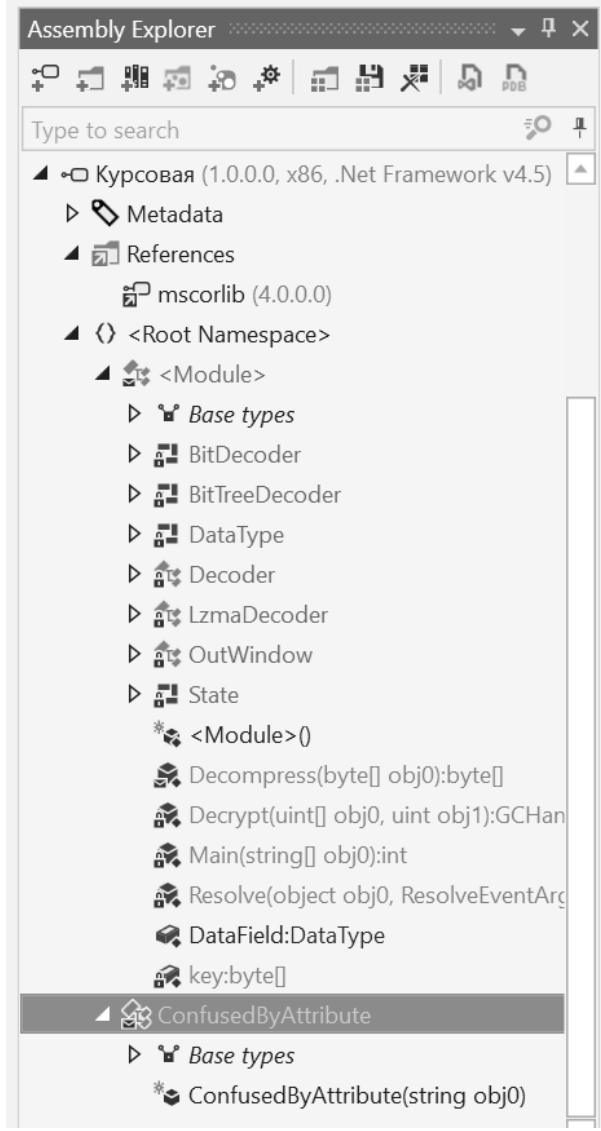


Рис. 3. Модули исследуемого файла

Parol.cs

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication3
{
    public partial class Parol : Form
    {
        public Parol()
        {
            InitializeComponent();
        }

        private void Parol_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text == "TUSUR2018!")
            {
                this.Hide();
                Form1 main = new Form1();
                main.Show();
            }
            else
            {
                MessageBox.Show("invalid password!", "Error", I
            }
        }
    }
}
```

Рис. 4. Декомпилированный код

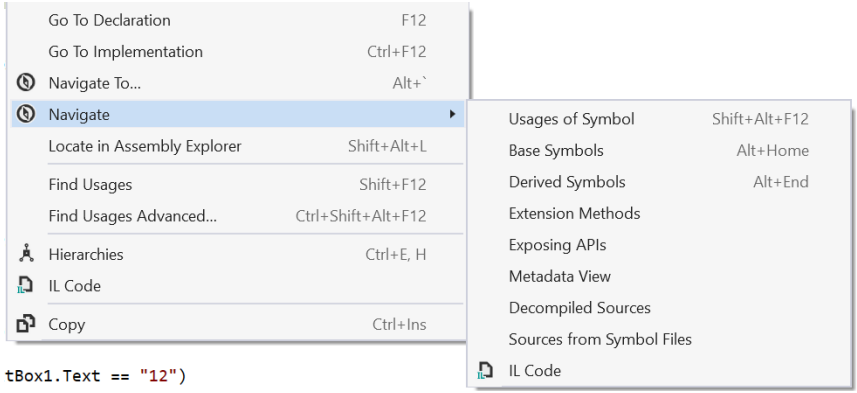


Рис. 5. Использование навигации

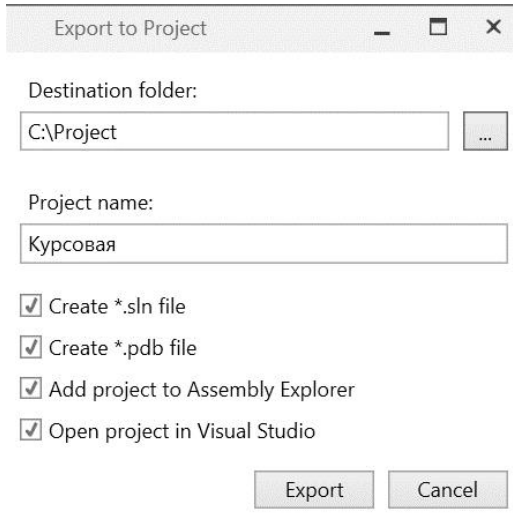


Рис. 6. Сохранение проекта Visual Studio

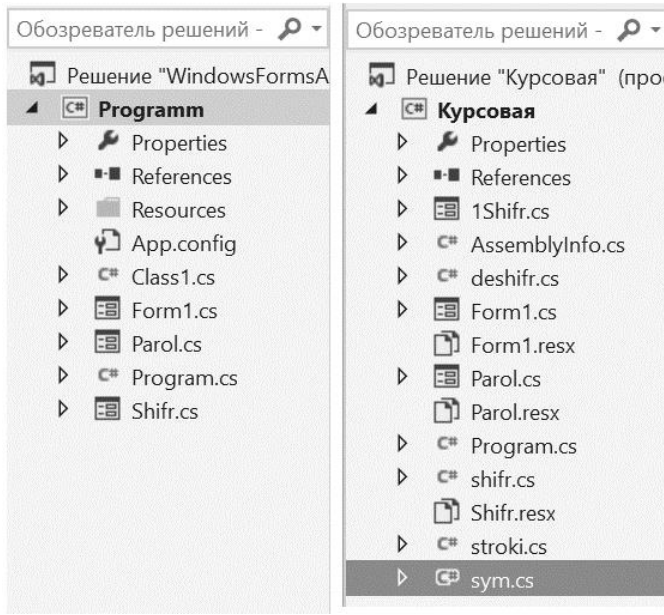


Рис. 7. Использование навигации

Задание

1. Посредством JetBrains dotPeek произвести преодоление защиты предоставленной программы «Курсовая». Сохранить декомпилированный исходный код в проект Visual Studio, сделать код работоспособным.

2. Создать собственную программу .Net реализующую простейшую математическую операцию, создайте механизм защиты программы, обменяйтесь с одногруппником вашими программами, попытайтесь обойти механизмы защиты друг друга по аналогии с первым заданием сделайте код одногруппника работоспособным.

Контрольные вопросы

– Назовите основные особенности .Net приложений?

– Что такое MSIL сборка?

– Почему .Net приложения поддаются декомпиляции?

– Что такое метаданные?

– Этапы компиляции .Net приложений?

– Что такое декомпиляторы?

– Можно ли при помощи декомпилятора изменить исходный файл программы? Как происходит редактирование?

– Какие языки программирования не поддаются декомпиляции, почему?

– Как сохранить декомпилированный код в проект Visual Studio?

– Как сделать сохраненный проект Visual Studio работоспособным?

Практическая работа № 2

Исследование Windows-PE модуля программы с помощью декомпиляции

Цель работы – познакомиться с особенностями работы с дизассемблером-отладчиком, преодолеть защиту исполняемого модуля программы с его помощью.

Краткие теоретические сведения

Применение дизассемблера позволяет напрямую работать с последовательностью инструкций целевого процессора, в которую исходный текст программы был превращён при дизассемблировании. Однако для этого необходимо обладать навыками по работе с ассемблерным кодом, а также знать особенности и функции дизассемблера.

OllyDbg является отладчиком уровня ассемблера для операционных систем Windows. Предназначен для модификации и анализа откомпилированных библиотек и исполняемых файлов, работающих в режиме пользователя.

Для преодоления защиты программы дизассемблером, используются следующие способы:

– работа с флагами и регистрами;

– создание условных точек останова;

– поиск строковых сообщений;

– создание точки останова на память процессора;

– работа с командами вызова функции, условного и безусловного переходов;

– создание собственной функции для поиска необходимого участка дизассемблерного кода.

Ход работы

Часть 1

OllyDBG можно скачать с ресурса http://www.ollydbg.de/. Для начала работы с программой необходимо во вкладке File выбрать «Open», либо нажать клавишу F3 (рис. 8).

Рис. 8. Добавление нового файла

Далее указать путь к необходимому файлу «Program2» и выбрать его (рис. 8).

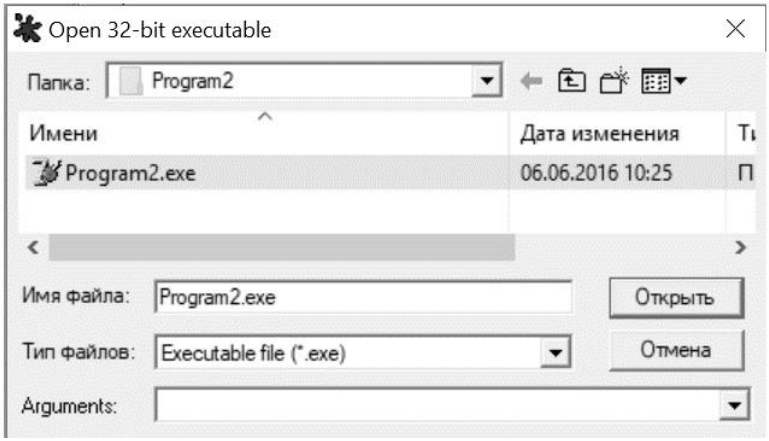


Рис. 8. Выбор файла

Откроется окно работы с программой, состоящее из следующих полей:

– окно дизассемблированного кода и окно дампа памяти (рис. 9);

– окно регистров и окно стека (рис. 10).

Для удобства работы с программой предусмотрена подсветка синтаксиса, чтобы ее включить, необходимо нажать правой кнопкой мыши на окне работы программы, далее выбрать Appearance → Highlightingc → Jumps'n'calls.

После чего появится подсветка наиболее важных для исследования команд: вызова, возврата, условных переходов.

Для запуска отладки необходимо перейти на панель, представленную на рис. 12.

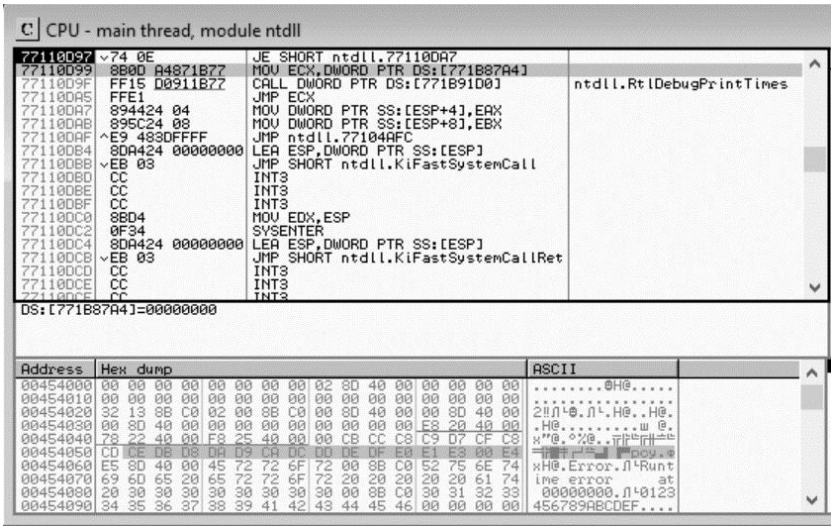


Рис. 9. Дизассемблированный код и дамп памяти

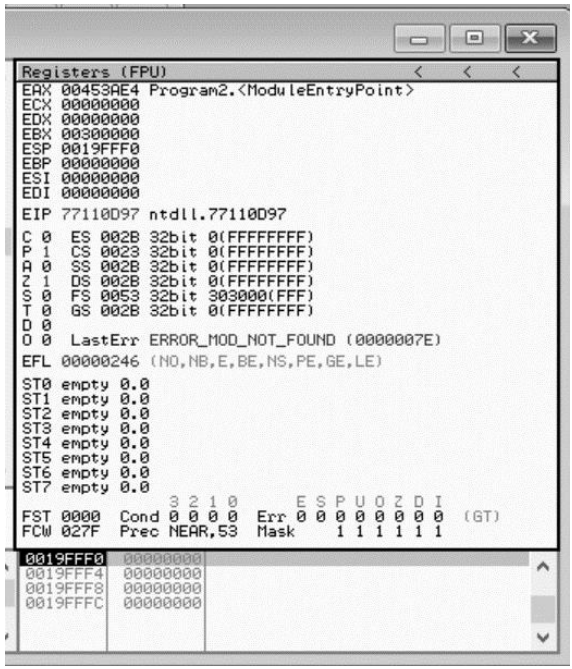


Рис. 10. Окно значения регистров и флагов

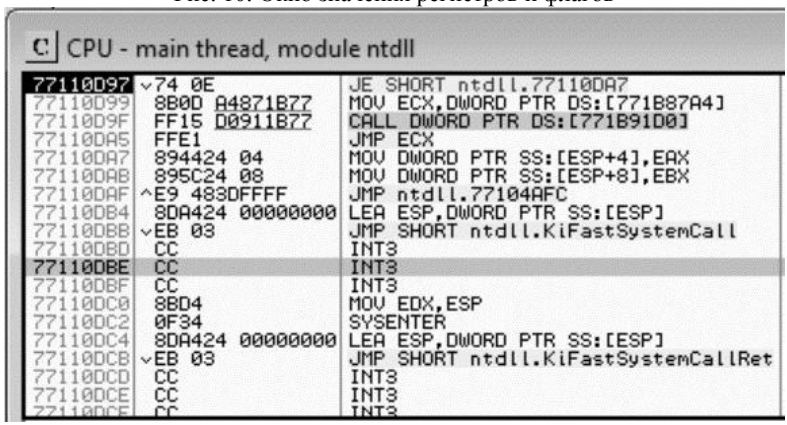


Рис. 11. Подсветка синтаксиса

В появившемся окне ввода пароля нужно ввести любые символы, чтобы получить сообщение об ошибке (рис. 13).

Чтобы обойти защиту, можно использовать несколько способов.

Первый: используя сообщение об ошибке, для этого необходимо проделать следующие действия: правая кнопка мыши → Search for $→ All referenced text strings, откроется окно со всеми строками что есть в программе (рис. 14), в которых необходимо найти нужную строку с сообщением об ошибке: правая кнопка мыши → Search for text.

Рис. 12. Запуск отладки

Далее в строку необходимо ввести первое слово из сообщения об ошибке, если оно не находится, стоит попробовать ввести другие слова из сообщения (рис. 15).

Нужная строка найдена (рис. 16).

Нажав два раза по строке, осуществляется переход в процедуру вызова данной строки (рис. 17).

Чтобы проверит та ли эта процедура, что нужна, необходимо поставит точку останова: правая кнопка мыши → BreakPoint → Toggle, либо два щелчка по адресу где расположена функция.

Запустить отладку, ввести пароль. Программа должна остановится и в отладчике появится сообщение об ошибке, это значит, что найденная команда была верной.

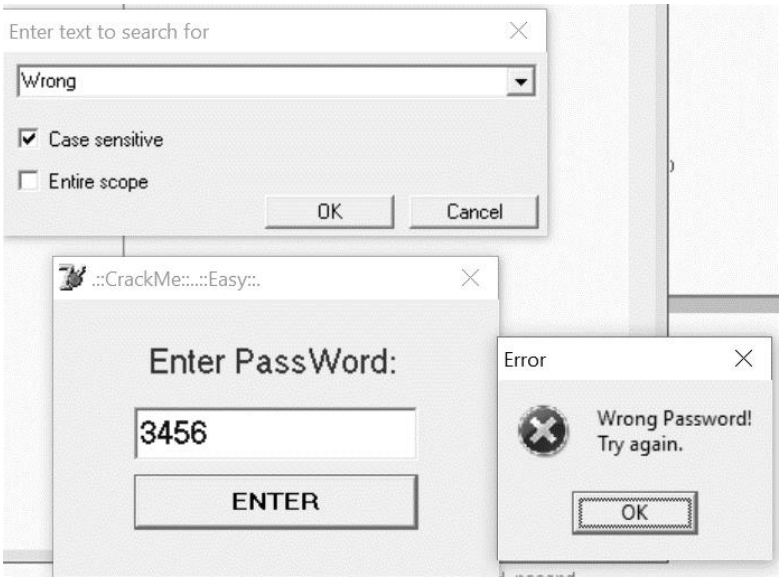


Рис. 13. Сообщение об ошибке

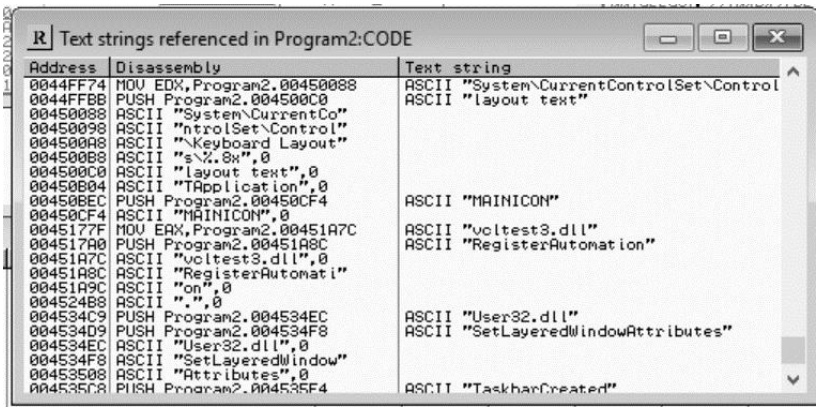


Рис. 14. Поиск текстовой строки

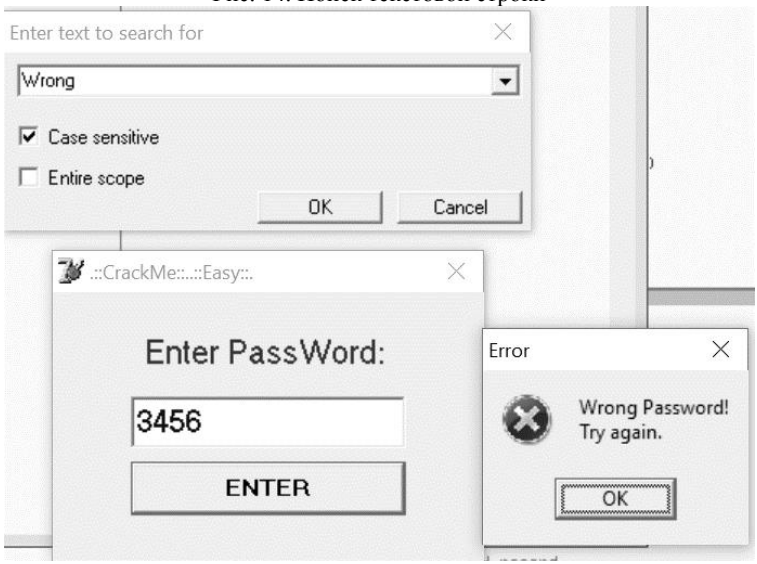


Рис. 15. Окно поиска строки

Для начала стоит попробовать найти сам пароль в коде. Обратив внимание на строку под адресом 004537DF, стоит попробовать ввести данную строку в окно ввода пароля, перед этим не забыв убрать точку останова. Вход произошёл успешно (рис. 18).

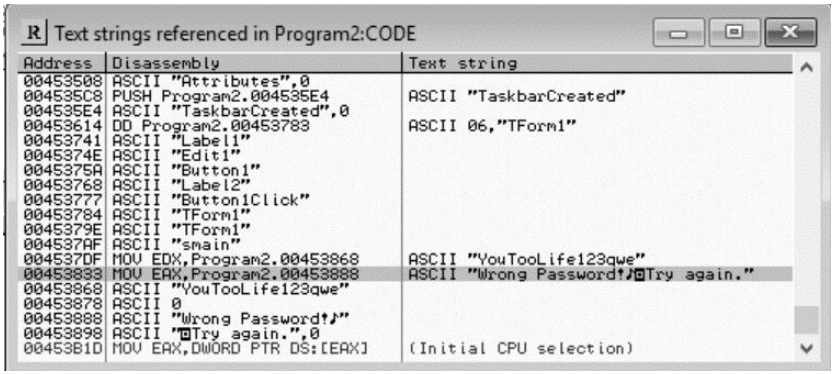


Рис. 16. Нужная строка найдена

Второй способ: так как зачастую пароль не хранится в открытом виде, то необходимо найти команду, отвечающую за сравнение введенного пароля с правильным. За это отвечают команды сравнения, а также условного и безусловного переходов.

Благодаря подсветке, данные команды выделены желтым цветом. Две ближайшие из них расположены рядом к команде вызова сообщения об ошибке. Первая их них jmp команда безусловного перехода, необходимо выделить ее и нажать Enter, тогда произойдет переход по адресу 00453833.

Рис. 17. Переход в нужный участок кода

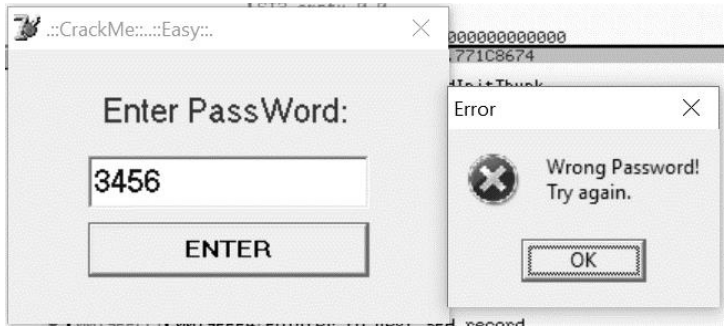


Рис. 18. Успешное преодоление защиты

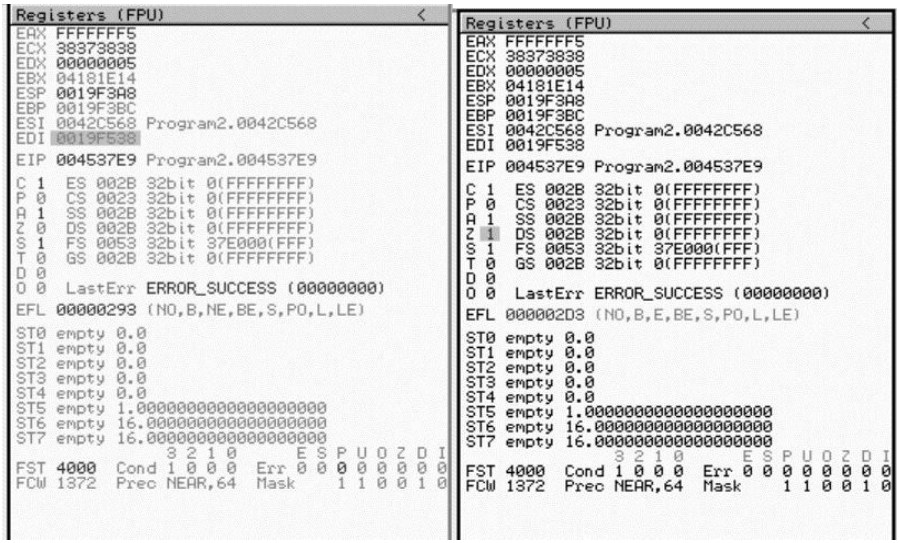
Как можно заметить, осуществился переход через команду вызова сообщения об ошибке, следовательно, код, что находится выше данного jmp выполняется в случае верного ввода пароля.

Поднимемся выше до команды jnz и поставим на ней точку останова, запустим отладку, снова введем пароль, произойдет переход в отладчик на команду jnz.

Тут и происходит работа с флагами (рис. 19 (а)).

Если флаг равен 1, то программа не делает переход, если 0, то осуществляет переход по указанному адресу. Необходимо поменять значение флага с нуля на единицу (рис. 19 (б)).

Продолжим отладку и увидим, что программа больше не требует ввода пароля и не выводит сообщение об ошибке.



а) б)

Рис. 19 Окно регистров

Третий способ: команду условного перехода jnz заменить на команду безусловного перехода jmp или командой nop.

Произведем замену на jmp и прыгнем по адресу 004537EB, это тот адрес команды, что следует сразу после команды jne, для этого два раза нужно кликнуть на строку с командой, откроется окно редактирования ассемблерного кода (рис. 20, рис. 21, рис. 22).

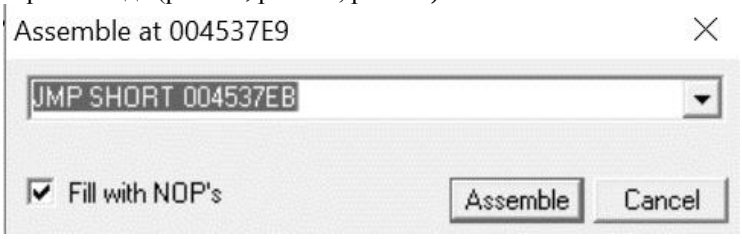


Рис. 20 Изменение значений в коде

Рис. 21 Удаление функции сравнения

Рис. 22 Замена значения

Теперь осталось сохранить изменения в программе, для этого: правая кнопка мыши → Copy to executable → all modification → copy all (рис. 23). В появившемся окне нажимаем правую кнопку мыши → Save file .

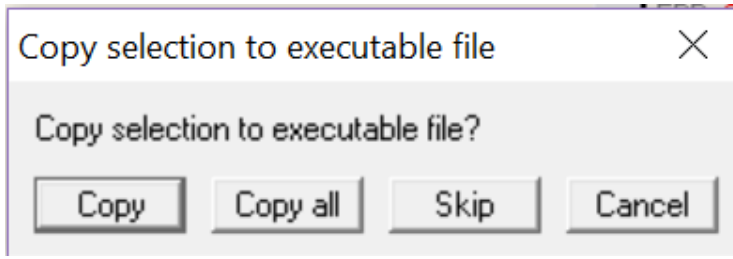


Рис. 23 Сохранение изменений

Часть 2

Запускаем программу и в поле Command вводим команду «bp TranslateMessage», данная команда bp создаст точку останова на функции TranslateMessage (рис. 24).

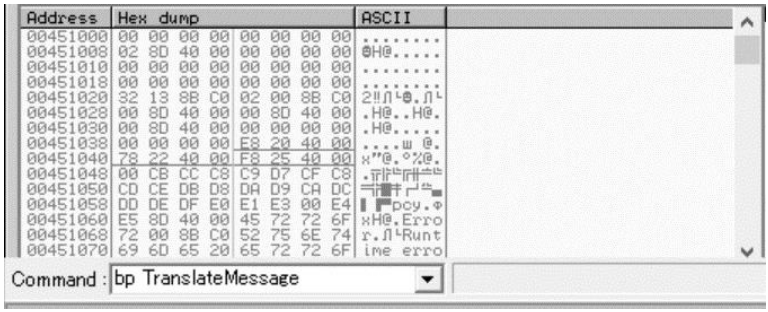


Рис. 24. Создание точки останова

В окне «Breakpoints» проверим созданную точку (рис. 25).

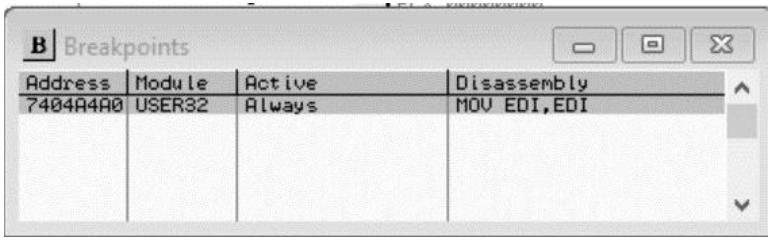


Рис. 25. Проверка наличия созданной точки

Запускаем отладку и осуществится переход в отладчик на нужную точку (рис. 26).

Далее нужно создать условие остановки программы, чтобы затем в памяти найти нужное значение. Для этого правой кнопкой мыши по строке с точкой останова → Breakpoint → Conditional log, либо сочетание клавиш Shift+F4. В появившемся окне, нужно указать параметры как на рис. 27, где msg - сообщение, 202 - код сообщения, срабатывающий, когда отпускается левая клавиша мыши, в момент выполнения программы.

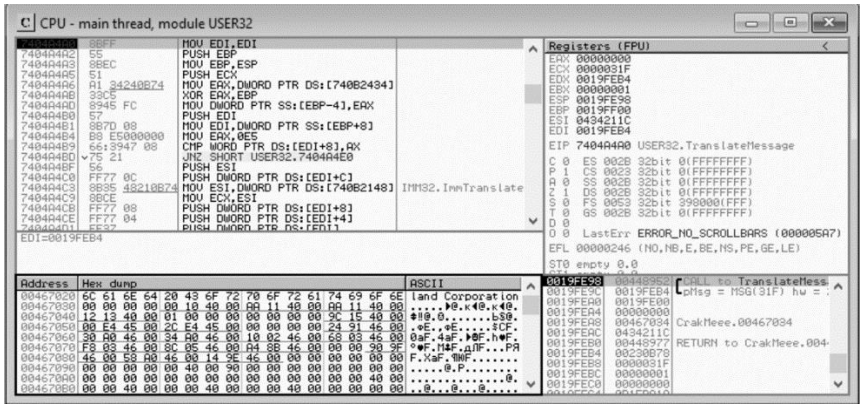


Рис. 26. Запуск отладки

После выполненных действий, действующая точка останова должна принять розовый цвет, это означает, что был установлен условный breakpoint, сработающий только тогда, когда опуститься левая кнопка мыши (рис. 28). Проверим это, запустим отладку и нажмем на окне программы, снова переход в отладчик.

Далее жмем на кнопку W, как показано на рис. 29, откроется следующее окно (рис. 30).

Во вкладке «Title» ищем строку OK → правая кнопка мыши по этой строке → Conditional log breakpoint on ClassProc, откроется уже знакомое окно, в котором нужно установить те же параметры, что и до этого. Вид окна после проделанных действий показан на рис. 30.

Дальнейший поиск будет осуществляться на основании строки, которую необходимо ввести в окно ввода пароля, поэтому запускаем отладчик, затем не нажимая на окно программы, иначе выкинет в отладчик (мы сами создали ранее такое условие), вводим наш пароль и нажимаем «OK» (рис. 31).

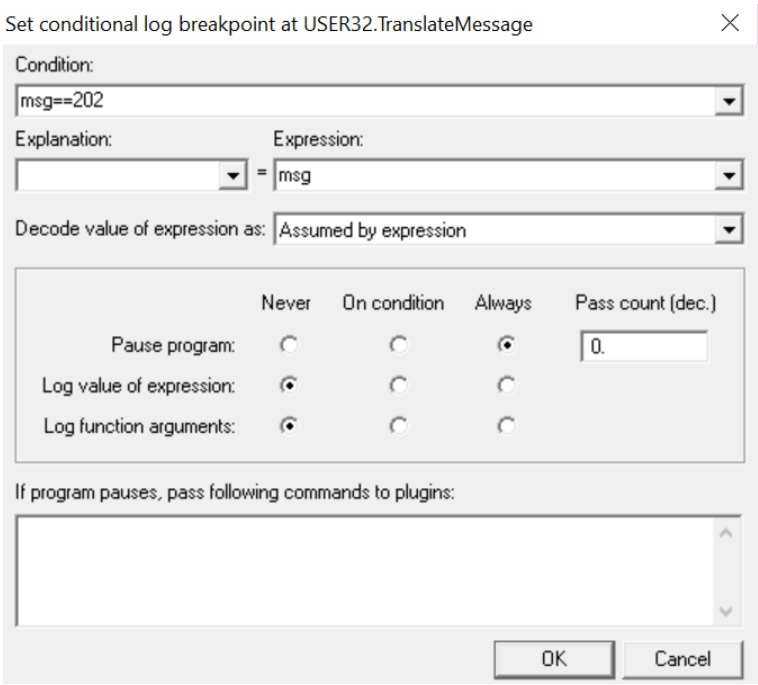


Рис. 27. Создание условной точки останова

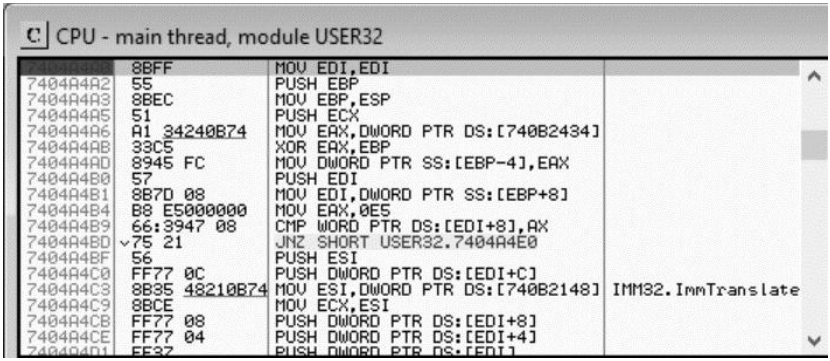


Рис. 28. Проверка точки

Рис. 28 Переход в окно Windows

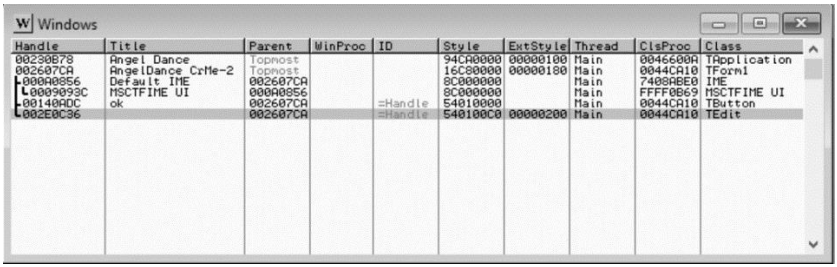


Рис 29. Окно Windows

Рис. 30. Точка останова с условием

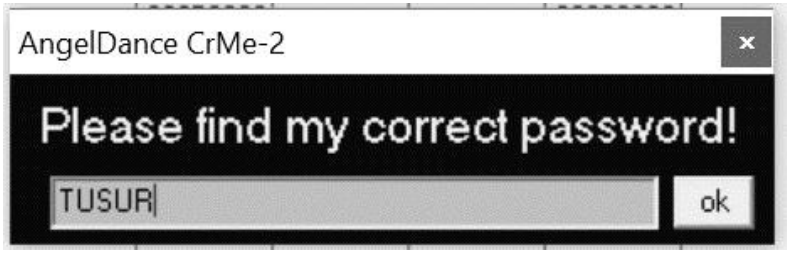


Рис. 31. Ввод пароля

На той же панели где расположена кнопка «B», нажимаем кнопку «M», в открывшимся окне правая кнопка мыши → Search. Нужно найти в памяти введенную строку (рис. 32).

После того как строка будет найдена, выделяем ее → правая кнопка мыши → Breakpoint → Memory on, access (рис. 33). Тем самым была поставлена точка останова на участок памяти и как только программа обратится по данному участку памяти, то сработает точка останова.

Теперь необходимо снять установленные до этого точки останова, для этого перейдем во вкладку Breakpoint, выделим точку и нажмем пробел.

Запускаем отладку, осуществится переход на то место в памяти, в котором начнет копироваться строка «TUSUR» (рис. 34).

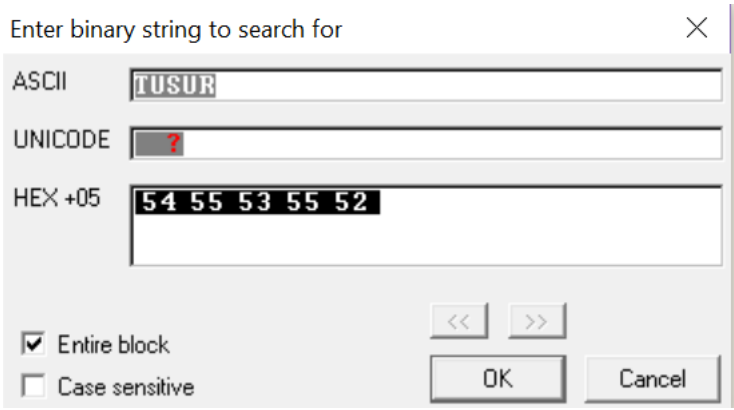


Рис. 32 Поиск пароля в строковых значениях

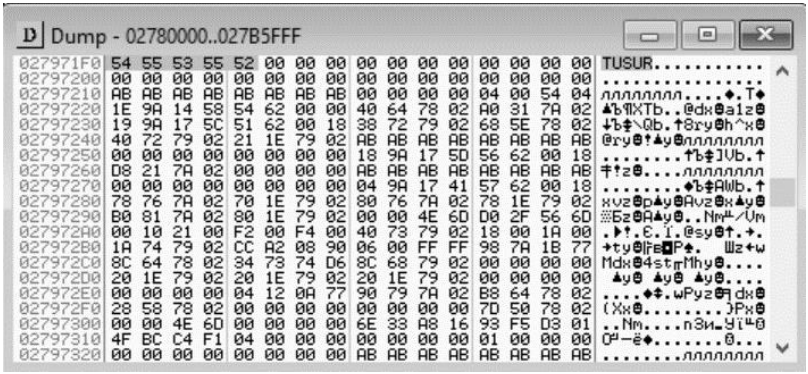


Рис. 33. Строка найдена

Затем нажмем один раз на кнопку, выделенную на рис. 35, так будет осуществлена пошаговая отладка программы. Выполним действия, показанные на рис. 36, чтобы отслеживать процесс в дампе памяти, затем продолжим пошаговую отладку. Постепенно можно увидеть, как в памяти появится строка целиком (рис. 37), выделяем его и снова ставим точку основа на память, как было уже сделано до этого (рис. 38).

Запускаем отладку, нажимаем кнопку отладки до тех пор, пока программа не остановится на точке останова (рис. 39). Остановка произойдет в том месте, где в регистр ECX заносится введенный пароль, а в регистр EBX

правильный пароль, далее можно заметить команду сравнения данных регистров. Цель достигнута, найдена место сравнения паролей, а дальнейшие действия по обходу защиты были уже рассмотрены ранее.

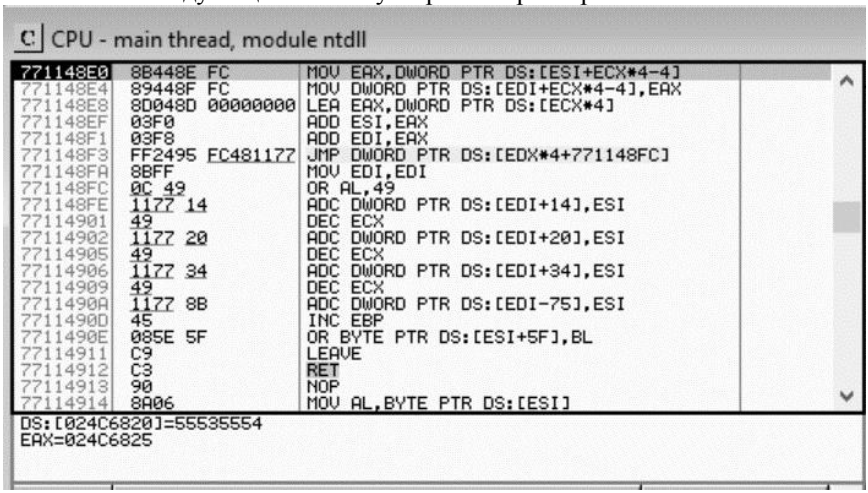


Рис. 34. Переход в нужный участок кода

Рис. 35. Пошаговая отладка

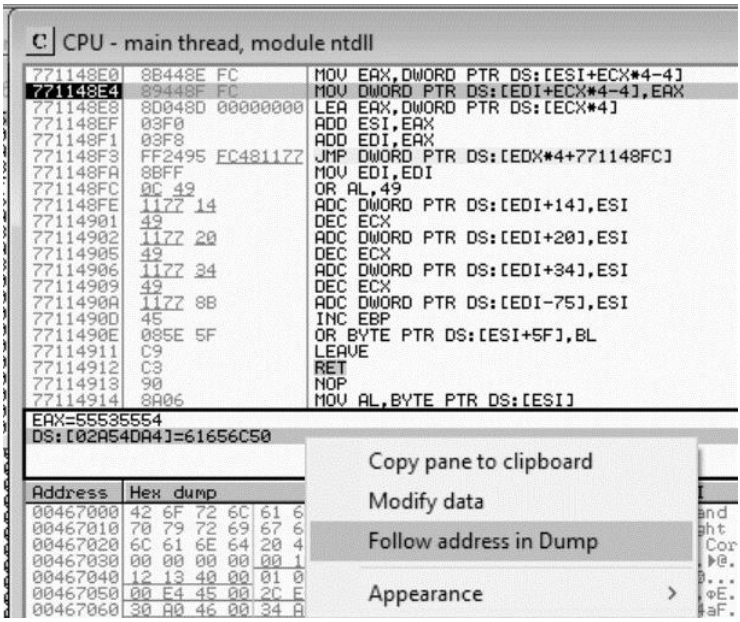


Рис. 36. Дамп памяти

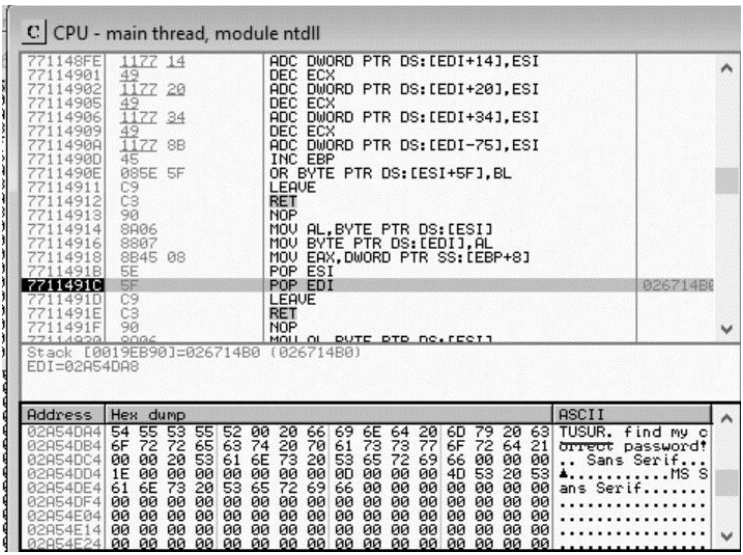


Рис. 37. Копирование пароля в память

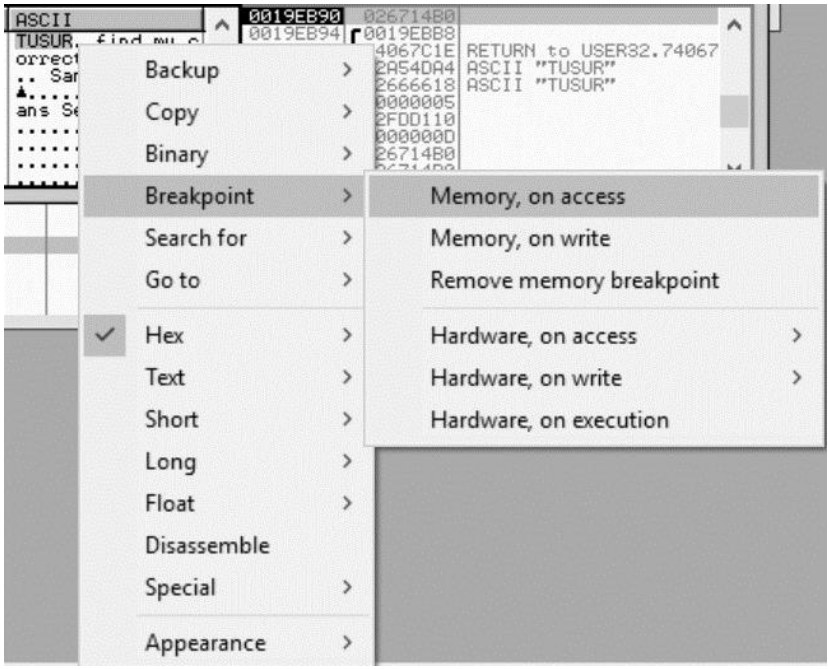


Рис. 38. Точка останова на память

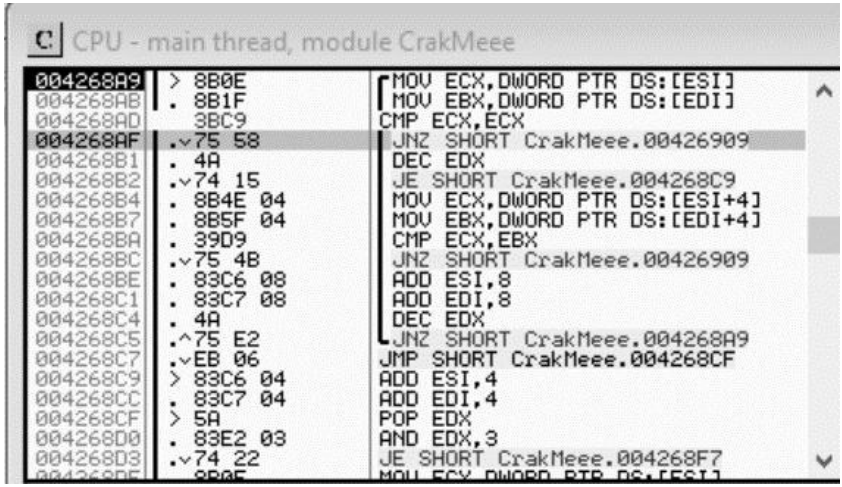


Рис. 39. Место сравнения паролей

Задание

Преодолеть защиту программы «Program2» с помощью дизассемблера. На основании полученного опыта, исправить ошибки реализации защиты в собственной программе.

Контрольные вопросы

– Что такое дизассемблер? Основные возможности?
– Что такое отладчик? Основные возможности?
– Что такое регистр? Какие регистры бывают?
– Что такое флаги?
– Какие команды ассемблера вы знаете?
– Для чего нужны контрольные точки?
– Как найти необходимое строковое сообщение?
– Какие способы преодоления защиты вы знаете?

Практическая работа № 3

Защита исполняемого модуля программы от исследования

Цель работы – изучить возможности средства защиты программных продуктов ConfuserEx, использовать данное средство для защиты собственной программы, произвести анализ эффективности поставленной защиты при помощи декомпилятора и дизассемблера.

Краткие теоретические сведения

ConfuserEx – является средством защиты с открытым исходным кодом для .Net приложений.

Предлагает расширенную защиту приложений, написанных на языках C \#, VB, F \# и других языках .NET.

Является бесплатным протектором .NET, который имеет защиту, сопоставимую с коммерческими защитниками. Открытость исходного кода, позволяет изменять его в соответствии с собственными потребностями разработчиков.

ConfuserEx работает под операционными системами Microsoft Windows 7/8/8.1/10 (32-х и 64-х битные версии).

Плагиновая система позволяет разработчикам создавать собственные средства защиты и интегрировать их в ConfuserEx, что дает возможность применять различные модульные средства защиты в одном приложении и значительно повышать безопасность защищаемой программы.

ConfuserEx использует параметры, отраженные в исполняемом файле проекта. К таким параметрам относится сборка .NET, получаемая на этапе компиляции и хранящаяся в исходном файле программы.

Обладает следующими функциями защиты:

– WPF (Windows Presentation Foundation) renaming (переименование типов, методов и полей, к которым имеются обращения из WPF);

– control flow obfuscation (запутывание потока управления (противодействие декомпиляторам);

– method reference hiding (скрытие логической связи между фрагментами кода);

– anti-debuggers/profilers (противодействие отладке);

– anti-memory dumping (противодействие дампу памяти);

– anti-tampering (method encryption) (контроль целостности (шифрование));

– constant encryption (шифрование констант);

– resource encryption (шифрование ресурсов);
– compressing packer (сжатие исполняемого файла);
– reference proxy protection (защита прокси-ссылок);
– invalid metadata protection (защита метаданных);
– anti IL disassembly protection (противодействие дизассемблированию сборки на языке IL).

WPF – это система построения клиентских приложений, как автономных так и браузерных, с акцентом на интерфейс программы, визуализацию и расчетом на возможности современного графического оборудования.

Защита подразделяется на четыре уровня: minimum, normal, aggressive, maximum. Накладываемые функции защиты оставляют отпечаток на производительность приложения, поэтому не стоит использовать максимальный уровень защиты, если в нем нет необходимости.

Соответствие функций защиты уровням защиты представлено в таб.1.

Таб. 1. Соответствие функций защиты уровням защиты

Уровень защиты	Функции
Minimum	– anti IL disassembly protection; – anti debug protection; – name protection;
Normal	– reference proxy protection; – constants protection; – resources protection;
Aggressive	– control flow protection;
Maximum	– anti tamper protection; – anti dump protection; – invalid metadata protection.

Ход работы

ConfuserEx необходимо скачать с ресурса https://github.com/yck1509/ConfuserEx/releases/tag/v1.0.0. После распаковки архива, запустите ConfuserEx, далее перейдите на вкладку «Project», нажмите «+» чтобы добавить файл для защиты. Во вкладке «Output Directory» можно

выбрать директорию, в которой будет сохранен защищенный файл или оставить ее по умолчанию (рис. 40).

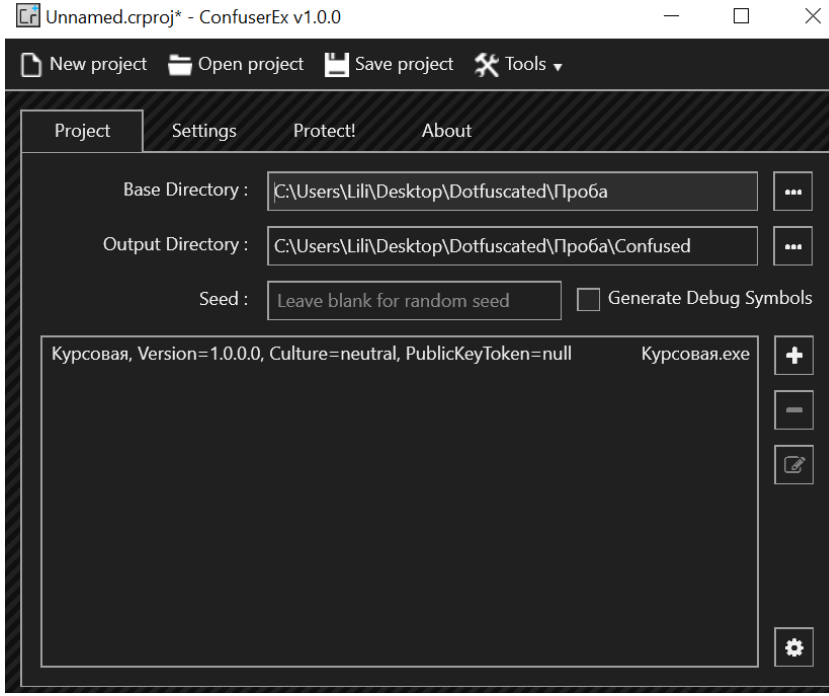


Рис. 40. Вкладка «Project»

Если нажать на кнопку с шестеренкой, откроется окно «Advanced Settings», которое позволяет разработчикам разрабатывать собственный модули защиты и интегрировать их в ConfuserEx в качестве плагинов (рис. 41).

Вкладка «Settings» определяет какими методами будет защищена программа. Пункт «Packer» производит сжатие исходного файла.

Compressor – это упаковщик, уменьшающий размер вывода с использованием алгоритма сжатия LZMA.

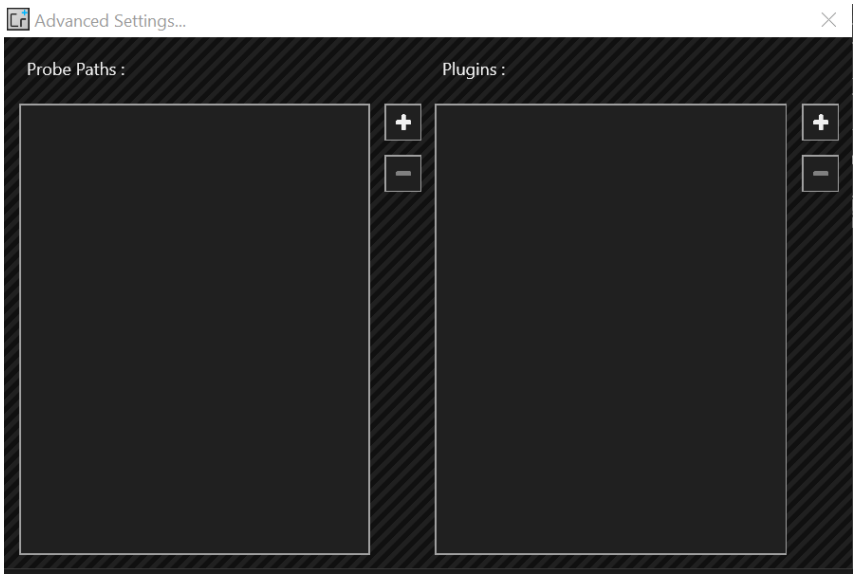


Рис. 41. Окно «Advanced Settings»

«Modules» позволяет создавать общие настройки «Global settings» ко всем защищаемым файлам или же к каждому файлу отдельно. Выберите свою программу во вкладке «Modules» и нажмите «+», чтобы добавить правила защиты (рис. 42).

Кнопка с карандашом открывает окно с правилами защиты (рис. 43). В поле «Preset» можно выбрать следующие уровни защиты: none, minimum, normal, aggressive, maximum. Поле «Protection» отвечает за выбор правил защиты:

– anti ildasm (anti IL disassembly protection); противодействие дизассемблированию сборки на языке IL;

– anti tamper (anti tamper protection); контроль целостности (шифрование);

– constants (constants protection); шифрование констант;

– ctrl flow (control flow protection); скрытие логической связи между фрагментами кода;

– anti dump (anti dump protection); противодействие дампу памяти;

– anti debug (anti debug protection); противодействие отладке;

– invalid metadata (invalid metadata protection); защита метаданных;

– ref proxy (reference proxy protection); защита прокси-ссылок;

– resources (resources protection); шифрование ресурсов;

– rename (name protection). переименование констант;
– compressing packer; сжатие итогового файла.

Выберите необходимые правила и нажмите кнопку «Done».

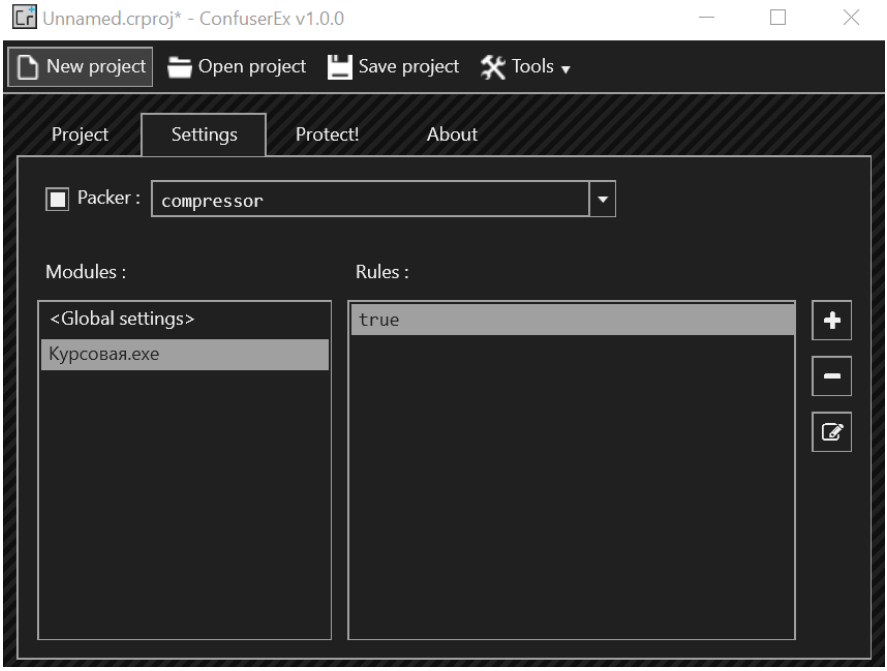


Рис. 42. Вкладка «Settings»

После того как все необходимые правила защиты выбраны, можно перейти на вкладку «Protect!» для реализации функций защиты (рис. 44). После нажатия кнопки «Protect!», произойдет защита вашей программы в соответствии с выбранными правилами защиты.

Задание

1. Посредством ConfuserEx произвести защиту собственной программы, используя различные функции защиты.
2. Поменяться с одногруппником программами и произвести анализ эффективности поставленной защиты, сделать вывод.

Контрольные вопросы

– Что такое протектор?

– Каким методам исследования программных средств противодействует ConfuserEx?

– Для чего необходимо сжатие исполняемого файла?

– Зачем нужны уровни защиты?

– Какие функции защиты используются в ConfuserEx?

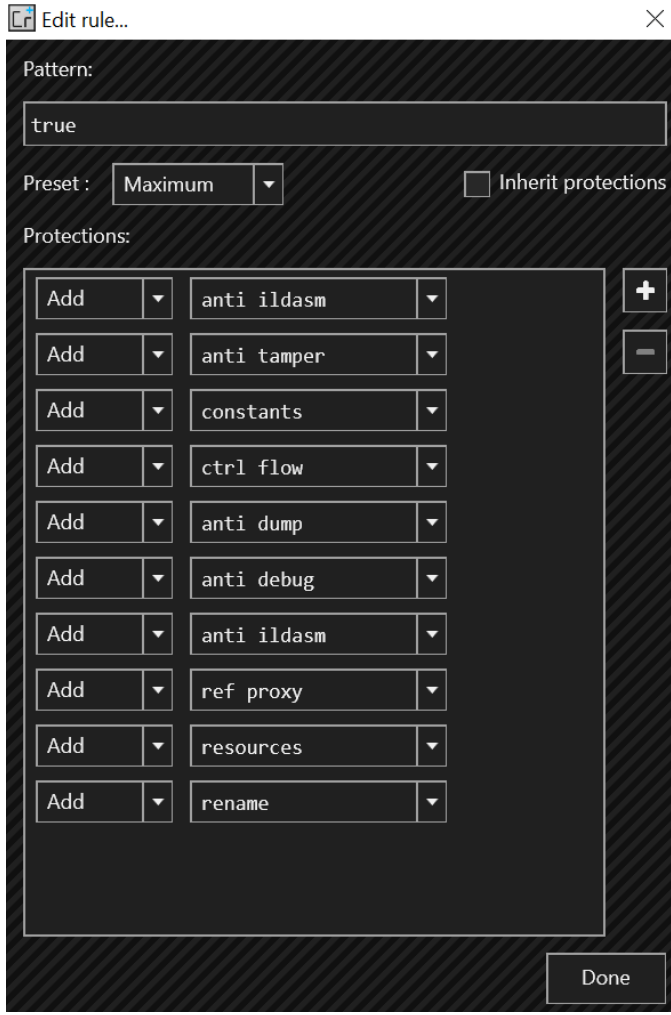


Рис. 43. Вкладка «Edit rule»

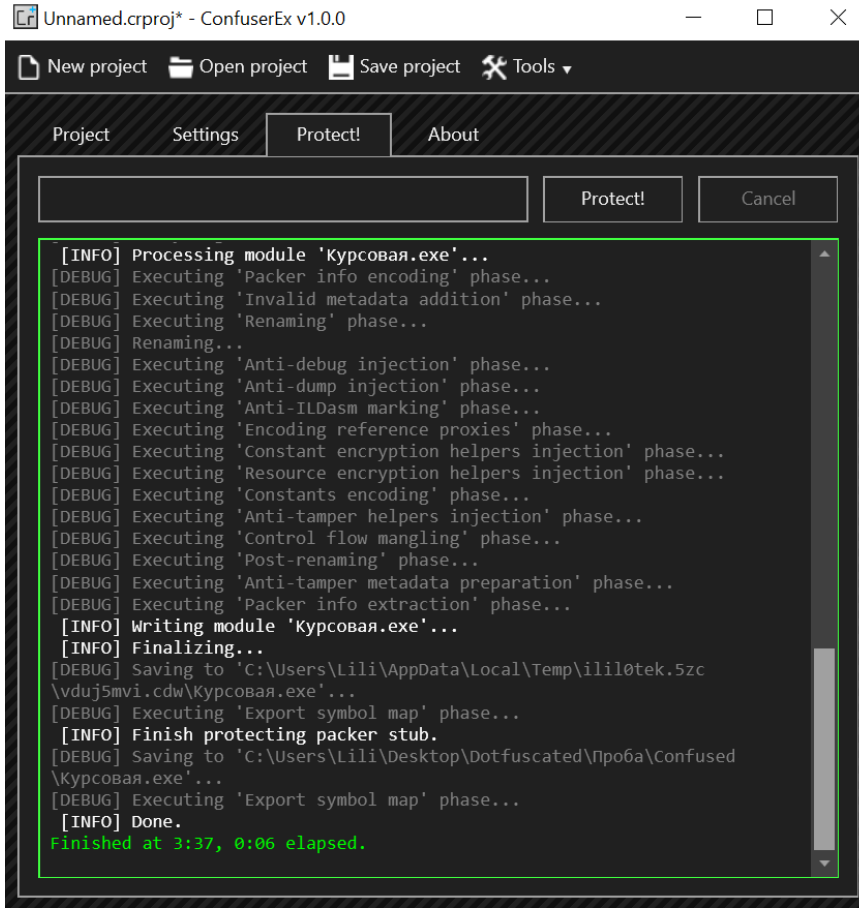


Рис.44. Вкладка «Project»

Практическая работа № 4

Защита исполняемого модуля программы от исследования
Часть 1. Обфускация

Обфускация - приведение исполняемого кода к виду, сохраняющему функциональность программы, но затрудняющему анализ и понимание алгоритмов работы.

Методы обфускации

1 Лексическая обфускация. Данный вид обфускации заключается в форматировании кода программы, изменении его структуры, таким образом, чтобы он стал нечитабельным, менее информативным и трудным для изучения.

Лексическая обфускация включает в себя:

- удаление всех комментариев в коде программы, или изменение их на дезинформирующие;
- удаление различных пробелов, отступов, которые обычно используют для лучшего визуального восприятия кода программы;
- замену имен идентификаторов (имен переменных, функций и т.д.), на произвольные длинные наборы символов;
- добавление различных лишних (мусорных) операций;
- изменение расположения блоков (функций, процедур).

2 Вычислительная обфускация. Изменение касающиеся главной структуры потока управления. К ним можно отнести:

- расширение условий циклов;
- добавление недостижимого кода;
- добавление избыточных операций.

Ход работы

1. Выбрать код программы по варианту (варианты в конце).
2. Скомпилировать программу, проверить работоспособность.

2.1 Запустить MS Visual Studio.

2.2 Создать проект Visual C++/Win32/Консольное приложение Win32 с именем «test» (рис.1).

2.3 Вставить код программы, полученный в пункте 1, в «test.cpp».

2.4 Нажать F5 для компиляции и запуска программы.

3. Осуществить лексическую обфускацию.

3.1 Заменить названия переменных на трудночитаемые, не несущие смысла (таб. 2).

Таб. 2. Замена символов

До	После

#include <stdafx.h>
#include <stdio.h>
// главная функция
int main()
{
// номер варианта int q = 1;
switch(q)
{
// пусто (вариант 0)
case 0:
printf("null\n");
break;
// первый вариант
case 1:
for (int i=0;i<=1;i++)
printf("variant 1\n");
break;
// второй вариант
case 2:
printf("variant 2\n");
break;
// другие варианты
default:
printf("other variant\n");
}
scanf("%i",&q);
return 0;
}
	#include <stdafx.h>
	#include <stdio.h>
	// главная функция
	int main()
	{
	// номер варианта
	int xX03123012 = 1;
	switch(xX03123012)
	{
	// пусто (вариант 0)
	case 0:
	printf("null\n");
	break;
	// первый вариант
	case 1:
	for (int weq83efas_23=0;weq83efas_23<= 1;weq83efas_23 ++)
	printf("variant 1\n");
	break;
	// второй вариант
	case 2:
	printf("variant 2\n");
	break;
	// другие варианты
	default:
	printf("other variant\n");
	}
	scanf("%weq83efas_23",&xX03123012);
	return 0;
	}

Комментарии к коду:

– заменено название переменной q на xX03123012;

– заменено название переменной i на weq83efas_23.

3.2 Использовать шестнадцатеричное представление чисел (пример приведен в описании лексической обфускации).

Таб. 3. Замена символов

До	После

#include <stdafx.h>
#include <stdio.h>
// главная функция
int main()
{
// номер варианта
int xX03123012 = 1;
switch(xX03123012)
{
// пусто (вариант 0)
case 0:
printf("null\n");
break;
// первый вариант
case 1:
for (int
weq83efas_23=0;weq83efas_23<=1;
weq83efas_
23++)
printf("variant
1\n");
break;
// второй вариант
case 2:
printf("variant 2\n");
break;
// другие варианты
default:
printf("other variant\n");
}
scanf("%weq83efas_23",&xX031230
12)
;
return 0;
}
	#include <stdafx.h>
	#include <stdio.h>
	// главная функция
	int main()
	{
	// номер варианта
	int xX03123012 = 0x26d2+60-
	0x270d;
	switch(xX03123012)
	{
	// пусто (вариант 0)
	case 0x36d-877:
	printf("null\n");
	break;
	// первый вариант
	case -3550+0xddf:
	for (int
	weq83efas_23=0;weq83efas_23<=1;
	weq83efas_23
	++)
	printf("variant 1\n");
	break;
	// второй вариант
	case 0xca4-3233+1:
	printf("variant 2\n");
	break;
	// другие варианты
	default:
	printf("other variant\n");
	}
	scanf("%weq83efas_23",&xX03123
	012);
	return xX03123012-1;
	}

Комментарии к коду: применено шестнадцатеричное представление чисел, вместо десятичного (например, 0x26d2+60-0x270d=9938+60-9997=1, 2=0xca4-3233+1);

3.3 Убрать пробелы и отступы и заменить комментарии на дезинформирующие.

Таб. 3. Замена символов

До	После

#include <stdafx.h>
#include <stdio.h>
// главная функция
int main()
{
// номер варианта
int xX03123012 = 0x26d2+60-0x270d;
switch(xX03123012)
{
// пусто (вариант 0)
case 0x36d-877:
printf("null\n");
break;
// первый вариант
case -3550+0xddf:
for (int
weq83efas_23=0;weq83efas_23<=1;
weq83efas_
23++)
printf("variant
1\n");
break;
// второй вариант
case 0xca4-3233+1:printf("variant 2\n");
break;
// другие варианты
default:
printf("other variant\n");
}
scanf("%weq83efas_23",&xX03123012)
;
return xX03123012-1;
}
	// модуль распознавания речи
	#include <stdafx.h>
	#include <stdio.h>
	Int main()
	{
	/*номер записи*/
	int
	xX03123012=0x26d2+60-
	0x270d;switch(xX03123012){
	/*голос
	шпиона*/case 0x36d-
	877:printf("null\n");
	break;
	/* голос офицера*/
	case -3550+0xddf:
	for (int
	weq83efas_23=0;weq83efas_23<=1;
	weq83efas_23
	++)printf("variant 1\n");
	break;
	/*мой голос*/
	case 0xca4-3233+1:printf
	("variant 2\n");
	break;
	/*другие голоса*/
	default:printf("other
	variant\n");}return xX03123012-1;}

3.4 Скомпилировать программу клавишей F5 и проверить ее функциональность. При появлении ошибок необходимо восстановить код согласно варианту и повторить лексическую обфускацию.

3.5 Сделать резервную копию кода программы (например, в текстовом файле).

4. Осуществить вычислительную обфускацию.

4.1 Расширить условия циклов.

Таб. Изменения кода программы

До	После

// модуль распознавания речи
#include <stdafx.h>
#include <stdio.h>
int
main()
{/*номер записи*/int
xX03123012=0x26d2+60-
0x270d;switch(xX03123012){
/*голос
шпиона*/case 0x36d-
877:printf("null\n");
break;
/* голос офицера*/case -3550+0xddf:
for (int
weq83efas_23=0;weq83efas_23<=1;weq
83efas_
23++)printf("variant 1\n");
break;
/*мой голос*/case 0xca4-
3233+1:printf("variant
2\n");break;
/*другие голоса*/
default:printf("other
variant\n");}
return xX03123012-1;}
	// модуль распознавания речи
	#include <stdafx.h>
	#include <stdio.h>
	int
	main(){/*номер записи*/int
	xX03123012=0x26d2+60-
	0x270d;switch(xX03123012)
	{/*голос
	шпиона*/case 0x36d-
	877:printf("null\n");
	break;
	/* голос
	офицера*/case -3550+0xddf: for
	(int
	weq83efas_23=0,
	j33floor1=2;weq83efas_23<=1
	&&
	j33floor1==0x2;weq83efas_23++)
	printf("varia
	nt 1\n");
	break;/*мой голос*/case 0xca4-
	3233+1:printf("variant 2\n");
	break;
	/*другие
	голоса*/default:printf("other
	variant\n");}scanf
	("%i",&xX03123012);
	return xX03123012-1;}

Комментарии к коду: в цикл for была включена переменная j33floor1, равная

двум всегда, то есть, добавленное условие j33floor1==0x2 будет всегда выполняться.

1.2 Добавить избыточные операции.

Таб. 3. Замена символов

До	После

// модуль распознавания речи
#include <stdafx.h>
#include <stdio.h>
int
main(){/*номер записи*/int
xX03123012=0x26d2+60-
0x270d;switch(xX03123012)
{/*голос
шпиона*/case 0x36d-
877:printf("null\n");
break;/* голос офицера*/
case -3550+0xddf:
for (int
weq83efas_23=0,
j33floor1=2;weq83efas_23<=1 &&
j33floor1==0x2;weq83efas_23++)
printf("var
iant 1\n");
break;/*мой голос*/case 0xca4-
3233+1:printf("variant
2\n");break;/*другие
голоса*/default:printf("other
variant\n");}scanf
("%i",&xX03123012);return
xX03123012-1;}
	// модуль распознавания речи
	#include <stdafx.h>
	#include <stdio.h>
	int
	main(){/*номер записи*/int
	xX03123012=0x26d2+60-
	0x270d;switch(xX03123012){
	/*голос шпиона*/
	case 0x36d-
	877:printf("null\n");
	break;
	/* голос офицера*/
	case -3550+0xddf:for (int
	weq83efas_23=0, j33floor1=2;
	weq83efas_23<=1 &&
	j33floor1==0x2;
	weq83efas_23++){ j33floor1=
	j33floor1*(21+0x4)/(15+0xa);
	printf("variant
	1\n");}break;
	/*мой голос*/case 0xca4-
	3233+1:printf("variant
	2\n");break;/*другие
	голоса*/default:printf("other
	variant\n");}scanf
	("%i",&xX03123012);return
	xX03123012-1;}

Комментарии к коду: добавлена избыточная операция j33floor1=j33floor1*(21+0x4)/(15+0xa) (эта операция никак не изменяет значение j33floor1, так как множитель всегда равен единице).

1.3 Добавить недостижимый код.

Таб. 3. Замена символов

До	После

// модуль распознавания речи
#include <stdafx.h>
#include <stdio.h>
int
main(){/*номер записи*/int
xX03123012=0x26d2+60-
0x270d;switch(xX03123012){/*голос
шпиона*/case 0x36d-
877:printf("null\n");break;/* голос
офицера*/case -3550+0xddf:for (int
weq83efas_23=0, j33floor1=2;
weq83efas_23<=1 &&
j33floor1==0x2;
weq83efas_23++){ j33floor1=
j33floor1*(21+0x4)/(15+0xa);
printf("variant 1\n");}break;/*мой
голос*/case 0xca4-
3233+1:printf("variant
2\n");break;/*другие
голоса*/default:printf("other
variant\n");}scanf
("%i",&xX03123012);return
xX03123012-1;}
	// модуль распознавания речи
	#include <stdafx.h>
	#include <stdio.h>
	int
	main()
	{/*номер записи*/
	int
	xX03123012=0x26d2+60-
	0x270d;
	switch(xX03123012)
	{/*голос шпиона*/
	case 0x36d877:printf("null\n");
	break;/* голос
	офицера*/case -3550+0xddf:for (int
	weq83efas_23=0, j33floor1=2;
	weq83efas_23<=1 &&
	j33floor1==0x2;
	weq83efas_23++){ j33floor1=
	j33floor1*(21+0x4)/(15+0xa);if
	(j33floor1==0x13-2)
	weq83efas_23++;printf("variant
	1\n");}
	break;
	/*мой голос*/
	case 0xca4-3233+1:printf("variant
	2\n");
	break;
	/*другие голоса*/
	default:printf("other
	variant\n");}
	scanf
	("%i",&xX03123012);return
	xX03123012-1;}

Комментарии к коду: добавлен недостижимый код в цикл if (j33floor1==0x13-2) weq83efas_23++ (условие никогда не выполнится, так как в данной

программе j33floor1 равен 2 всегда).

4.4 Скомпилировать программу клавишей F5 и проверить ее функциональность. При появлении ошибок необходимо восстановить код из резервной копии и повторить вычислительную обфускацию.

4.5 Сохранить проект.

5. Поменяться обфусцированным кодом с одногруппником.

6. Попытаться разобраться в коде программы, полученном от одногруппника, и привести код к виду, близкому к исходному (под исходным видом понимается читабельный код с пробелами и отступами, без мусорных операций, с вашими подробными комментариями того, что делает программа).

7. Скомпилировать программу. Убедиться, что программа выполняет требуемые функции. В случае ошибок при компиляции, повторить пункт 6.

8. Ответить на вопросы.

Вопросы:

1. Что такое обфускация?
2. Зачем применяют обфускацию?
3. Какие методы обфускации рассмотрены в данной работе?
4. В чем заключается лексическая обфускация?
5. В чем заключается вычислительная обфускация?
6. Есть обычный код программы и обфусцированный код этой же программы. Каждый код скомпилировали и создали исполняемые файлы программ. Как вы думаете, какая программа потребует больше вычислительных ресурсов при выполнении? Обоснуйте свой ответ.

Варианты:

1)

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{
int X;
```

```
cout<<"Vvedite cifru: ";
cin>>X;
// По-шаговые вычесления
for(int i(1) ; i < 10 ; i++)
{
X = X * i;
cout<<endl<<"Resultat: "<<X;
}
cin>>X;
}
```

2)

```
#include <stdafx.h>
#include<iostream>
#include<conio.h>
using std::cout;
using std::endl;
using std::cin;
const int n = 5;
void main()
{
int mas[n];
cout<<"Vvedite masiv: \n";
for (int i = 0; i<n; i++)
{
cin>>mas[i];
}
int i = 0;
do{
mas[i] = mas[i]+5;
i++;
}while(i<5);
cout<<"\nMasiv posle dobavleniya:\n";
for (i = 0; i<n; i++)
{
cout<<mas[i]<<"\t";
}
_getch();
```

```
}

3)
#include <stdafx.h>
#include <iostream>
using namespace std;
int main()
{
int i; // счетчик цикла
int sum = 0; // сумма чисел от 1 до 1000.
setlocale(0, "");
for (i = 1; i <= 1000; i++) // задаем начальное значение 1, конечное 1000 и
задаем шаг цикла - 1.
{
sum = sum + i;
}
cout << "Сумма чисел " << sum << endl;
cin >> i;
return 0;
}
4)
#include "stdafx.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
int speed = 5, count = 1;
while ( speed < 60 )
{
speed += 10; // приращение скорости
cout << count <<"-speed = " << speed << endl;
count++; // подсчёт повторений цикла
}
system("pause");
return 0;
}
5)
#include "stdafx.h"
```

```
#include <iostream>
#include <ctime>
using namespace std;
int main(int argc, char* argv[])
{
srand( time( 0 ) );
int unknown_number = 1 + rand() % 10; // загадываемое число
int enter_number; // переменная для хранения введённого числа
cout << "Enter number [1:10] : "; // начинаем отгадывать
cin >> enter_number;
while ( enter_number != unknown_number )
{
cout << "Enter number [1:10] : ";
cin >> enter_number; // продолжаем отгадывать
}
cout << "You win!!!\n";
system("pause");
return 0;
}
```

6)

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
int counter_even = 0;
for (int count = 2; count <= 50; count += 2) // заголовок цикла
counter_even ++; // подсчёт чётных чисел
cout << "number of even numbers = " << counter_even << endl;
system("pause");
return 0;
}
```

7)

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;
int main(int argc, char* argv[])
```

```
{
srand(time(0));
int balance = 8; // баланс
do // начало цикла do while
{
cout << "balance = " << balance << endl; // показать баланс
int removal = rand() % 3; // переменная, для хранения вычитаемого
значения
cout << "removal = " << removal << endl; // показать вычитаемое
значение
balance -= removal; // управление условием
}
while ( balance > 0 ); // конец цикла do while
system("pause");
return 0;
}
```

8)

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
cout << "Enter the first limit: "; // начальное значение из интервала
int first_limit;
cin >> first_limit;
cout << "Enter the second limit: "; // конечное значение из интервала
int second_limit;
cin >> second_limit;
int sum = 0, count = first_limit;
do
{
sum += count; // наращивание суммы
count++; // инкремент начального значения из задаваемого интервала
} while (count <= second_limit); // конец цикла do while
cout << "sum = " << sum << endl; // печать суммы
system("pause");
 return 0;
}
```

Часть 2. Деобфускация

Деобфускация – процесс обратный обфускации, т.е. приведение кода к виду, близкому к исходному. Деобфускация предполагает оптимизацию кода. В процессе обфускации в программный код часто производиться добавление лишних операций, они обычно никоим образом не влияют на результаты работы самой программы, и предназначены для усложнения процесса изучения кода программы потусторонними лицами.

В свою очередь процесс оптимизации программного кода направлен на ликвидацию лишних операций, поэтому в частных случаях он может выступать в качестве квинтэссенции процесса деобфускации. Полностью восстановить первоначальный код не удастся, так как названия переменных, функций, классов и других объектов программы обычно изменяются, а узнать что-либо об их первоначальных значениях не представляется возможным.

Ход работы

1. Поменяться с одногруппником обфусцированным кодом, полученным выше.

2. Попытаться разобраться в коде программы, полученном от одногруппника, и привести код к виду, близкому к исходному (под исходным видом понимается читабельный код с пробелами и отступами, без мусорных операций, с вашими подробными комментариями того, что делает программа).

2.1 Удалить все комментарии, так они могут быть изменены на дезинформирующие. Расставить отступы, пробелы табуляции.

Таб.

Полученный код

// модуль распознавания речи
#include <stdafx.h> #include <stdio.h>
Int main(){/*номер записи*/int xX03123012=0x26d2+60-
0x270d;switch(xX03123012){/*голос шпиона*/case 0x36d-
877:printf("null\n");break;/* голос офицера*/case -3550+0xddf:for (int
weq83efas_23=0, j33floor1=2; weq83efas_23<=1 && j33floor1==0x2;
weq83efas_23++){
j33floor1= j33floor1*(21+0x4)/(15+0xa);if (j33floor1==0x13-2)
weq83efas_23++;printf("variant 1\n");}break;/*мой голос*/case 0xca4-
3233+1:printf("variant 2\n");break;/*другие голоса*/default:printf("other
variant\n");}scanf ("%i",&xX03123012);return xX03123012-1;}

Таб.

Результат

#include <stdafx.h>
#include <stdio.h>
int main()
{
int xX03123012=0x26d2+60-0x270d;
switch(xX03123012)
{
case 0x36d-877:
printf("null\n");
break;
case -3550+0xddf:
for (int weq83efas_23=0, j33floor1=2; weq83efas_23<=1 &&
j33floor1==0x2; weq83efas_23++)
{
j33floor1= j33floor1*(21+0x4)/(15+0xa);
if (j33floor1==0x13-2) weq83efas_23++;
printf("variant 1\n");
}
break;
case 0xca4-3233+1:
printf("variant 2\n");
break;
default:
printf("other variant\n");
}
scanf ("%i",&xX03123012);
return xX03123012-1;
}

2.2 Заменить имена переменных, классов, функций на более привычные Вам.

Результат

```
#include <stdafx.h>
#include <stdio.h>
int main()
{
int var1=0x26d2+60-0x270d;
switch(var1)
{
case 0x36d-877:
printf("null\n");
break;
case -3550+0xddf:
for (int var2=0, var3=2; var2<=1 && var3==0x2; var2++)
{
var3= var3*(21+0x4)/(15+0xa);
if (var3==0x13-2) var2++;
printf("variant 1\n");
}
break;
case 0xca4-3233+1:
printf("variant 2\n");
break;
default:
printf("other variant\n");
}
scanf ("%i",&var1);
return var1-1;
}
```

2.3 Вычислить значения выражений и заменить их полученными константами.

Таб.

Результат

#include <stdafx.h>
#include <stdio.h>
int main()
{
int var1=1;
switch(var1)
{
case 0:
printf("null\n");
break;
case 1:
for (int var2=0, var3=2; var2<=1 && var3==2; var2++)
{
var3= var3*25/25;
if (var3==17) var2++;
printf("variant 1\n");
}
break;
case 2:
printf("variant 2\n");
break;
default:
printf("other variant\n");
}
scanf ("%i",&var1);
return var1-1;
}

2.4 Упростить циклы за счет исключения расширенных условий, не влияющий на функционирование цикла. Заметим, что в условии цикла for переменная var3 сравнивается с «2». Перед выполнением цикла в var3 заносится значение «2», а в теле цикла эта переменная меняет значение на var3*25/25, то есть значение неизменно. Далее происходит сравнение «if (var3==17)», которое никогда не выполняется, ввиду того, что var3 всегда равно «2». Делаем вывод, что сравнение «var3==2» в условной части оператора лишнее, а переменная var3 не влияет на выполнение цикла, а ее

значение нигде больше не используется. Поэтому, ее можно исключить из цикла.

Результат

```
#include <stdafx.h>
#include <stdio.h>
int main()
{
int var1=1;
switch(var1)
{
case 0:
printf("null\n");
break;
case 1:
for (int var2=0; var2<=1; var2++)
{
printf("variant 1\n");
}
break;
case 2:
printf("variant 2\n");
break;
default:
printf("other variant\n");
}
scanf ("%i",&var1);
return var1-1;
}
```

2.5 Если просмотреть код, то можно обнаружить, что значение var1 всегда равно «1». Следовательно, можно исключить выражение «var1-1» в последней строке, заменив его «0».

Результат

```
#include <stdafx.h>
#include <stdio.h>
int main()
{
int var1=1;
switch(var1)
{
case 0:
printf("null\n");
break;
case 1:
for (int var2=0; var2<=1; var2++)
{
printf("variant 1\n");
}
break;
case 2:
printf("variant 2\n");
break;
default:
printf("other variant\n");
}
scanf ("%i",&var1);
return 0;
}
```

3. Скомпилировать программу, проверить работоспособность.

4. Разобраться в том, что делает данная программа. Добавить ваши комментарии к коду, поясняющие основные моменты.

Результат

```
#include <stdafx.h>
#include <stdio.h>
//главная функция программы
int main()
{
//инициализация переменной, от которой зависит выбор ветки switch
int var1=1;
//выбор варианта вывода на экран
switch(var1)
{
case 0:
//вывод фразы null и выход из оператора switch
printf("null\n");
break;
case 1:
//цикл двух повторений вывода на экран "variant 1"
for (int var2=0; var2<=1; var2++)
{
printf("variant 1\n");
}
break;
case 2:
//вывод на экран "variant 2"
printf("variant 2\n");
break;
default:
//вывод на экран "other variant"
printf("other variant\n");
}
//задержка вывода
scanf ("%i",&var1);
return var0;
}
```

2. Ответить на вопросы.

Вопросы:

1. Что такое деобфускация?
2. Зачем применяют деобфускацию?
3. С какими проблемами вы столкнулись при деобфускации кода?
4. Что невозможно восстановить при деобфускации кода?

Практические работы

Сарин Константин Сергеевич

Сеитбекова Лиали Даулетовна

БЕЗОПАСНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Текст дан в авторской редакции

Издательство «В-Спектр»

Подписано к печати 10.09.2018.

Формат 60×841/16. Печать трафаретная.

Печ. л. 6,9. Тираж 100 экз. Заказ 23.

Тираж отпечатан в издательстве «В-Спектр»

ИНН/КПП 7017129340/701701001, ОГРН 1057002637768

634055, г. Томск, пр. Академический, 13-24, тел. 49-09-91.

E-mail: bvm@sibmail.com