1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**Изучение методов защиты программного обеспечения**

по дисциплине «Модели безопасности компьютерных систем»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ст. преподаватель Овасапян Т.Д.
4. Санкт-Петербург
5. 2021

# цель работы

Изучение методов защиты программного обеспечения и возможных способов их преодоления.

# ход работы

В качестве исходной программы была взята утилита, оценивающая состояние компонентов безопасности Windows. Атрибуты безопасности извлекаются с помощью функции WscGetSecurityProviderHealth, расположенной в библиотеке wscapi.h. На Рисунке 1 представлен результат работы программы.

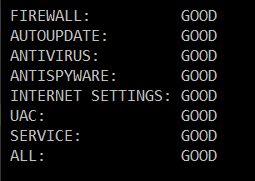


Рисунок 1 ― Результат работы программы

Добавим в программу примитивную проверку пароля, которая заключается в сравнении вводимой строки с некоторым константным значением. Если пароль корректен, то выполняется функционал программы, иначе появляется сообщение об ошибке. После пяти неудачных попыток ввода пароля программа завершится. На Рисунке 2 представлены попытки ввода верного и неверного пароля.

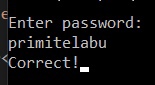
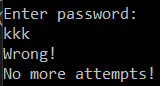


Рисунок 2 ― Проверка ввода пароля.

## Бинарный патчинг

Такую защиту достаточно легко обойти путём бинарного патчинга. С помощью утилиты IDA откроем исполняемый файл и найдём место, где осуществляется проверка пароля. Интерфейс IDA представляет программу в виде логических блоков, поэтому нужная функция находится достаточно быстро. На Рисунке 3 представлен интересующий взломщика участок кода.

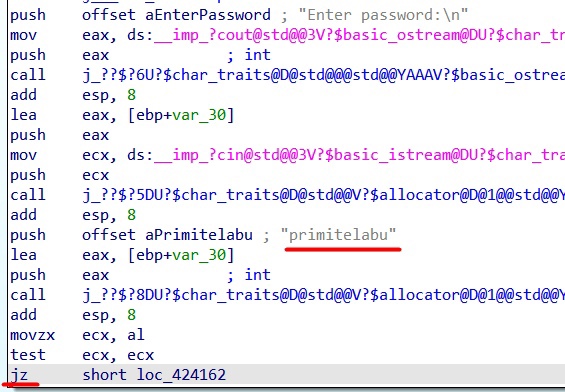


Рисунок 3 ― Функция проверки пароля

При открытии этого же участка кода в HEX-представлении можно понять, что инструкция jz, которая как раз отвечает за условный переход, имеет опкод 74 17. С помощью любого hex-редактора изменим этот код на 75 17. После этого любой неверный пароль будет восприниматься как верный, а верный тем временем будет вызывать ошибку (Рисунки 4 и 5).

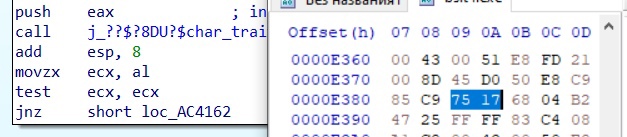


Рисунок 4 ― Модификация опкода

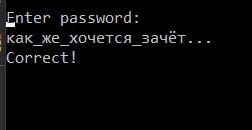


Рисунок 5 ― Демонстрация корректности неверного пароля

## Хранение строк в зашифрованном виде

На Рисунке 3 также видно, что строки с приглашением ввести пароль, а также сам пароль хранятся в открытом виде. Для усложнения задачи взлома введём операцию xor(4) для каждой строки. Теперь строки хранятся в зашифрованном виде, а перед печатью в консоль они снова приводятся к нормальному виду той же операцией.

for (unsigned int i = 0; i < 2; i++)

{

system("cls");

std::cout << XOR("Ajpav$tewwskv`>") << std::endl; //Enter password:

std::cin >> password;

if (password == XOR("tvmimpahefq")) { //primitelabu

std::cout << XOR("Gkvvagp%"); //Correct!

return TRUE;

}

else

std::cout << XOR("Svkjc%") << std::endl; //Wrong!

}

std::cout << XOR("Jk$ikva$eppaitpw%") << std::endl; //No more attempts!

return FALSE;

В таком случае злоумышленник не сможет определить, что это именно функция проверки пароля. Поиск по опкоду операции jz выдает 5 результатов, при изменении не той операции программа завершится с ошибкой. Конечно, в рамках данного примера перебрать 5 вариантов не составит труда, но для более серьезного ПО таких условных переходов может быть намного больше, что усложнит взлом. К тому же, следующие методы защиты также затруднят анализ кода.

## Проверка целостности

Следующим шагом была реализована проверка целостности кода. Вычисляется размер функции проверки пароля непосредственно в процессе выполнения кода и сравнивается с эталонным значением, посчитанным заранее. Данный метод защищает функцию от изменения.

BOOL check\_crc(bool(\*ptr\_check\_correct\_password)())

{

ULONG64 addr;

size\_t size = ((PUCHAR)help - (PUCHAR)check);

HMODULE hDLL = GetModuleHandle(TEXT("ntdll.dll"));

UndocFoo ComputeCrc32 = (UndocFoo)GetProcAddress(hDLL, "RtlComputeCrc32");

BYTE\* buffer = (BYTE\*)malloc(size \* sizeof(BYTE));

memcpy(buffer, ptr\_check\_correct\_password, size \* sizeof(BYTE));

INT iCRC32 = ComputeCrc32(INT(0), (BYTE\*)buffer, size);

//std::cout << (iCRC32 ) << std::endl; //589200476

if ((iCRC32 ^ 0xF4E7AA20) == 4119351741)

return TRUE;

return FALSE;

}

BOOL(\*p\_check\_crc)(bool(\*ptr\_check\_correct\_password)()) = &check\_crc;

Также был изменён класс, принадлежащий полезному функционалу: туда была добавлена дополнительная строка с паролем (зашифрованная через xor). В методах класса были добавлены дополнительные проверки на соответствие введенного пароля хранимому в классе. Это сделано на тот случай, если злоумышленник всё-таки сможет обойти основную функцию подмены пароля. Как видно из Рисунка 6, программа просто завершится с кодом -1 и функционал будет недоступен.

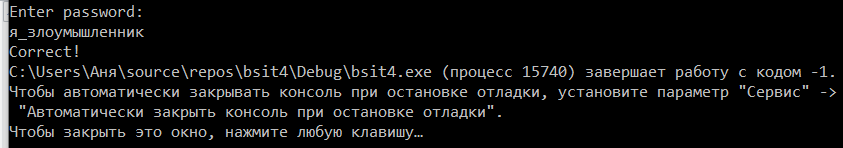


Рисунок 6 ― Реализация дополнительных проверок

## Обфускация кода

Также для усложнения анализа кода были добавлены «мусорные» функции и условия, как в main, так и внутри полезного функционала.

void trash() {

char c;

int a;

for (unsigned int i = 0; i < 1000; i++) {

a = i + 8465212;

c = a % 100;

a = 5521822;

i \*= 27;

}

}

int garbage() {

std::string s = "plmw$mw$cevfeca$bqjg"; //this is garbage func

char c = '!';

s += c;

return 0;

}

## Антиотладочные средства

### IsDebuggerPresent()

Простейшей проверкой на дебаг является функция WinAPI IsDebuggerPresent(). Данная проверка была помещена в самом начале функции main. При наличии отладчика программа завершается с кодом -1.

Данную проверку можно обойти, если поместить 0 в поле BeingDebugged в PEB:

moveax, dwordptrfs:[0x30]

movbyte ptrds:[eax+2], 0

### NtGlobalFlag

Как было сказано выше, проверка IsDebuggerPresent() может быть взломана, к тому же она находится в самом начале кода, что сразу бросается в глаза. Поэтому необходимо реализовать ещё несколько проверок и разместить их по всему коду.

PEB (Process Environment Block) является структурой данных в семействе операционных систем Windows NT. Это структура данных, большинство полей которой не предназначены для использования ничем, кроме операционной системы. DWORD значение в PEB хранит параметры запуска процесса [HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\ Control\Session Manager\GlobalFlag].

При отладке используются следующие флаги:

FLG\_HEAP\_ENABLE\_TAIL\_CHECK (0x10),

FLG\_HEAP\_ENABLE\_FREE\_CHECK(0x20),

FLG\_HEAP\_VALIDATE\_PARAMETERS(0x40).

void CheckNtGlobalFlag()

{

unsigned long NtGlobalFlag = 0;

\_\_asm {

mov eax,fs:[30h]

mov eax, [eax + 68h]

mov NtGlobalFlag, eax

}

if (NtGlobalFlag & 0x70)

{

std::cout << XOR("@afqccav$`apagpa`%") << std::endl;

exit(-1);

}

}

Данная проверка была внедрена после функции проверки пароля. Обход такого способа осуществляется путём установки в 0 значения поля NtGlobalFlag в PEB до проверки его программой.

### Trap Flag (TF)

Trap Flag это один из флагов в регистре EFLAGS. Если TF установлен в 1, то процессор после выполнения каждой инструкции будет генерировать INT 01h или «Single step» исключение, то есть производится защита от отладки по шагам (Рисунок 7).

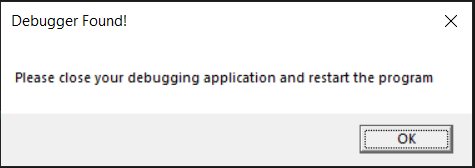


Рисунок 7 ― Реализация средства антиотладки TrapFlag

inline bool TrapFlagCheck()

{

BOOL isDebugged = TRUE;

\_\_try

{

\_\_asm

{

pushfd

or dword ptr[esp], 0x100 // set the Trap Flag

popfd // Load the value into EFLAGS register

nop

}

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

// If an exception has been raised – debugger is not present

isDebugged = FALSE;

}

if (isDebugged)

{

MessageBoxW(NULL, L"Please close your debugging application and restart the program", L"Debugger Found!", 0);

ExitProcess(0);

}

}

Данная проверка была реализована в полезном функционале в методе get(), который получает информацию об уровне защиты. Для обхода данного метода во время отладки необходимо поставить точку останова после кода проверки, перепрыгнув инструкцию PUSHFD. После выполнения проверки трассировка программы может быть продолжена.

### NtQueryInformationProcess: ProcessDebugPort0x07

Данный метод реализован с помощью функции CheckRemoteDebuggerPresent(), которая возвращает булево значение наличия отладчика (сообщение аналогично Рисунку 7).

void ProcessDebugPort() {

BOOL IsDbgPresent = FALSE;

CheckRemoteDebuggerPresent(GetCurrentProcess(), &IsDbgPresent);

if (IsDbgPresent)

{

MessageBoxW(NULL, L"Please close your debugging application and restart the program", L"Debugger Found!", 0);

ExitProcess(0);

}

}

Для обхода такой защиты необходимо подменить значение возвращаемое функцией NtQueryInformationProcess. Также можно использовать хукинг: внедрить dll в отлаживаемый процесс и установить обработчик, перехватывающий вызов функции (hook).

## Средства антидизассемблирования

Дизассемблер — это транслятор, преобразующий машинный код, объектный файл или библиотечные модули в текст программы на языке ассемблера. Соответственно необходимо усложнить анализ ассемблерного кода.

1. Выше уже были описаны мусорные функции на языке С++,
2. Была реализована дополнительная функция проверки пароля, расположенная рядом с основной проверкой:

bool check\_pass() {

std::string welcome\_message = "Ajpav$tewwskv`>";

std::string pass = "tvmimpahefq";

std::string correct = "Gkvvagp%";

std::string wrong = "Svkjc%";

if (password == pass)

return TRUE;

else

return FALSE;

}

1. Были реализованы мусорные ассемблерные вставки и распределены по всему полезному коду. Такие вставки затруднят анализ кода на языке ассемблер.

\_\_asm {

push eax

xor eax, eax

setpo al

push edx

xor edx, eax

sal edx, 2

xchg eax, edx

inc edx

dec edx

pop edx

or eax, ecx

pop eax

}

## Упаковка файла

### UPX

В качестве упаковщика была выбрана утилита UPX. Данный упаковщик поддерживает множество форматов исполняемых файлов как для Windows, так и для Linux. UPX использует алгоритм сжатия UCL, который относится к классу алгоритмов сжатия без потерь. То есть закодированные данные однозначно могут быть восстановлены с точностью до бита.

Скачаем данную утилиту, распакуем на диске С:\ и туда же поместим исходный файл. В командной строке под управлением администратора вызовем команду «upx my.exe», после чего исходный файл будет упакован (Рисунок 8).

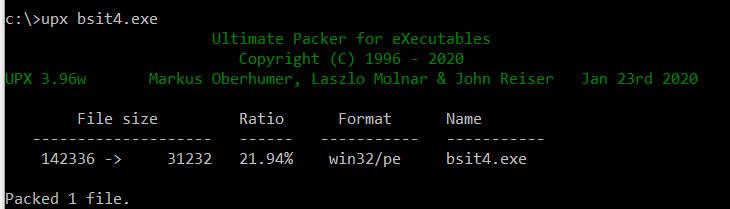


Рисунок 8 ― Упаковка файла

Упакованный файл скрывает исполняемый код программы, сохраняя этот код с помощью какого-либо типа сжатия или шифрования. Он также добавляет секцию в памяти, откуда будет распакован исходный код для дальнейшего выполнения. Рассмотрим упакованный файл в IDA. Как видно из Рисунка 9, часть кода скрыта.

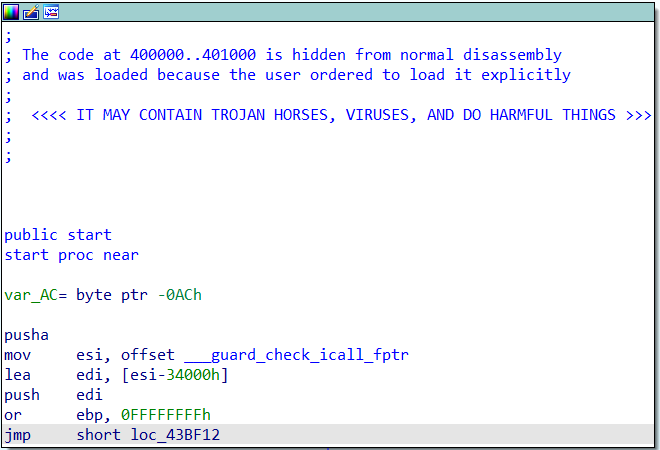


Рисунок 9 ― Изучение упакованного файла

Если обратиться к коду по адресу 400000..401000, то можно увидеть такое сообщение (Рисунок 10). При этом сама IDA не предоставляет никаких логических блоков с кодом по этим адресам.

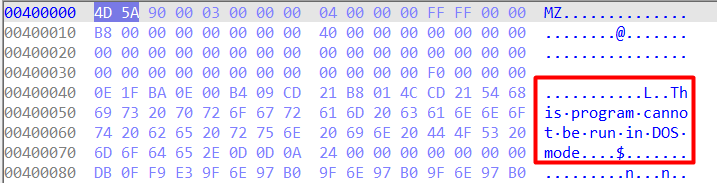


Рисунок 10 ― Изучение сегментов памяти

Рассмотрим сегменты файла (Рисунок 11).

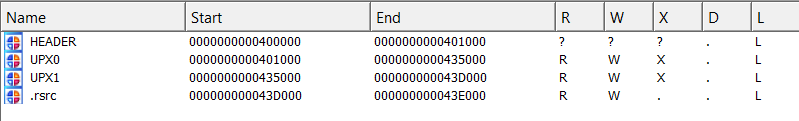


Рисунок 11 ― Сегменты файла в памяти

По адресу 0х0043C0A0 находится блок, представленный на Рисунке 12. Команда jmp отсылает на распакованный код, который будет выполнен в конце работы всего загрузчика. Соответственно предполагаем, что именно по адресу 0х4119C4 будет находиться код исходной программы.

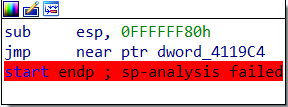


Рисунок 12 ― Адрес распакованного кода

### ASPack

В качестве второго упаковщика была выбрана утилита ASPack.

Преимущества:

* Данная программа имеет внутренний архиватор, который эффективнее ZIP на 10–20%.
* Сжатие защищает программы от дизасемблирования и декомпиляции.
* Упаковщик полностью совместим с программами, которые созданы с помощью [Microsoft Visual C++](https://ru.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B), [Visual Basic](https://ru.wikipedia.org/wiki/Visual_Basic" \o "Visual Basic), [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)" \o "Delphi (среда разработки)), [C++ Builder](https://ru.wikipedia.org/wiki/C%2B%2B_Builder).
* Позволяет создать [резервную копию файла](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (BAK файл) перед началом упаковки.

Недостатки:

* [Закрытый исходный код](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4).
* Существуют распаковщики ASPackDie, RL!deASPack и другие.
* При проверке упакованных приложений некоторые антивирусы могут выдавать предупреждающие сообщения.

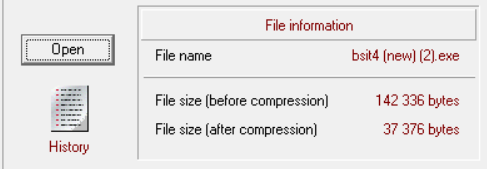
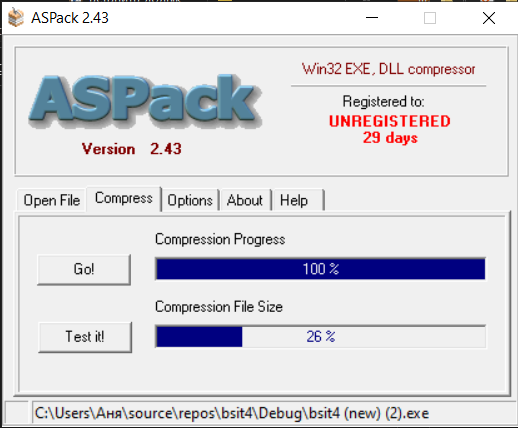


Рисунок 13 ― Упакованный файл

### Энтропия кода

Энтропия означает хаос, следовательно, энтропия программного кода отражает сложность анализа кода. С точки зрения разработки ПО энтропия должна быть минимальной, что свидетельствует о «чистом» коде. Однако данный показатель никогда не может быть равен 0. Суть ана­лиза энтро­пии зак­люча­ется в том, что в ском­пилиро­ван­ном фай­ле обыч­ной прог­раммы учас­тки кода рас­пре­деле­ны более‑менее рав­номер­но. При исполь­зовании кодиров­щиков или обфуска­ции, упа­ков­щиков, алго­рит­мов сжа­тия или вста­вок подоб­ного рода кода в исходный файл такая рав­номер­ность наруша­ется. В фай­ле появ­ляют­ся высоко­энтро­пий­ные области (белый шум) и области, менее под­вер­гну­тые обфуска­ции или шиф­рованию (кодиро­ванию).

Для подсчёта показателя энтропии файла используется метод скользящего окна по каждому байту файла. Сна­чала рас­счи­тыва­ются час­тоты fi появ­ления для каж­дого воз­можно­го зна­чения бай­та (i = 0..255). За­тем най­ден­ные час­тоты fi сум­миру­ются по фор­муле ниже, и в резуль­тате получается зна­чение энтро­пии.

[https://st768.s3.eu-central-1.amazonaws.com/0d172265ccd0603c088afc6010f40ab7/17606/formula.png](https://st768.s3.eu-central-1.amazonaws.com/0d172265ccd0603c088afc6010f40ab7/17606/formula.png)

Од­но из харак­терных свой­ств алго­рит­мов сжа­тия — перерас­пре­деле­ние час­тот встре­чаемос­ти бай­тов кода, что и ста­нет замет­но при ана­лизе. В таких фай­лах будет высокая сте­пень энтро­пии, близ­кая к мак­сималь­ному зна­чению 8 (28 = 256). Для закоди­рован­ных (сжа­тых, зашиф­рован­ных) фай­лов на прак­тике зна­чение энтро­пии свы­ше семи мож­но счи­тать приз­наком при­мене­ния пре­обра­зова­ния кода, в то вре­мя как обыч­ные фай­лы име­ют энтро­пию 2-6.

С помощью утилиты Detect It Easy (DIE) проанализируем энтропию исходного файла и упакованных двумя разными упаковщиками (Рисунки 14, 15, 16).

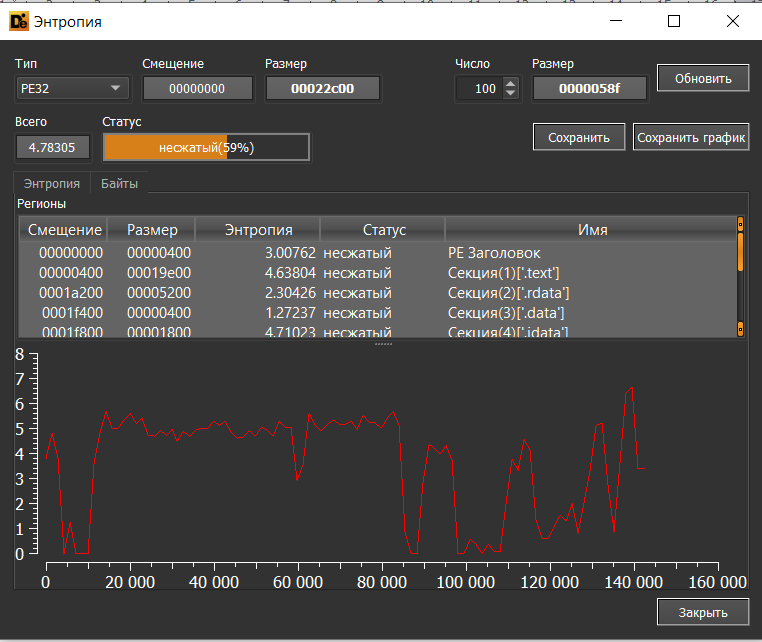


Рисунок 14 ― Энтропия исходного файла

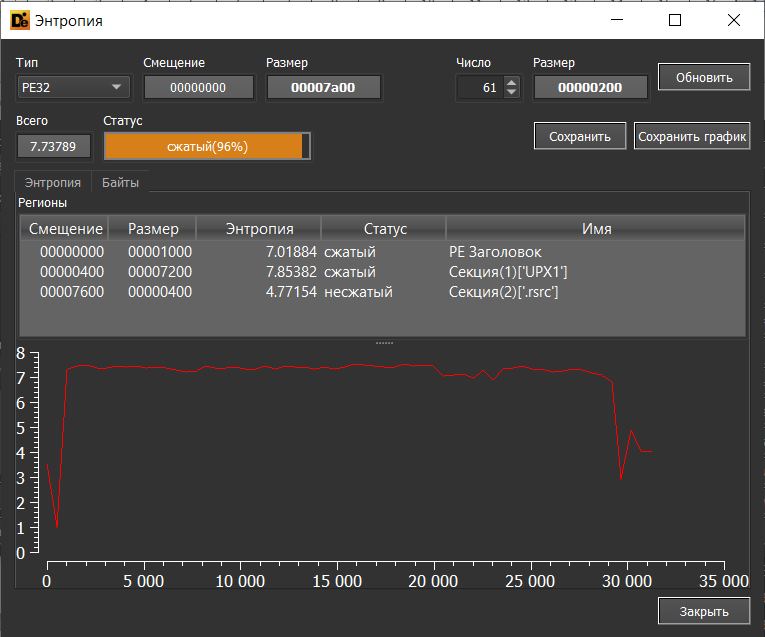


Рисунок 15 ― Энтропия файла после сжатия UPX

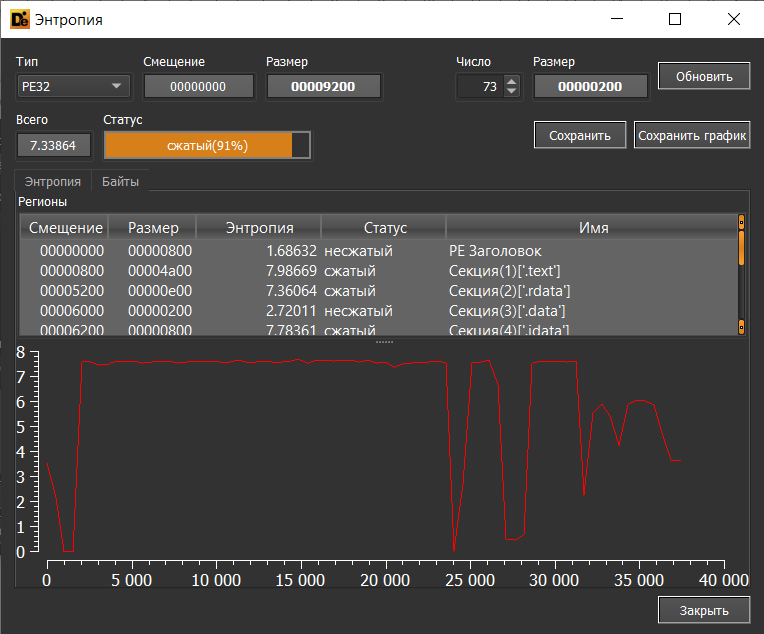


Рисунок 16 ― Энтропия файла после сжатия ASPack

ВЫВОД

В ходе выполнения данной лабораторной работы были приобретены навыки защиты ПО и реверс инжиниринга. Были изучены средства антиотладки и антидизассемблирования, а также способы их обхода. Данные методы защиты были реализованы на практике. Также был изучен такой механизм защиты, как упаковка файлов. Был изучен алгоритм упаковки и распаковки, проанализирован загрузчик упакованного файла. В ходе сравнения двух различных программ упаковки UPX и ASPack первая утилита лучше по проценту сжатия файла и по показателю энтропии кода, хотя различия не слишком критичны.

приложение 1

**Main.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <Wscapi.h>

#include <string>

#include <vector>

#include <ctime>

#include <string>

#pragma comment(lib, "Wscapi.lib")

#include "sec.h"

int main()

{

while (1)

{

Status Current;

Current.get();

if (change(Current))

{

Stats.push\_back(Current);

Current.print();

}

if (Stats.size() > 100)

Stats.erase(Stats.begin());

Sleep(1000);

}

return 0;

}

bool change(Status new\_stat)

{

if (Stats.empty())

return true;

else

if

(

Stats[Stats.size() - 1].Antispyware != new\_stat.Antispyware ||

Stats[Stats.size() - 1].All != new\_stat.All ||

Stats[Stats.size() - 1].Antivirus != new\_stat.Antivirus ||

Stats[Stats.size() - 1].AutoupdateSettings != new\_stat.AutoupdateSettings ||

Stats[Stats.size() - 1].Firewall != new\_stat.Firewall ||

Stats[Stats.size() - 1].InternetSettings != new\_stat.InternetSettings ||

Stats[Stats.size() - 1].None != new\_stat.None ||

Stats[Stats.size() - 1].UserAccountCcontrol != new\_stat.UserAccountCcontrol ||

Stats[Stats.size() - 1].Service != new\_stat.Service

)

return true;

else

return false;

}

**Sec.cpp**

#include "sec.h"

Status::Status(void)

{

this->Firewall = std::string();

this->AutoupdateSettings = std::string();

this->Antivirus = std::string();

this->Antispyware = std::string();

this->InternetSettings = std::string();

this->UserAccountCcontrol = std::string();

this->Service = std::string();

this->None = std::string();

this->All = std::string();

//this->timestamp = std::string();

}

int clear()

{

HANDLE hStdOut;

hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

// Fetch existing console mode so we correctly add a flag and not turn off others

DWORD mode = 0;

if (!GetConsoleMode(hStdOut, &mode))

{

return ::GetLastError();

}

// Hold original mode to restore on exit to be cooperative with other command-line apps.

const DWORD originalMode = mode;

mode |= ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING;

// Try to set the mode.

if (!SetConsoleMode(hStdOut, mode))

{

return ::GetLastError();

}

// Write the sequence for clearing the display.

DWORD written = 0;

PCWSTR sequence = L"\x1b[2J";

if (!WriteConsoleW(hStdOut, sequence, (DWORD)wcslen(sequence), &written, NULL))

{

// If we fail, try to restore the mode on the way out.

SetConsoleMode(hStdOut, originalMode);

return ::GetLastError();

}

// To also clear the scroll back, emit L"\x1b[3J" as well.

// 2J only clears the visible window and 3J only clears the scroll back.

// Restore the mode on the way out to be nice to other command-line applications.

SetConsoleMode(hStdOut, originalMode);

COORD homeCoords = { 0, 0 };

SetConsoleCursorPosition(hStdOut, homeCoords);

return 0;

}

void Status::print()

{

//std::cout << "Information from " << this->timestamp;

//clear();

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD homeCoords = { 0, 0 };

SetConsoleCursorPosition(hStdOut, homeCoords);

for (unsigned int i = 0; i < 1024; i++)

{

std::cout << " ";

}

SetConsoleCursorPosition(hStdOut, homeCoords);

std::cout << " FIREWALL: " << this->Firewall << std::endl;

std::cout << " AUTOUPDATE: " << this->AutoupdateSettings << std::endl;

std::cout << " ANTIVIRUS: " << this->Antivirus << std::endl;

std::cout << " ANTISPYWARE: " << this->Antispyware << std::endl;

std::cout << " INTERNET SETTINGS: " << this->InternetSettings << std::endl;

std::cout << " UAC: " << this->UserAccountCcontrol << std::endl;

std::cout << " SERVICE: " << this->Service << std::endl;

std::cout << " ALL: " << this->All << std::endl;

std::cout << std::endl;

}

void Status::get()

{

WSC\_SECURITY\_PROVIDER\_HEALTH health;

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_FIREWALL, &health))

this->Firewall = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_AUTOUPDATE\_SETTINGS, &health))

this->AutoupdateSettings = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_ANTIVIRUS, &health))

this->Antivirus = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_ANTISPYWARE, &health))

this->Antispyware = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_INTERNET\_SETTINGS, &health))

this->InternetSettings = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_USER\_ACCOUNT\_CONTROL, &health))

this->UserAccountCcontrol = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_SERVICE, &health))

this->Service = win\_to\_human(health);

if (S\_OK == WscGetSecurityProviderHealth(WSC\_SECURITY\_PROVIDER\_ALL, &health))

this->All = win\_to\_human(health);

//this->date();

}

/\*void Status::date()

{

time\_t rawtime;

struct tm\* timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

this->timestamp = asctime(timeinfo);

}\*/

const char\* Status::win\_to\_human(WSC\_SECURITY\_PROVIDER\_HEALTH status)

{

switch (status)

{

case WSC\_SECURITY\_PROVIDER\_HEALTH\_GOOD:

return "GOOD";

case WSC\_SECURITY\_PROVIDER\_HEALTH\_NOTMONITORED:

return "NOTMONITORED";

case WSC\_SECURITY\_PROVIDER\_HEALTH\_POOR:

return "POOR";

case WSC\_SECURITY\_PROVIDER\_HEALTH\_SNOOZE:

return "SNOOZE";

default:

return "Status Error";

}

}