Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе**

**Дисциплина**: Проектирование ОС и компонентов

**Тема**: Обфускация кода

Выполнил студент гр. 13541/4 Степанов Д.С.

(подпись)

Руководитель Душутина Е.В.

(подпись)

“ ” 2017 г.

Санкт - Петербург

2017

**Цель работы**

Изучить методы, цели и средства обфускации кода. Попробовать и изучить существующее средство обфускации кода. Написать простейший обфускатор.

**Выполнение работы**

Обфускация или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции. Цели обфускации:

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

Обфускация кода может проводиться на нескольких уровнях:

* На уровне исходного кода (не меняют логику исходного кода)
* На уровне машинного кода
* На уровне промежуточного (байт) кода

**Изучение существующего обфускатора**

Для изучения был выбран обфускатор, разрабатываемый компанией stunnix. Данный обфускатором является межплатформенным проприетарным программным обеспечением, но имеет триальную бесплатную версию. Данный обфускатор предлагает множество шаблонов изменения кода, которые можно настраивать. Выбранный шаблон представлен на рисунке 1.

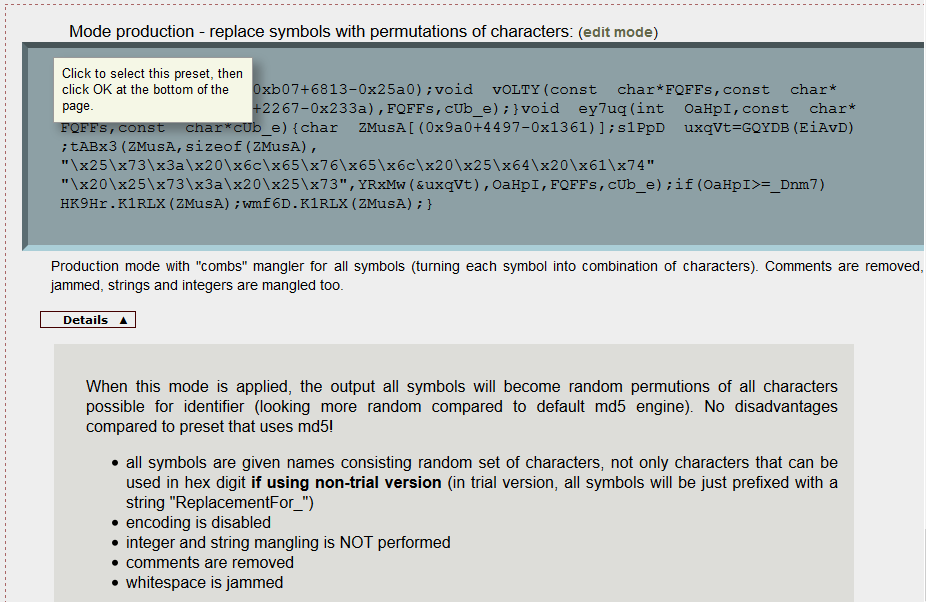


Рис.1. Возможности одного из режима обфускатора

Для испытания обфускатора была написана простейшая программа, подсчитывающая длину гипотенузы прямоугольного треугольника по двум катетам. Исходный код и его обфусцированная версия представлены ниже:

#include <iostream>

#include <cmath>

using namespace std;

// calc gipot

double getGip(int a, int b){

return std::sqrt(a\*a + b\*b);

}

int main(){

const int a = 3; //katet 1

const int b = 4; //katet 2

cout << "kat1=" << a << " kat2=" << b << endl; //output

int res = getGip(a, b);

cout<<"gipot="<<res<<endl;

}

#include <iostream>

#include <cmath>

using namespace std;double ReplacementFor\_getGip(int ReplacementFor\_a,int

ReplacementFor\_b){return std::sqrt(ReplacementFor\_a\*ReplacementFor\_a+

ReplacementFor\_b\*ReplacementFor\_b);}int main(){const int ReplacementFor\_a=

(0x1db5+356-0x1f16);const int ReplacementFor\_b=(0x3e5+8650-0x25ab);cout<<

"\x6b\x61\x74\x31\x3d"<<ReplacementFor\_a<<"\x20\x6b\x61\x74\x32\x3d"<<

ReplacementFor\_b<<endl;int ReplacementFor\_res=ReplacementFor\_getGip(

ReplacementFor\_a,ReplacementFor\_b);cout<<"\x67\x69\x70\x6f\x74\x3d"<<

ReplacementFor\_res<<endl;}

Обфусцированная версия компилируется и запускается:

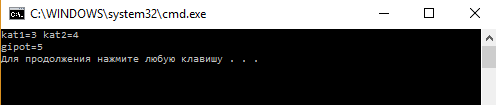


Рис. 2. Запуск обфусцированной версии кода.

**Собственный обфускатор**

Напишем собственный простейший обфускатор, который будет убирать все пробелы и переносы строк, удалять все комментарии и заменять строковые константы на их эквиваленты в 16-ном формате.

Исходный код приведен ниже:

#include <iostream>

#include <fstream>

#include <algorithm>

#include <vector>

#include <sstream>

void deleteComments(std::string& str) {

size\_t commentStart = 0;

size\_t commentEnd = 0;

while (true) {

commentStart = str.find("/\*", commentStart);

if (commentStart == std::string::npos) break;

commentEnd = str.find("\*/", commentStart + 1);

str.erase(str.begin() + commentStart, str.begin() + commentEnd + 2);

}

commentStart = 0;

while (true) {

commentStart = str.find("//", commentStart);

if (commentStart == std::string::npos) break;

commentEnd = str.find("\n", commentStart + 1);

str.erase(str.begin() + commentStart, str.begin() + commentEnd);

}

}

void replaceStrings(std::string& str) {

size\_t start = 0;

size\_t end\_ = 0;

while (true) {

start = str.find("\"", start);

if (start == std::string::npos) break;

end\_ = str.find("\"", start + 1);

std::ostringstream ss;

for (auto i = start + 1; i < end\_; ++i) {

if (str[i] == '\\' && str[i + 1] == 'n') {

ss << "\\x" << std::hex << (int)'\n';

++i;

}

else

ss << "\\x" << std::hex << (int)str[i];

}

str.erase(str.begin() + start + 1, str.begin() + end\_);

str.insert(start + 1, ss.str());

start = start + 2 + ss.str().size();

}

}

void removeFreeLinesAndSpaces(std::string& str) {

size\_t start = 0;

while (true) {

start = str.find("\n\n", start);

if (start == std::string::npos) break;

str.erase(str.begin() + start, str.begin() + start + 1);

}

start = 0;

while (true) {

start = str.find(" ", start);

if (start == std::string::npos) break;

str.erase(str.begin() + start, str.begin() + start + 1);

}

}

int main(int argc, char\*\* argv) {

std::ifstream in("example.cpp");

std::string str((std::istreambuf\_iterator<char>(in)),

std::istreambuf\_iterator<char>());

std::cout << str << std::endl << std::endl;

std::string res;

std::string cur;

std::vector<size\_t> indexes;

indexes.push\_back(str.rfind("#include"));

indexes.erase(std::remove(indexes.begin(), indexes.end(), std::string::npos), indexes.end());

auto max = std::max\_element(indexes.begin(), indexes.end());

if (max == indexes.end()) {

cur = str;

}

else {

auto endOfProproc = str.find('\n', \*max);

res.append(str.substr(0, endOfProproc + 1));

cur = str.substr(endOfProproc + 1, str.size());

}

//Рассматриваем всё только после include

deleteComments(cur);

cur.erase(std::remove(cur.begin(), cur.end(), '\n'), cur.end()); //удаляем переносы

cur.erase(std::remove(cur.begin(), cur.end(), '\t'), cur.end()); //удаляем табуляции

res.append(cur);

removeFreeLinesAndSpaces(res);

replaceStrings(res);

std::cout << res << std::endl;

return 0;

}

Демонстрация работы на том же примере:

#include <iostream> #include <cmath> using namespace std; // calc gipot double getGip(int a, int b){ return std::sqrt(a\*a + b\*b); } int main(){ const int a = 3; //katet 1 const int b = 4; //katet 2 cout << "kat1=" << a << " kat2=" << b << endl; //output int res = getGip(a, b); cout<<"gipot="<<res<<endl; }

#include <iostream> #include <cmath> using namespace std;double getGip(int a, int b){return std::sqrt(a\*a + b\*b);}int main(){const int a = 3; const int b = 4; cout << "\x6b\x61\x74\x31\x3d" << a << "\x20\x6b\x61\x74\x32\x3d" << b << endl; int res = getGip(a, b);cout<<"\x67\x69\x70\x6f\x74\x3d"<<res<<endl;}

Результаты не изменились:

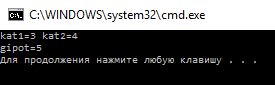


Рис. 3. Результаты запуска обфусцированного кода.

Теперь напишем деобфускатор, чтобы вернуть коду первоначальный вид. Для этого необходимо восстановить строковые константы к исходному виду и расставить пробелы и знаки переноса строки, где это возможно.

Исходный код представлен ниже:

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

void insertSpacesAndLines(std::string& str) {

size\_t start = 0;

while (true) {

start = str.find\_first\_of(";", start);

if (start == std::string::npos) break;

str.insert(start + 1, "\n");

start += 2;

}

start = 0;

while (true) {

start = str.find\_first\_of("{}", start);

if (start == std::string::npos) break;

str.insert(start + 1, "\n");

start += 2;

}

start = 0;

size\_t end = 0;

while (true) {

start = str.find("\n ", start);

if (start == std::string::npos) break;

end = str.find\_first\_not\_of(" ", start + 1);

if (end == std::string::npos) break;

str.erase(str.begin() + start + 1, str.begin() + end);

}

}

void normalizeStrings(std::string& str) {

size\_t start = 0;

size\_t end\_ = 0;

while (true) {

start = str.find("\"", start);

if (start == std::string::npos) break;

for (auto i = start + 1; i; ++i) {

if (str[i] == '\"') {

end\_ = i;

break;

}

}

auto conststr = str.substr(start + 1, end\_ - start - 1);

auto len = conststr.length();

std::string newString;

for (auto i = 0U; i < len; i += 4) {

std::string byte = conststr.substr(i + 2, 2);

char chr = (char)(int)strtol(byte.c\_str(), NULL, 16);

if (chr == '\n') {

newString += "\\n";

}

else {

newString.push\_back(chr);

}

}

str.erase(str.begin() + start + 1, str.begin() + end\_);

str.insert(start + 1, newString);

start = start + newString.size() + 2;

}

}

int main(int argc, char\*\* argv) {

std::ifstream in("example.cpp");

std::string str((std::istreambuf\_iterator<char>(in)),

std::istreambuf\_iterator<char>());

std::cout << str << std::endl << std::endl;

insertSpacesAndLines(str);

normalizeStrings(str);

std::ofstream out;

out.open("deobfus.cpp");

out << str;

std::cout << str << std::endl;

return 0;

}

Выходной файл:

#include <iostream>

#include <cmath>

using namespace std;

double getGip(int a, int b){

return std::sqrt(a\*a + b\*b);

}

int main(){

const int a = 3;

const int b = 4;

cout << "kat1=" << a << " kat2=" << b << endl;

int res = getGip(a, b);

cout<<"gipot="<<res<<endl;

}

Проверка работы деобфусцированного кода прошла успешно.

**Вывод**

В ходе лабораторной работы были изучены методы обфускации и часть методов было опробовано на практике. В ходе лабораторной работы был создан свой обфускатор для языка C++, а также деобфускатор.

Обфускация позволяет изменять исходный код до вида, который сложно анализировать (например, меняя названия переменных, вводя новые переменные, удаляя форматирование и комментарии). Однако, на практике было выяснено, что некоторые вещи (например, форматирование и внешний констант) возможно привести к изначальному виду. Для того, чтобы исходный текст было сложно восстановить необходимо использовать более сложные методы обфускации.

**Список использованных источников**

1. Интернет-источник. Официальный сайт обфускатора “Stunnix” <http://stunnix.com/prod/cxxo/> Дата обращения 30.04.2017
2. Интернет-источник. Википедия. Определение обфускации. <https://ru.wikipedia.org/wiki/Обфускация_(программное_обеспечение)> Дата обращения 30.04.2017
3. Интернет-источник . Статья «Обфускация кода» <https://habrahabr.ru/post/255871/> Дата обращения 30.04.2017.