Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №7
Курс: Системное программирование
Тема: Обфускация

Выполнил студент группы 13541/3 _____ Д.В. Круминьш
(подпись)

Преподаватель _____ Е.В. Душутина
(подпись)

Санкт-Петербург
2017 г.

Содержание

Постановка задачи

В данной работе необходимо:

1. использовать для обфускации существующие(коммерческие средства);
2. самостоятельно обфусцировать свою программу (не травиальные методы).

Теория

2.1 Обфускация

Обфускация (”obfuscation запутывание), это один из методов защиты программного кода, который позволяет усложнить процесс реверсивной инженерии кода защищаемого программного продукта.[1]

Суть процесса обфускации заключается в том, чтобы запутать программный код и устранить большинство логических связей в нем, то есть трансформировать его так, чтобы он был очень труден для изучения и модификации посторонними лицами, но в тоже время сохранял свою функциональность.

Пусть **P1** - исходный код программы, а **P2** - код обфусцированной программы. Тогда **P2** должен удовлетворять следующим требованиям:

- код программы **P2** в результате трансформации будет существенно отличаться от кода программы **P1**, но при этом он будет выполнять те же функции что и код программы **P1**, а также будет работоспособным.
- изучение принципа работы, то есть процесс реверсивной инженерии, программы **P2** будет более сложным, трудоемким, и будет занимать больше времени, чем программы **P1**.
- при каждом процессе трансформации одного и того же кода программы **P1**, код программ **P2** будут различны.
- создание программы детрансформирующей программу **P2** в ее наиболее похожий первоначальный вид, будет неэффективно.

Программный код может быть представлен в двоичном виде (последовательность байтов представляющих собой так называемый машинный код, который получается после компиляции исходного кода программы) или исходном виде (текст содержащий последовательность инструкций какого-то языка программирования, который понятен человеку, этот текст в последствии будет подвержен компиляции или интерпретации на компьютере пользователя).

Процесс обфускации может быть осуществлен над любым из выше перечисленных видов представления программного кода, поэтому принято выделять следующие уровни процесса обфускации:

- низший уровень, когда процесс обфускации осуществляется над ассемблерным кодом программы, или даже непосредственно над двоичным файлом программы хранящим машинный код.
- высший уровень, когда процесс обфускации осуществляется над исходным кодом программы написанном на языке высокого уровня.

Осуществление обфускации на низшем уровне считается менее комплексным процессом, но при этом более трудно реализуемым по ряду причин. Одна из этих причин заключается в том, что должны быть учтены особенности работы большинства процессоров, так как способ обфускации, приемлемый на одной архитектуре, может оказаться неприемлемым на другой.

2.2 Виды обфускации

2.2.1 Лексическая обфускация

Обфускация такого вида 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, массивов, структур, хешей, функций, процедур и т.д.), на произвольные длинные наборы символов, которые трудно воспринимать человеку.
- добавление различных лишних (мусорных) операций.
- изменение расположения блоков (функций, процедур) программы, таким образом, чтобы это не коим образом не повлияло на ее работоспособность.

2.2.2 Обфускация данных

Такая обфускация связана с трансформацией структур данных. Она считается более сложной, и является наиболее продвинутой и часто используемой. Ее принято делить на три основные группы, которые описаны ниже.

2.2.3 Обфускация хранения

Заключается в трансформации хранилищ данных, а также самих типов данных (например, создание и использование необычных типов данных, изменение представления существующих и т.д.)

- изменение интерпретации данных определенного типа. Как известно сохранение, каких либо данных в хранилищах (переменных, массивах и т.д.) определенного типа (целое число, символ) в процессе работы программы, очень распространенное явление. Например, для перемещения по элементам массива очень часто используют переменную типа "целое число которая выступает в роли индекса. Использование в данном случае переменных иного типа возможно, но это будет не тривиально и может быть менее эффективно.
- изменение срока использования хранилищ данных, например переход от локального их использования к глобальному и наоборот. Преобразование статических (неменяющихся) данных в процедурные. Большинство программ, в процессе работы, выводят различную информацию, которая чаще всего в коде программы

представляется в виде статических данных таких как строки, которые позволяют визуально ориентироваться в ее коде и определять выполняемые операции. Такие строки также желательно предать обфускации, это можно сделать, просто записывая каждый символ строки, используя его ASCII код, например символ "A"можно записать как 16-ричное число "0x41 но такой метод банален. Наиболее эффективный метод, это когда в код программы в процессе осуществления обфусации добавляется функция, генерирующая требуемую строку в соответствии с переданными ей аргументами, после этого строки в этом коде удаляются, и на их место записывается вызов этой функции с соответствующими аргументами.

- разделение переменных. Переменные фиксированного диапазона могут быть разделены на две и более переменных. Для этого переменную "V"имеющую тип "x"разделяют на "k"переменных "v1,...,vk"типа "y"то есть "V == v1,...,vk". Потом создается набор функций позволяющих извлекать переменную типа "x"из переменных типа "y"и записывать переменную типа "x"в переменные типа "y". В качестве примера разделения переменных, можно рассмотреть способ представления одной переменной "B"логического типа (boolean) двумя переменными "b1, b2"типа короткого целого (short), значение которых будет интерпретироваться таким образом:

```
1       B    | b1 | b2
2     ------------------
3     false  | 0  | 0
4     true   | 0  | 1
5     true   | 1  | 0
6     false  | 1  | 1
```

2.2.4 Обфускация соединения

Один из важных этапов, в процессе реверсивной инженерии программ, основан на изучении структур данных. Поэтому важно постараться, в процессе обфускации, усложнить представление используемых программой структур данных. Например, при использовании обфускации соединения это достигается благодаря соединению независимых данных, или разделению зависимых. Ниже приведены основные методы, позволяющие осуществить такую обфускацию:

- объединение переменных. Две или более переменных "v1,...,vk"могут быть объединены в одну переменную "V если их общий размер ("v1,...,vk") не превышает размер переменной "V".

- реструктурирование массивов, заключается в запутывании структуры массивов, путем разделения одного массива на несколько подмассивов, объединения нескольких массивов в один, сворачивания массива (увеличивая его размерность) и наоборот, разворачивая (уменьшая его размерность).

- изменение иерархий наследования классов, осуществляется путем усложнения иерархии наследования при помощи создания дополнительных классов или использования ложного разделения классов.

2.2.5 Обфускация переупорядочивания

Заключается в изменении последовательности объявления переменных, внутреннего расположения хранилищ данных, а также переупорядочивании методов, масси-

вов (использование нетривиального представления многомерных массивов), определенных полей в структурах и т.д.

2.2.6 Обфускация управления

Обфускация такого вида осуществляет запутывание потока управления, то есть последовательности выполнения программного кода.

Большинство ее реализаций основывается на использовании непрозрачных предикат, в качестве, которых выступают, последовательности операций, результат работы которых сложно определить (само понятие ”предикат”выражает свойство одного объекта (аргумента), или отношения между несколькими объектами).

Сведения о системе

Работа производилась на реальной системы, с параметрами представленными ниже.

Элемент	Значение
Имя ОС	Майкрософт Windows 10 Pro (Registered Trademark)
Версия	10.0.16299 Сборка 16299
Дополнительное описание ОС	Недоступно
Изготовитель ОС	Microsoft Corporation
Имя системы	USER-PC
Изготовитель	HP
Модель	OMEN by HP Laptop 15-ce0xx
Тип	Компьютер на базе x64
SKU системы	1ZB00EA#ACB
Процессор	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 МГц, ядер: 4, логических процессоров: 4
Версия BIOS	American Megatrends Inc. F.04, 10.05.2017
Версия SMBIOS	3.0
Версия встроенного контроллера	40.20
Режим BIOS	Устаревший
Изготовитель основной платы	HP
Модель основной платы	Недоступно
Имя основной платы	Основная плата
Роль платформы	Мобильный
Состояние безопасной загрузки	Не поддерживается
Конфигурация PCR7	Привязка невозможна
Папка Windows	C:\Windows
Системная папка	C:\Windows\system32
Устройство загрузки	\Device\HarddiskVolume1
Язык системы	Россия
Аппаратно-зависимый уровень (HAL)	Версия = ”10.0.16299.98”
Имя пользователя	USER-PC\Tom

Часовой пояс	RTZ 2 (зима)
Установленная оперативная память (RAM)	8,00 ГБ
Полный объем физической памяти	7,87 ГБ
Доступно физической памяти	3,54 ГБ
Всего виртуальной памяти	12,6 ГБ
Доступно виртуальной памяти	6,82 ГБ
Размер файла подкачки	4,75 ГБ
Файл подкачки	C:\pagefile.sys

Таблица 3.1: Информация об используемой системе

Для разработки использовалась Microsoft Visual Studio Enterprise 2017 (Версия 15.3.0).

Существующие решения

4.1 StarForce C++ Obfuscator

Эффективное решение, предназначенное для обфускации (преобразования) исходных текстов программ, написанных на языках C и C++ с целью их защиты от реверс-инжиниринга. В результате обфускации код программы получает надежную защиту от анализа, выполняемого как человеком, так и машиной.[2]

В результате защиты обфусцированным получается как исходный текст, так и бинарный код, что значительно увеличивает уровень взломостойкости.

Обфускатор поддерживает более 30 методов обфускации, которые можно независимо включать, выключать и настраивать с помощью конфигурационного файла. Среди наиболее эффективных методов можно назвать:

- преобразование кода C++ в код виртуальной машины;
- шифрование строк и массивов;
- преобразование кода в цифровой автомат;
- введение ложных связей;
- объединение участков кода.

Провести эксперименты с данным решением не удалось, так как для получения хотябы бесплатной версии необходимо предварительно падать заявку на использование и дождаться её принятия.

4.2 Stunnix CXX Obfuscator

Stunnix CXX-Obfus – это продвинутый кросс-платформенный (Windows, MacOS X и любой Unix поддерживаем!) профессиональный кодировщик и шифратор – решение для защиты интеллектуальной собственности, которая делает невозможным прочитать и использовать C/C++ коды, с продвинутым GUI – менеджером проектов – поддержка проектов с файлами, которыми сложно управлять, с использованием директив и макросов, смешанным C и C++ языков, которые используют любой диалект C/C++.[3]

В общем случае это лишь лексический обфускатор, который изменяет имена классов и переменных.

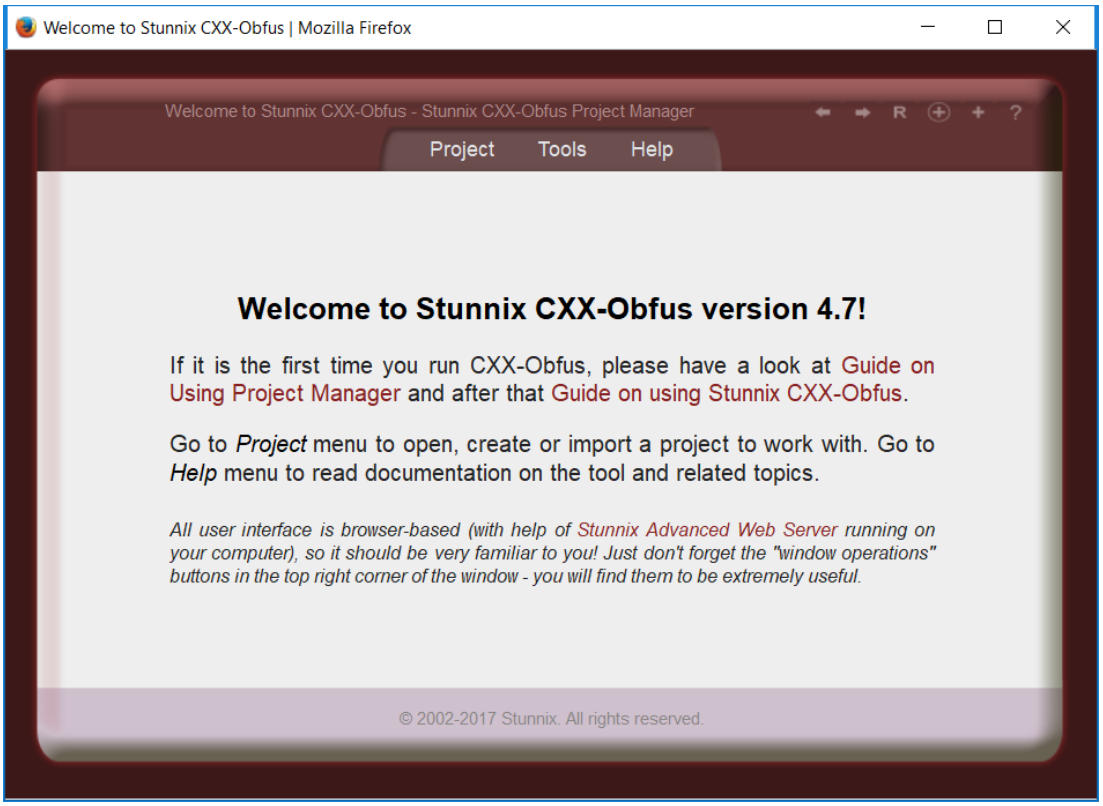

Рис. 4.1: Меню приветствия

Для обфускации был выбран проект с библиотекой перехвату функций.

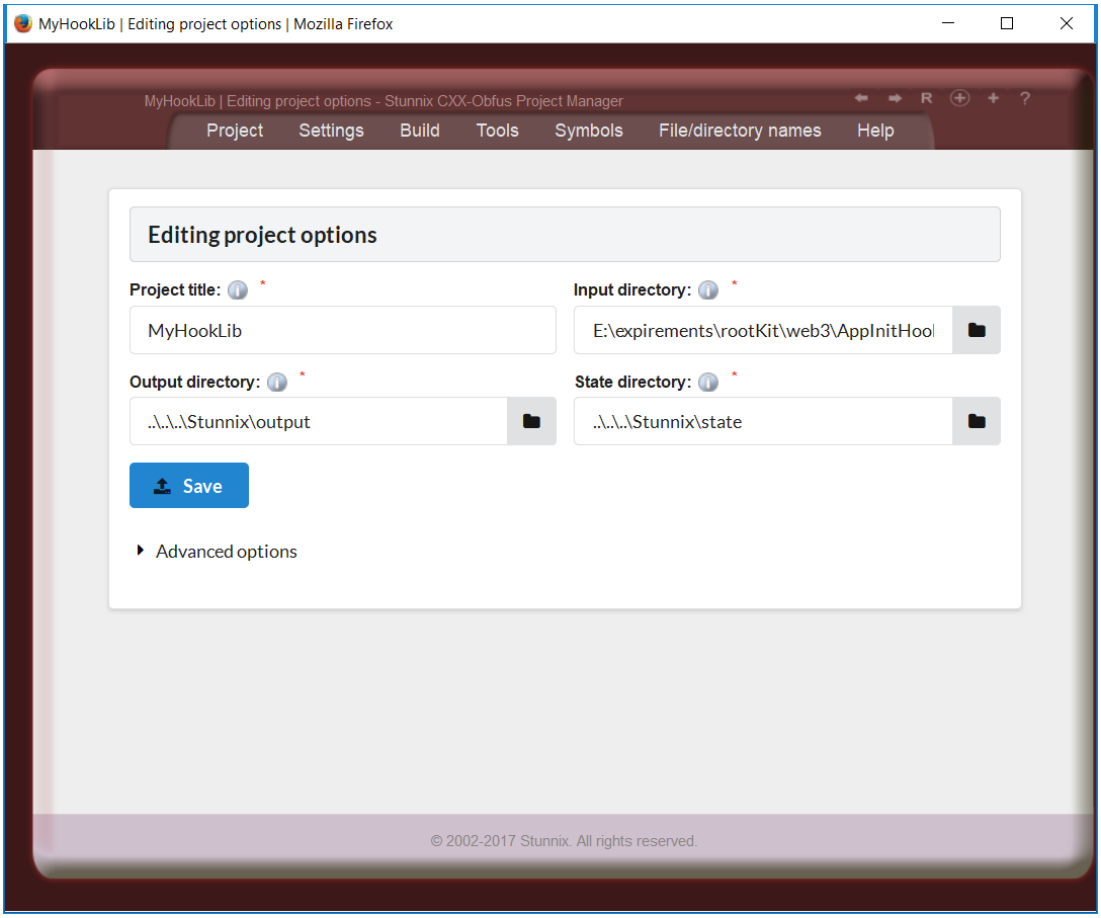

Рис. 4.2: Настройки проекта

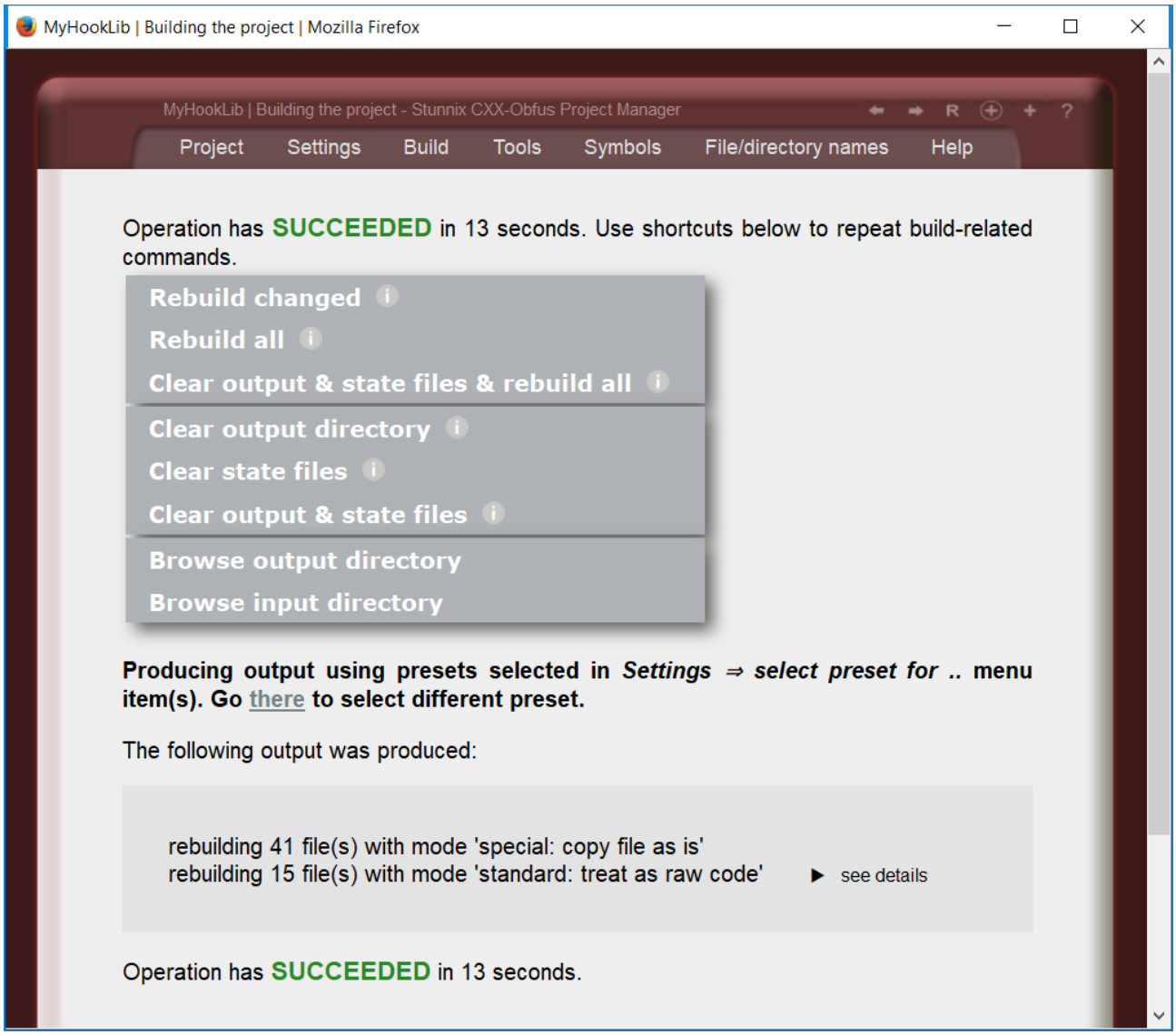

Рис. 4.3: Сообщение об успешной обфускации

```
1  ...
2  #include <psapi.h>
3  #include <strsafe.h>
4  using namespace std;
5
6  #define ReplacementFor_PATH "E:\\testFolder"
7  #define ReplacementFor_PATH_RECYCLE "E:\\"
8
9  HMODULE ReplacementFor_DllHandle;
10
11
12 time_t ReplacementFor_rawtime;
13 struct tm * ReplacementFor_timeinfo;
14 char ReplacementFor_buffer[80];
15
16
17 TCHAR name[UNLEN + 1];
18
19
20 #define ReplacementFor_BUFSIZE 512
21 TCHAR ReplacementFor_pszFilename[MAX_PATH + 1];
22
```

```
23
24
25 void ReplacementFor_SetupTime() {
26     time(&ReplacementFor_rawtime);
27     ReplacementFor_timeinfo = localtime(&ReplacementFor_rawtime);
28     strftime(ReplacementFor_buffer, sizeof(ReplacementFor_buffer), "%d-%m-%Y %H
       ↪ :%M:%S", ReplacementFor_timeinfo);
29 }
30
31 ...
32
33 bool WINAPI ReplacementFor_HookCreateDirectoryW(LPCTSTR
       ↪ ReplacementFor_lpPathName, LPSECURITY_ATTRIBUTES
       ↪ ReplacementFor_lpSecurityAttributes) {
34     ReplacementFor_SetupTime();
35     ReplacementFor_SetupUserName();
36
37     CStringA ReplacementFor_stringLpPathName(ReplacementFor_lpPathName);
38     const char* ReplacementFor_charLpPathName = ReplacementFor_stringLpPathName
       ↪ ;
39
40     if (strstr(ReplacementFor_charLpPathName, ReplacementFor_PATH)) {
41         wofstream ReplacementFor_myfile;
42         ReplacementFor_myfile.open("E:\\log.txt", ios::app);
43         ReplacementFor_myfile << "Time: " << ReplacementFor_buffer << " |
       ↪ CreateDirectoryW | User: " << name << " | Path: " <<
       ↪ ReplacementFor_charLpPathName << endl;
44         ReplacementFor_myfile.close();
45     }
46
47     return ReplacementFor_RealCreateDirectory(ReplacementFor_lpPathName,
       ↪ ReplacementFor_lpSecurityAttributes);
48 }
49
50 ...
```

Листинг 4.1: обфусцированный main.cpp

Обфускация была выполнена успешно, и для 15 файлов исходный код был изменен. В частности в листинге выше представлены фрагменты обфусцированного кода основного файла написанной ранее библиотеки перехватчика. В логе видно что к именам функций, переменных, классов было добавлено **ReplacementFor_**, что является довольно примитивным видом обфускации.

Если открыть настройки и изменить тип обфускации, то программа выведет сообщение что прочие способы лексической обфускации доступны лишь в платной версии.

В платной версии доступна замена имен на md5 hash, случайные символы и т.д.

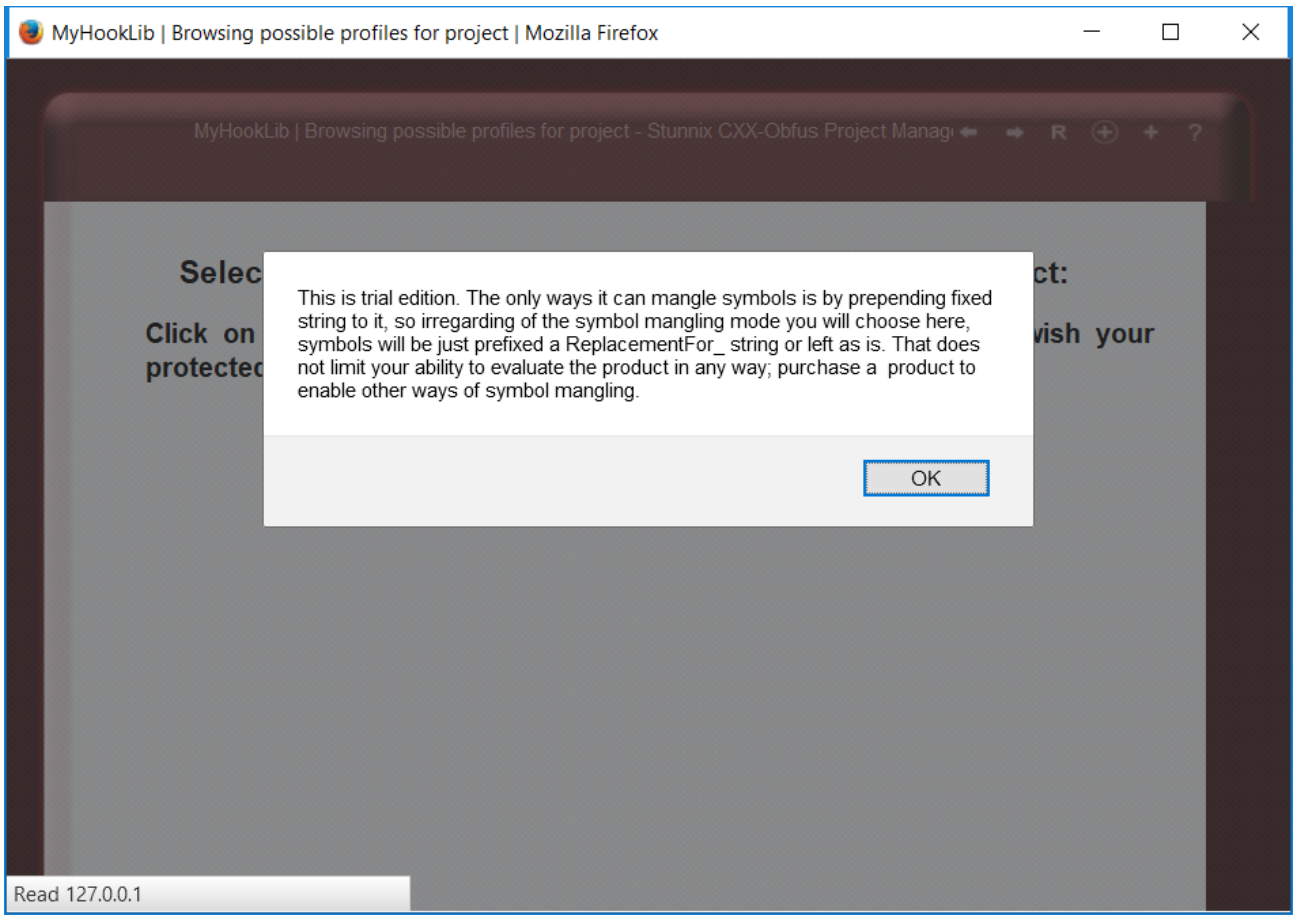

Рис. 4.4: Ограничение бесплатной версии

ADVobfuscator

Также существуют свободно распространяемые решения с открытым кодом. Далее будет произведена обфускация хранения данных. Исходный код проекта ADVobfuscator выложен на github[4].

5.1 Хранение данных

Создадим новый проект в visual studio, подключем к нему ADVobfuscator и напишем следующий исходный код:

```
1  #if !defined(DEBUG) || DEBUG == 0
2  #define BOOST_DISABLE_ASSERTS
3  #endif
4
5  #pragma warning(disable: 4503)
6
7  #define ADVLOG 1
8  #include <iostream>
9  #include "Log.h"
10 #include "MetaString.h"
11
12 using namespace std;
13 using namespace andrivet::ADVobfuscator;
14
15 void SampleEncryped()
16 {
17     cout << "Hello world" << endl;
18     cout << OBFUSCATED("Secret text 1") << endl;
19     cout << OBFUSCATED("Secret text 2") << endl;
20     cout << OBFUSCATED("Secret text 1") << endl;
21 }
22
23 // Entry point
24 int main(int, const char *[])
25 {
26     SampleEncryped();
27     return 0;
28 }
```

Листинг 5.1: ObfuscatedStrings.cpp

Код достаточно прост для понимания. В первой строчке метода происходит стандартный вывод в консоль некоторых символов. Далее используя функцию **OBFUSCATED** также идет вывод в консоль.

Запустим программу.

```
1  E:\expirements\obfuscation\test2\ADVobfuscator-master\x64\Debug>
       ↪ ObfuscatedStrings.exe
```

```
Hello world
—— Implementation #2 with key 0x4c
Secret text 1
—— Implementation #2 with key 0x5b
Secret text 2
—— Implementation #1 with key 0x6d
Secret text 1
```

Листинг 5.2: Запуск ObfuscatedStrings.cpp

Первая строка вывелась как и положена, в остальных строках при выводе было добавлено сообщение о том с каким ключем(об этом далее) был произведен вывод.

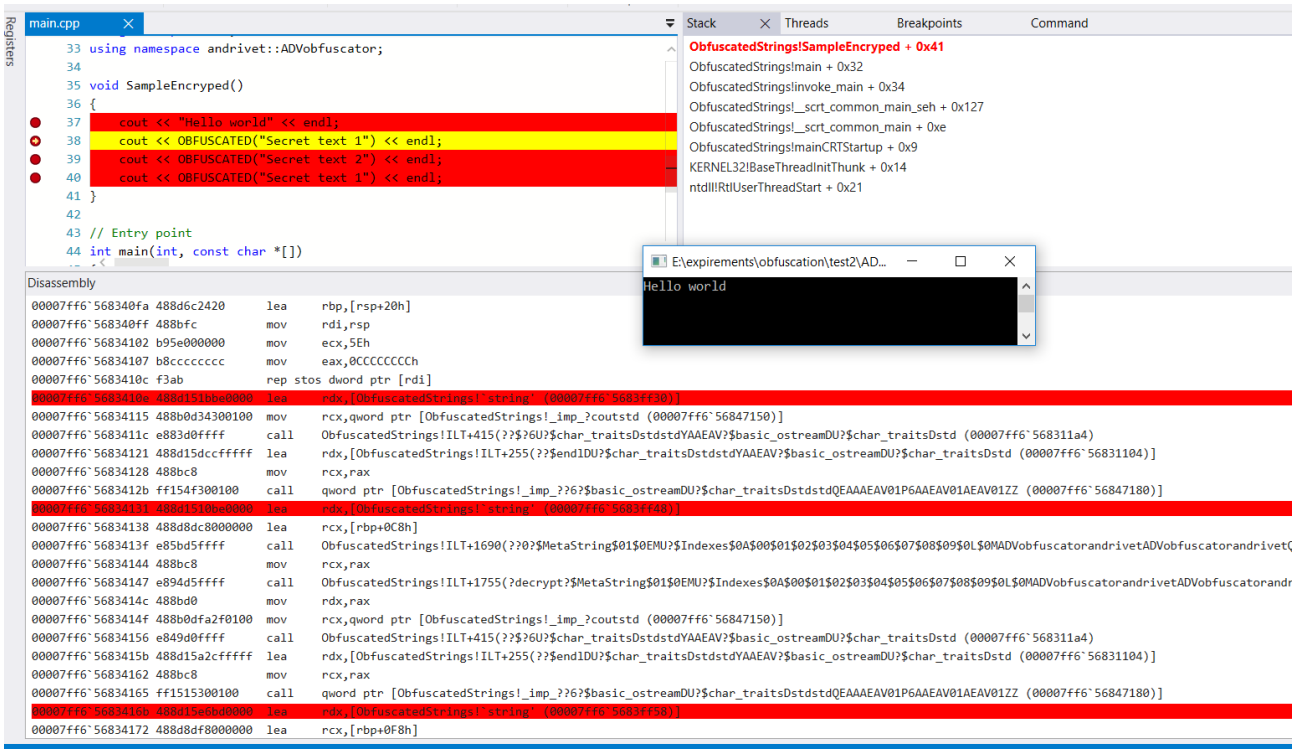

Рис. 5.1: Открытие в WinDbg

Если запустить программу в отладчике WinDbg, то по точкам останова можно заметить что перед выводом всех обфусцированных данных, происходят дополнительные действия. В частности вызываются функции по получению реального значения.

Для того чтобы разобраться что происходит, рассмотрим исходный код библиотеки, который был дополнен комментариями.

```
#ifndef MetaString_h
#define MetaString_h

#include "Inline.h"
#include "Indexes.h"
#include "MetaRandom.h"
#include "Log.h"

namespace andrivet { namespace ADVobfuscator {

  template<int N, char Key, typename Indexes>
  struct MetaString;

  // Структура состоит из индексов I, ключа K и алгоритма N = 0
```

```
15  // каждый символ зашифрован оператором XOR с определенным ключем (key)
16
17  template<char K, int... I>
18  struct MetaString<0, K, Indexes<I...>>
19  {
20    // конструктор
21    constexpr ALWAYS_INLINE MetaString(const char* str)
22    : key_{K}, buffer_ {encrypt(str[I], K)...} { }
23
24    // расшифровка
25    inline const char* decrypt()
26    {
27      for(size_t i = 0; i < sizeof...(I); ++i)
28        buffer_[i] = decrypt(buffer_[i]);
29      buffer_[sizeof...(I)] = 0;
30      LOG("--- Implementation #" << 0 << " with key 0x" << hex(key_));
31      return const_cast<const char*>(buffer_);
32    }
33
34  private:
35    // реализация некоторых функций
36    // для зашифровки используем ключ и символ
37    constexpr char key() const { return key_; }
38    constexpr char ALWAYS_INLINE encrypt(char c, int k) const { return c ^ k; }
39    // при расшифровке используем тот же метод которым шифровали
40    constexpr char decrypt(char c) const { return encrypt(c, key()); }
41
42    volatile int key_; // уникальный ключ для расшифровки
43    volatile char buffer_[sizeof...(I) + 1]; // буфер для хранения
      ↪ зашифрованного текста
44  };
45
46  // Структура состоит из индексов I, ключа K и алгоритма N = 1
47  // каждый символ зашифрован оператором XOR с определенным ключем (key)
48
49  template<char K, int... I>
50  struct MetaString<1, K, Indexes<I...>>
51  {
52    // конструктор
53    constexpr ALWAYS_INLINE MetaString(const char* str)
54    : key_(K), buffer_ {encrypt(str[I], I)...} { }
55
56    // расшифровка
57    inline const char* decrypt()
58    {
59      for(size_t i = 0; i < sizeof...(I); ++i)
60        buffer_[i] = decrypt(buffer_[i], i);
61      buffer_[sizeof...(I)] = 0;
62      LOG("--- Implementation #" << 1 << " with key 0x" << hex(key_));
63      return const_cast<const char*>(buffer_);
64    }
65
66  private:
67    // для каждого символа свой ключ, который зависит от его позиции
68    constexpr char key(size_t position) const { return static_cast<char>(key_ +
      ↪  position); }
69    constexpr char ALWAYS_INLINE encrypt(char c, size_t position) const {
      ↪ return c ^ key(position); }
70    constexpr char decrypt(char c, size_t position) const { return encrypt(c,
      ↪ position); }
```

```
71
72      volatile int key_; // уникальный ключ для расшифровки
73      volatile char buffer_[sizeof...(I) + 1]; // буфер для хранения
        ↪ зашифрованного текста
74    };
75
76    // Структура состоит из индексов I, ключа K и алгоритма N = 2
77    // каждый символ зашифрован оператором XOR с определенным ключем (key)
78
79    template<char K, int... I>
80    struct MetaString<2, K, Indexes<I...>>
81    {
82      // конструктор
83      constexpr ALWAYS_INLINE MetaString(const char* str)
84      : buffer_ {encrypt(str[I])..., 0} { }
85
86      // расшифровка
87      inline const char* decrypt()
88      {
89        for(size_t i = 0; i < sizeof...(I); ++i)
90          buffer_[i] = decrypt(buffer_[i]);
91        LOG("--- Implementation #" << 2 << " with key 0x" << hex(K));
92        return const_cast<const char*>(buffer_);
93      }
94
95    private:
96      // выполняется дополнительная работа с ключем
97      // получения остатка от деления на 13
98      // и прибавление 1
99      constexpr char key(char key) const { return 1 + (key % 13); }
100     constexpr char ALWAYS_INLINE encrypt(char c) const { return c + key(K); }
101     constexpr char decrypt(char c) const { return c - key(K); }
102
103     // буфер для хранения зашифрованного текста
104     volatile char buffer_[sizeof...(I) + 1];
105   };
106
107   // генератор случайного ключа
108   template<int N>
109   struct MetaRandomChar
110   {
111     // используем 0x7F как максимальное значение так как char является signed
112     static const char value = static_cast<char>(1 + MetaRandom<N, 0x7F - 1>::
        ↪ value);
113   };
114
115
116 }}
117
118 // функции деобфускации
119 #define DEF_OBFUSCATED(str) MetaString<andrivet::ADVobfuscator::MetaRandom<
        ↪ __COUNTER__, 3>::value, andrivet::ADVobfuscator::MetaRandomChar<
        ↪ __COUNTER__>::value, Make_Indexes<sizeof(str) - 1>::type>(str)
120
121 #define OBFUSCATED(str) (DEF_OBFUSCATED(str).decrypt())
122
123 #endif
```

Листинг 5.3: MetaString.h

В коде имеется структура **MetaString**, в которой имеются методы:

1. **encrypt** - для зашифровки;

2. **decrypt** - для расшифроки;

3. **key** - для получения ключа.

При вызове макроса **OBFUSCATED** вызывается макрос **DEF_OBFUSCATED** к которому применена функция **decrypt**.

При вызове макроса **DEF_OBFUSCATED** выполняется:

1. Определение алгоритма(0-2);

2. Генерация уникального ключа для зашифровки/расшифровки;

3. Преобразование входного набора символов в массив индексов.

Алгоритм определяется случайным образом, имеется три алгоритма, которые отличаются друг от друга методами шифрования/расшифрования.

Алгоритм 0
Для шифрования/расшифрования используется оператор **XOR**. Первый операнд код символа, второй операнд - ключ.

Алгоритм 1
Для шифрования/расшифрования используется оператор **XOR**. Первый операнд код символа, второй операнд - (ключ + позиция символа в массиве).

Алгоритм 2
Для шифрования/расшифрования используется сложение. Первый операнд код символа, второй операнд - (1 + (ключ % 13)).

Таким образом, при данном методе обфускации, исходное значение текстовой переменной было зашифровано, что осложняет процесс реверсивной инженерии.

5.2 Внедрение

Из эксперимента выше было выяснено, что текст можно зашифровать. Внедрим это в код библиотеки с функциями перехватчиками, в частности обфусцируем названия методов для перехвата из **kernel32.dll**.

```
1 ...
2
3 _CreateDirectory RealCreateDirectory = (_CreateDirectory)GetProcAddress(
    ↪ GetModuleHandle(L"Kernel32"), OBFUSCATED("CreateDirectoryW"));
4 _CreateDirectory RealCreateDirectoryA = (_CreateDirectory)GetProcAddress(
    ↪ GetModuleHandle(L"Kernel32"), OBFUSCATED("CreateDirectoryA"));
5 _CreateFile RealCreateFile = (_CreateFile)GetProcAddress(GetModuleHandle(L"
    ↪ Kernel32"), OBFUSCATED("CreateFileW"));
6 _MoveFile RealMoveFile = (_MoveFile)GetProcAddress(GetModuleHandle(L"Kernel32")
    ↪ , OBFUSCATED("MoveFileW"));
7 _RemoveDirectory RealRemoveDirectory = (_RemoveDirectory)GetProcAddress(
    ↪ GetModuleHandle(L"Kernel32"), OBFUSCATED("RemoveDirectoryW"));
8 _DeleteFile RealDeleteFileW = (_DeleteFile)GetProcAddress(GetModuleHandle(L"
    ↪ Kernel32"), OBFUSCATED("DeleteFileW"));
```

```
9  _DeleteFile RealDeleteFileA = (_DeleteFile)GetProcAddress(GetModuleHandle(L"
       ↪ Kernel32"), OBFUSCATED("DeleteFileA"));
10 _CopyFile RealCopyFile = (_CopyFile)GetProcAddress(GetModuleHandle(L"Kernel32")
       ↪ , OBFUSCATED("CopyFileW"));
11 _GetFileAttributes RealGetFileAttributes = (_GetFileAttributes)GetProcAddress(
       ↪ GetModuleHandle(L"Kernel32"), OBFUSCATED("GetFileAttributesW"));
12 _ReplaceFile RealReplaceFile = (_ReplaceFile)GetProcAddress(GetModuleHandle(L"
       ↪ Kernel32"), OBFUSCATED("ReplaceFileW"));
13 _LZOpenFile RealLZOpenFileW = (_LZOpenFile)GetProcAddress(GetModuleHandle(L"
       ↪ Kernel32"), OBFUSCATED("LZOpenFileW"));
14 _ReadFile RealReadFile = (_ReadFile)GetProcAddress(GetModuleHandle(L"Kernel32")
       ↪ , OBFUSCATED("ReadFile"));
15 _OpenFile RealOpenFile = (_OpenFile)GetProcAddress(GetModuleHandle(L"Kernel32")
       ↪ , OBFUSCATED("OpenFile"));
16 _DeleteFileTransacted RealDeleteFileTransactededW = (_DeleteFileTransacted)
       ↪ GetProcAddress(GetModuleHandle(L"Kernel32"), OBFUSCATED("
       ↪ DeleteFileTransactedW"));
17
18 ...
```

Листинг 5.4: Запуск ObfuscatedStrings.cpp

Соберем библиотеку. Конечный размер библиотеки вырос с 556 кБ до 586 кБ, это связано с добавлением библиотеки для обфускации. При работе с созданной библиотекой, функции также успешно перехватывались.

5.3 Запутывание логики вызовов

Для запутывания логики в данном решении используется **msm(Meta State Machine)** (некоторое подобие конечного автомата) из библиотеки **boost**. Суть заключается в дополнительных, ничего не значащих действиях перед вызовом защищаемой функции. По графу переходов мы видим что для запутывания программа переходим между состояними **State1, State2, State4**, затем при выполнении действия **Event3** происходит переход на состояние **State5** и затем на исходную функцию.

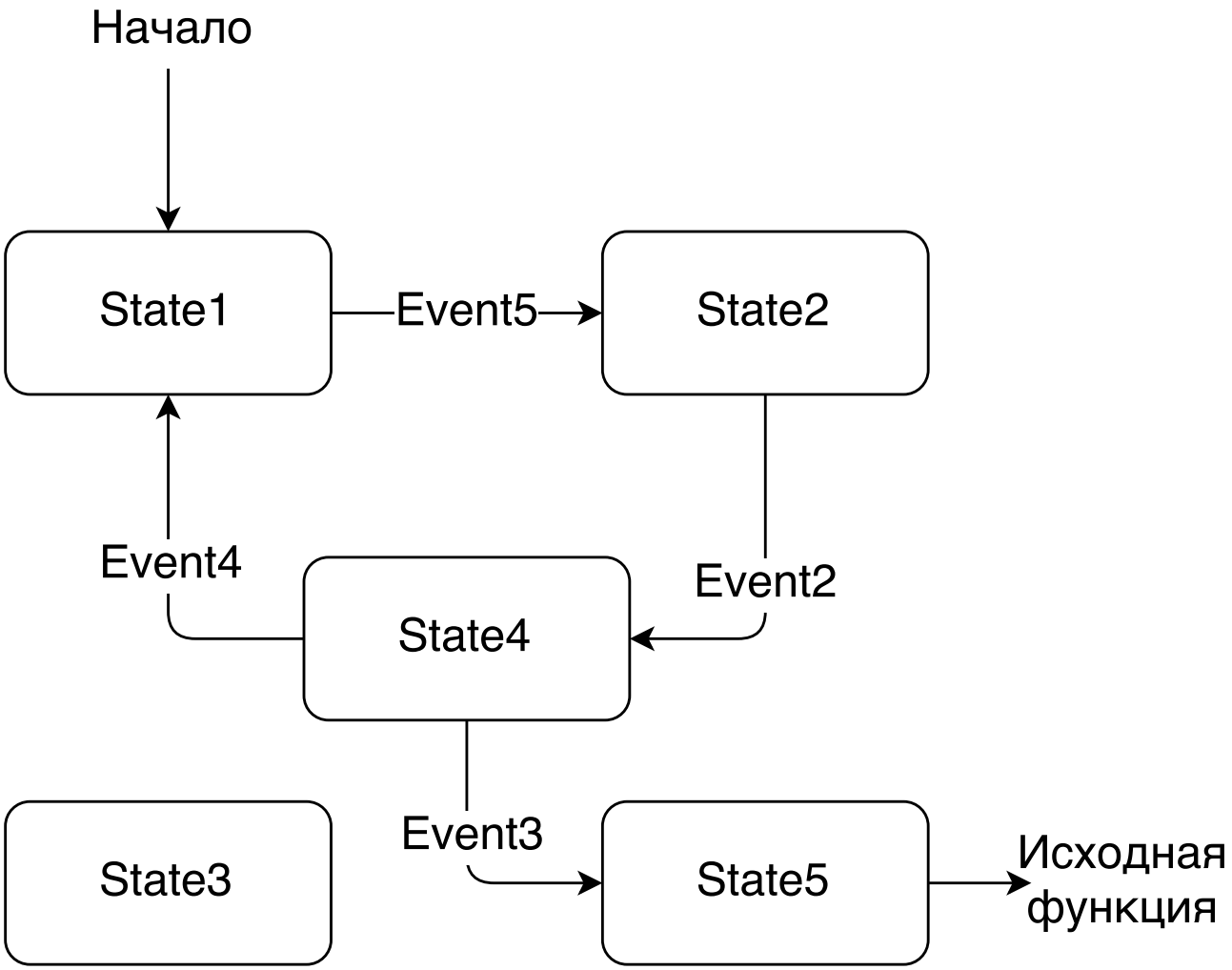

Рис. 5.2: Граф конечного автомата(созданный по коду)

Также автор пишет что исходную функцию лучше выполнить где-то в середине функций пустышек. Далее приведен исходный код библиотеки, отвечающий за данные действия, с добавлением комментариев.

```
1  #ifndef ObfuscatedCall_h
2  #define ObfuscatedCall_h
3
4  #include "MetaFSM.h"
5  #include "MetaRandom.h"
6
7  // Обфускация вызова функции используя (finite state machine (FSM))конечный(
       ↪ автомат).
8  // В данном примере, конечная вызываемая функция находится после действий FSM, так
       ↪ что она может быть найдена
9  // лучше прятать функцию гдето– в серединев( действиях FSM)
10
11 namespace andrivet { namespace ADVobfuscator { namespace Machine1 {
12
13     // Finite State Machine
14     // E: функция для обфускации
15     // R: тип возращаемого значения
16     template<typename E, typename R = Void>
17     struct Machine : public msm::front::state_machine_def<Machine<E, R>>
18     {
19         // ––– события для перехода по автомату
20         struct event1 {};
```

```
21          struct event2 {};
22          struct event3 {};
23          struct event4 {};
24          struct event5 {};
25
26          // ——— состояния автомата
27          struct State1 : public msm::front::state<>{};
28          struct State2 : public msm::front::state<>{};
29          struct State3 : public msm::front::state<>{};
30          struct State4 : public msm::front::state<>{};
31          struct State5 : public msm::front::state<>{};
32          struct Final  : public msm::front::state<>{};
33
34          // ——— переходы
35          struct CallTarget
36          {
37              template<typename EVT, typename FSM, typename SRC, typename TGT>
38              void operator()(EVT const& evt, FSM& fsm, SRC&, TGT&)
39              {
40                  fsm.result_ = evt.call();
41              }
42          };
43
44          // ——— инициализация начального состояния конечного автомата
45          using initial_state = State1;
46
47          // ——— таблица переходов
48          struct transition_table : mpl::vector<
49          //    Start     Event         Next      Action       Guard
50          //  +---------+-------------+---------+------------+---------------+
51          Row < State1  , event5      , State2  >,
52          Row < State1  , event1      , State3  >,
53          //  +---------+-------------+---------+------------+---------------+
54          Row < State2  , event2      , State4  >,
55          //  +---------+-------------+---------+------------+---------------+
56          Row < State3  , none        , State3  >,
57          //  +---------+-------------+---------+------------+---------------+
58          Row < State4  , event4      , State1  >,
59          Row < State4  , event3      , State5  >,
60          //  +---------+-------------+---------+------------+---------------+
61          Row < State5  , E           , Final   ,   CallTarget   >
62          //  +---------+-------------+---------+------------+---------------+
63          > {};
64
65          using StateMachine = msm::back::state_machine<Machine<E, R>>;
66
67          template<typename F, typename... Args>
68          struct Run
69          {
70              static inline void run(StateMachine& machine, F f, Args&&... args)
71              {
72                  machine.start();
73
74                  // генерация случайных действий минимум( 55, максимум 98)
75                  Unroller<55 + MetaRandom<__COUNTER__, 44>::value>{}([&]()
76                  {
77                      machine.process_event(event5{});
78                      machine.process_event(event2{});
79                      machine.process_event(event4{});
80                  });
```

```
81
82                  machine.process_event(event5{});
83                  machine.process_event(event2{});
84                  machine.process_event(event3{});
85                  // вызов оригинальной функции
86                  machine.process_event(E{f, args...});
87              }
88          };
89
90          // возращаем результат
91          R result_;
92      };
93
94  }}}
95
96
97  #pragma warning(push)
98  #pragma warning(disable : 4068)
99  #pragma clang diagnostic push
100 #pragma clang diagnostic ignored "-Wgnu-zero-variadic-macro-arguments"
101
102 #define OBFUSCATED_CALL0(f) andrivet::ADVobfuscator::ObfuscatedCall<andrivet::
        ↪ ADVobfuscator::Machine1::Machine>(MakeObfuscatedAddress(f, andrivet::
        ↪ ADVobfuscator::MetaRandom<__COUNTER__, 400>::value + 278))
103 #define OBFUSCATED_CALL_RET0(R, f) andrivet::ADVobfuscator::ObfuscatedCallRet<
        ↪ andrivet::ADVobfuscator::Machine1::Machine, R>(MakeObfuscatedAddress(f,
        ↪ andrivet::ADVobfuscator::MetaRandom<__COUNTER__, 400>::value + 278))
104
105 #define OBFUSCATED_CALL(f, ...) andrivet::ADVobfuscator::ObfuscatedCall<
        ↪ andrivet::ADVobfuscator::Machine1::Machine>(MakeObfuscatedAddress(f,
        ↪ andrivet::ADVobfuscator::MetaRandom<__COUNTER__, 400>::value + 278),
        ↪ __VA_ARGS__)
106 #define OBFUSCATED_CALL_RET(R, f, ...) andrivet::ADVobfuscator::
        ↪ ObfuscatedCallRet<andrivet::ADVobfuscator::Machine1::Machine, R>(
        ↪ MakeObfuscatedAddress(f, andrivet::ADVobfuscator::MetaRandom<__COUNTER__,
        ↪  400>::value + 278), __VA_ARGS__)
107
108 #pragma clang diagnostic pop
109 #pragma warning(pop)
110
111
112 #endif
```

Листинг 5.5: ObfuscatedCall.h

Само запутывание происходит с использованием функции **OBFUSCATED_CALL_RET**. Которая имеет следующие параметры:

1. Тип функции(bool, void, int ...);
2. Указатель на функцию;
3. 1 и более передаваемых в функцию параметров.

В общем случае сама логика функции следующая:

- Инициализация событий переходов;
- Инициализация состояний автоматов;
- Инициализация переходов;

- Инициализация начального состояния msm(конечного автомата);
- Установка таблица переходов и состояний;
- Выполнение случайного количества переходов по автомату;
- Вызов оригинальной функции;
- Возвращение результата.

Теперь добавим использование данного типа обфускации.

```
...

case DLL_PROCESS_ATTACH:

    DllHandle = hInstance;
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealCreateDirectory,
    ↪ HookCreateDirectoryW);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealCreateDirectoryA,
    ↪ HookCreateDirectoryA);
    //Mhook_SetHook((PVOID*)&RealCreateFile, HookCreateFileW); // система не
    ↪ запустится
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealMoveFile,
    ↪ HookMoveFile);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealRemoveDirectory,
    ↪ HookRemoveDirectoryW);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealDeleteFileW,
    ↪ HookDeleteFileW);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealDeleteFileA,
    ↪ HookDeleteFileA);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealCopyFile,
    ↪ HookCopyFileW);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealGetFileAttributes,
    ↪ HookGetFileAttributes);// слишком большой лог
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealReplaceFile,
    ↪ HookReplaceFile);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealLZOpenFileW,
    ↪ HookLZOpenFileW);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealReadFile,
    ↪ HookReadFile);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&RealOpenFile,
    ↪ HookOpenFile);
    OBFUSCATED_CALL_RET(BOOL, Mhook_SetHook, (PVOID*)&
    ↪ RealDeleteFileTransactededW, HookDeleteFileTransactededW);
    break;

...
```

Листинг 5.6: Запуск ObfuscatedStrings.cpp

После данных действий, файл библиотеки вырос с 586 Кб до 779 Кб. При работе с созданной библиотекой, функции также успешно перехватывались.

Вывод

В данной работе я наглядно познакомился с возможными методами обфускации.

Сперва были рассмотрены коммерческие решения. Даже получить их пробную версию достаточно трудно. Рассмотренное решение от **Stunnix** - **CXX Obfuscator** показало лишь лексическую обфускацию и ничего более.

Больший интерес представляет решение от энтузиастов - **ADVobfuscator**, с помощью которого удалось обфусцировать:

1. хранение текстовых переменных(для их расшифровки необходим специальный ключ);

2. вызов функции, путем перехода по конечному автомату и его функциях.

Подобная обфускация сильно осложнит попытки понимания работы программы по её коду.

Литература

[1] Обфускация и защита программных продуктов. [Электронный ресурс]. — URL: http://citforum.ru/security/articles/obfus/#5_3 (дата обращения: 2017-12-17).

[2] StarForce C++ Obfuscator [Электронный ресурс]. — URL: http://www.star-force.ru/products/starforce-obfuscator/ (дата обращения: 2017-12-17).

[3] Stunnix C and C++ Obfuscator [Электронный ресурс]. — URL: http://www.softsoft.ru/development/c-c-c/35124.htm (дата обращения: 2017-12-17).

[4] ADVobfuscator [Электронный ресурс]. — URL: https://github.com/andrivet/ADVobfuscator (дата обращения: 2017-12-17).

[5] Boost.MetaStateMachine [Электронный ресурс]. — URL: https://theboostcpplibraries.com/boost.msm (дата обращения: 2017-12-18).

[6] Sub Machine State [Электронный ресурс]. — URL: http://redboltz.wikidot.com/sub-machine-state (дата обращения: 2017-12-18).