**Лабораторная работа № 1 – Изучение методов защиты программного обеспечения**

**Цель работы**

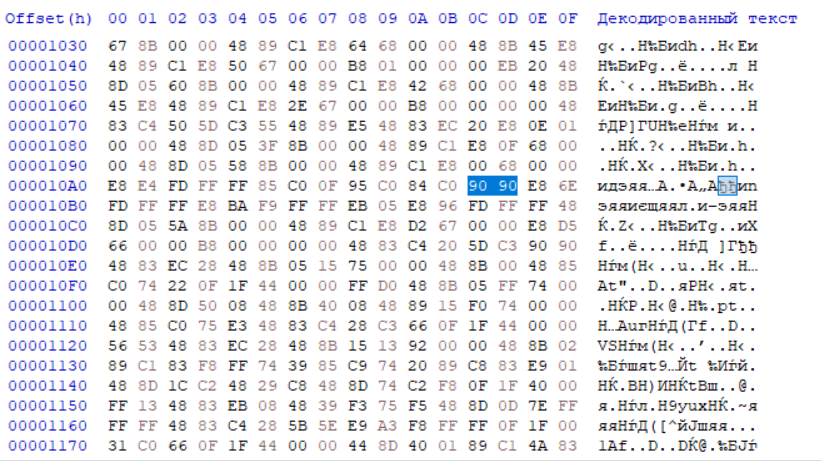
Изучение методов защиты программного обеспечения и возможных способов их преодоления.

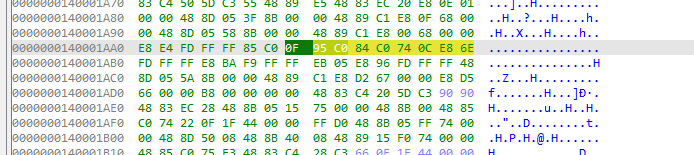
**Формулировка задания**

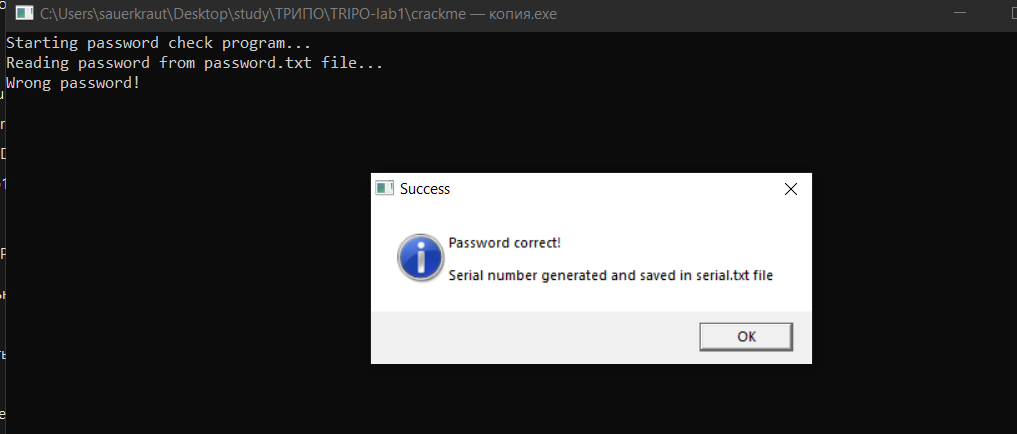
В ходе выполнения лабораторной работы необходимо выполнить следующие действия:

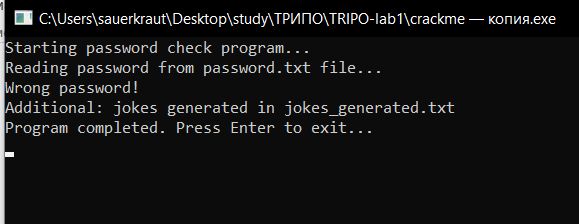
1. Изучить подходы, применяемые для защиты программного обеспечения [1-17].
2. Реализовать программу (crackme) на компилируемом языке (C\C++, Go, Rust, Swift), осуществляющую считывание пароля из файла “password.txt” и в случае совпадения его с заданным, вывод сформированного серийного номера в формате KEY$xxxxxxxxxx$ (x – произвольные символы) в файл “serial.txt”, а также вывод сообщения об успехе в интерфейсе программы, в обратном случае - сообщения об ошибке. Все файлы должны быть расположены в одной директории с исполняемым файлом программы.
3. В качестве защищаемого программного обеспечения (ПО) может быть выбрана программа, ранее разработанная студентом, либо любое ПО с открытым исходным кодом.
4. Разрабатываемая вами программа будет **частью** лабораторной работы №2, в которой необходимо будет решить crackme одногруппника. Соответственно, вам необходимо выбрать напарника (либо получить распределение у преподавателя), с которым нужно будет обменяться исполняемыми файлами (без предоставления исходного кода) разработанной программы для возможности дальнейшего их анализа и решения (в рамках Л.Р. №2).
5. В программе должен быть «полезный функционал», выполнение которого не должно происходить при введении неправильного пароля.
6. Осуществить модификацию исполняемого файла (бинарный патчинг) таким образом, чтобы результат проверки пароля всегда был положительным.

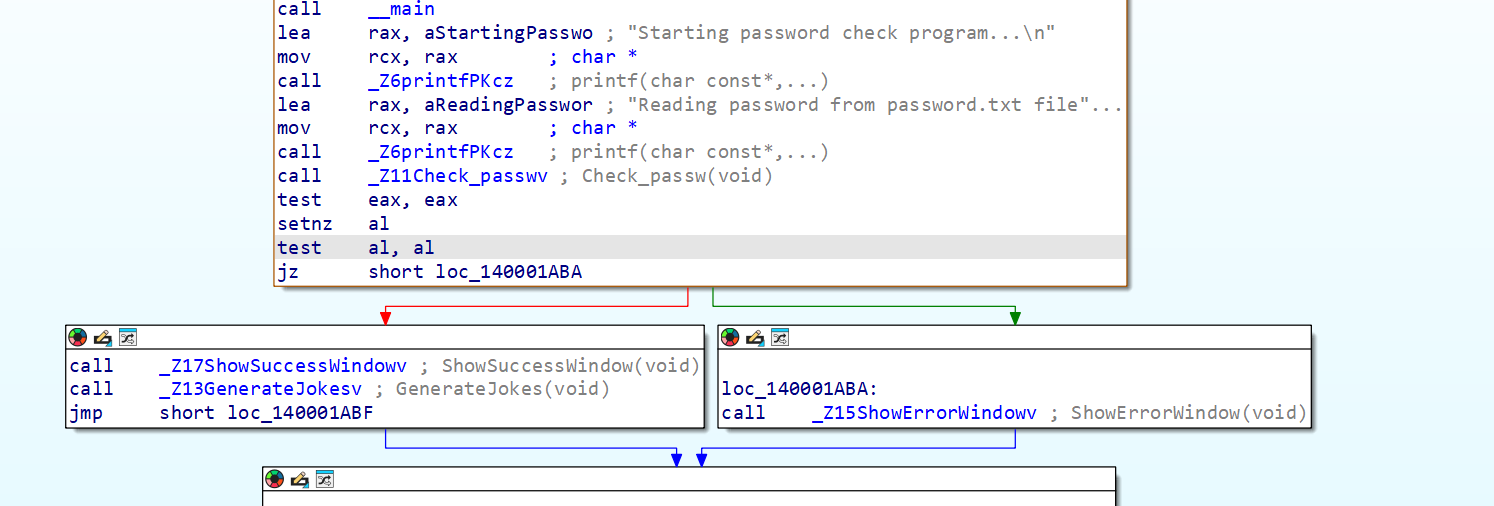
Замена jz short loc\_7FF6B16C1ABA на два nop nop те 74 OC на 90 90

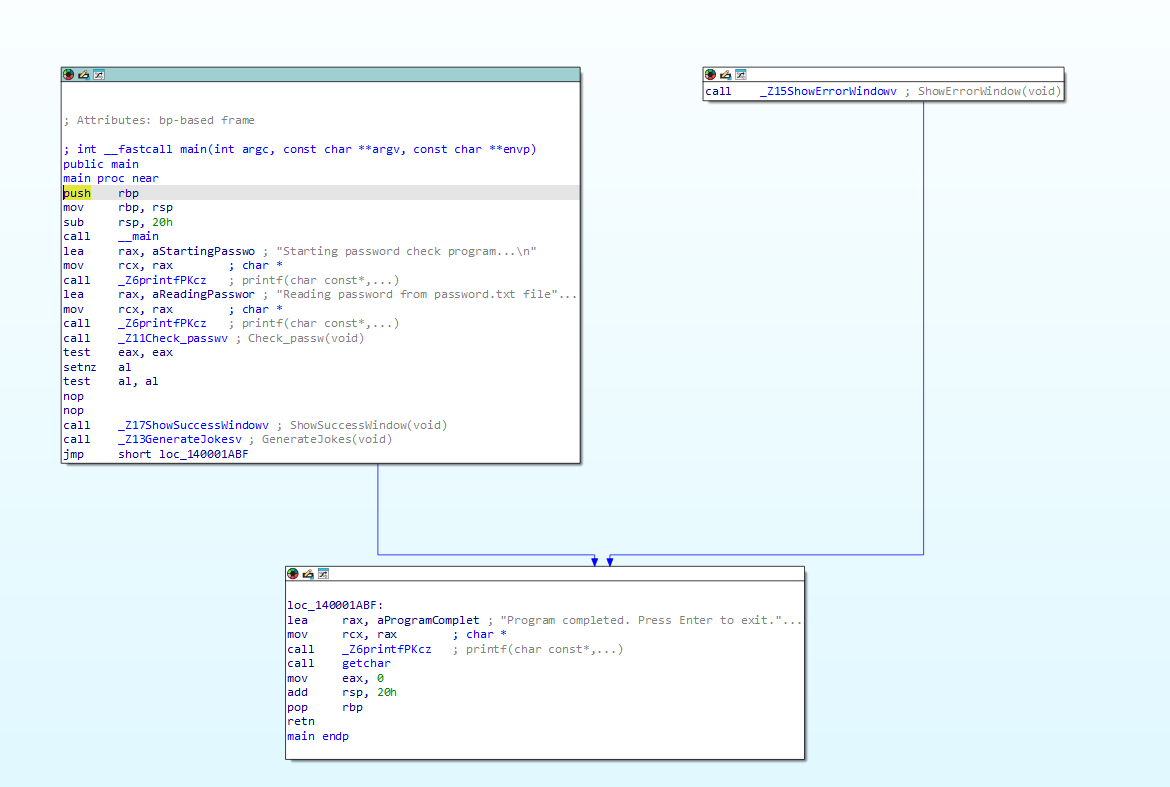










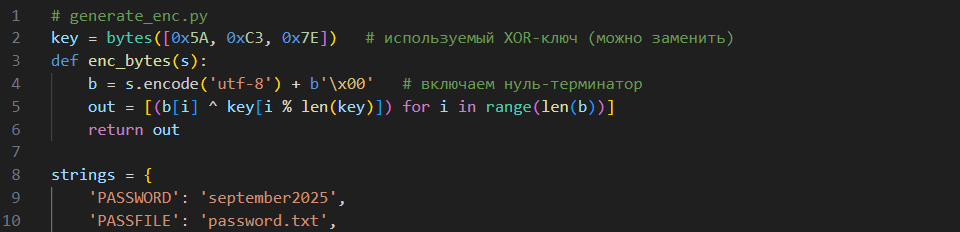
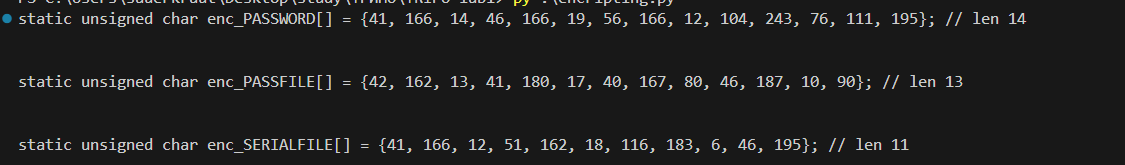
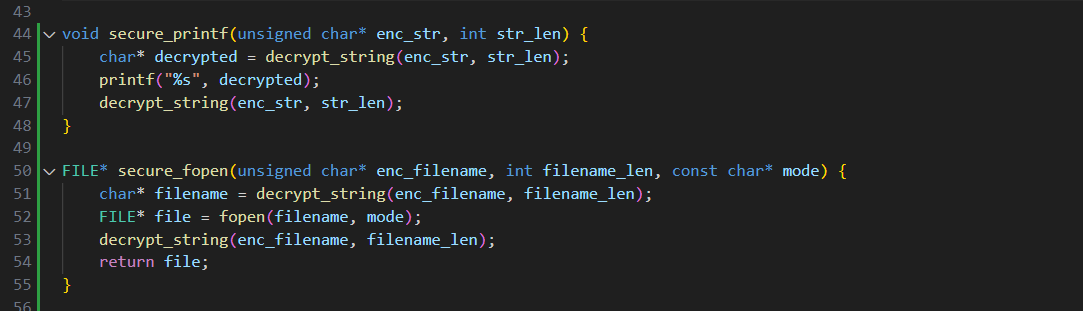


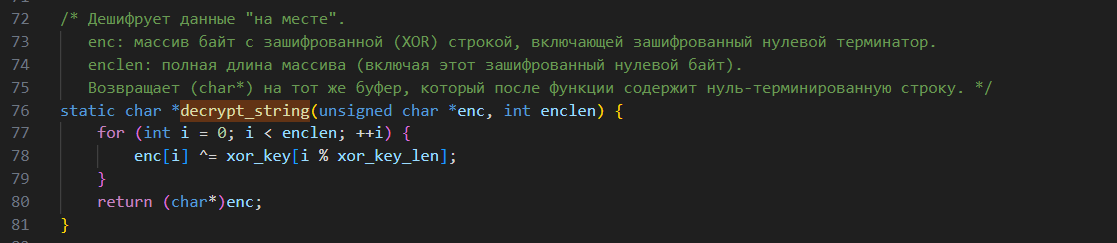
1. В отчете привести описание осуществленных действий для нахождения места проверки пароля и модификации файла.
2. Необходимо реализовать защиту разработанного программного обеспечения от возможности изменения поведения программы при помощи бинарного патчинга.
3. Реализованная программа должна использовать следующие общие подходы к защите ПО:
   1. Хранение в зашифрованном виде всех строк, в т.ч. выводимых на экран.

Отлично — реализуем **шифрование всех строк в программе** (включая выводимые и имена файлов) так, чтобы при запуске проги строки дешифровались в памяти и программа работала как раньше, а в бинарнике на диске строки хранились в зашифрованном виде.

# Идея (принцип работы)

* На этапе подготовки (локально) мы шифруем каждую строку с коротким ключом XOR (можно заменить на любой другой симметричный шифр).
* В исходнике вместо открытых литералов помещаем **массивы байт** — зашифрованные данные (включая завершающий '\0'), т.е. static unsigned char enc\_SOME[] = { ... };.
* В рантайме (до первого использования) мы вызываем функцию decrypt(enc\_SOME, size) — она XOR'ит байты на месте и возвращает char\* на расшифрованную строку.
* Для удобства добавляем decrypt\_all() — она дешифрует все массивы при старте main() (или перед первым использованием).
* Для выводов в printf и для файлов используем указатели (const char\*)enc\_NAME после дешифровки.
* Для MessageBoxW (Unicode) — если хочешь показывать wide-строки, конвертируем расшифрованную UTF-8 в UTF-16 перед вызовом (я покажу пример).

Был реализован скрипт на питоне который шифрует строки и выводит   
  
  
полученные строки потом были встроены в код crackme, были дописаны функции дешифровки (она и обратной шифровки) и функции для безопасного использования этих шифрованных строк  




* 1. Проверка и вывод информации в разных местах программы.
  2. Контроль целостности участков кода, ответственных за проверку «пароля» (подсчет CRC исполняемого кода функции проверки пароля **в оперативной памяти** во время выполнения программы).
  3. Наличие нескольких проверок пароля в разных местах программы.
  4. Наличие ложных проверок (в т.ч. сразу после считывания «пароля»).
  5. Использование методов запутывания кода для усложнения анализа кода программы.

1. Защитные механизмы должны быть тесно переплетены с логикой защищаемого кода и распределены по всему «полезному» коду, в случае модификации кода защитных механизмов код полезной нагрузки должен стать некорректным.
2. Реализованная программа должна использовать:
   1. Не менее 4 методов обнаружения средств отладки и противодействия им.
   2. Не менее 3 методов противодействия дизассемблированию (методы, которые приводят к некорректному дизассемблированию кода, см. главу 15 [1]).
   3. Не менее 2 методов выявления виртуальных машин.
   4. Фрагменты самомодифицирующегося кода.
   5. Методы бинарной обфускации и обфускации потока передачи управления (нелинейная передача управления, например, через исключения [2].
3. Для реализованной программы произвести обход внедренных механизмов защиты путем бинарного патчинга исполняемого кода.
4. Произвести упаковку исполняемого файла с использованием одного из распространенных упаковщиков исполняемых файлов (например, UPX).
5. Провести анализ принципов работы используемого средства защиты бинарного кода с помощью дизассемблера, отладчика и других программных средств. Найти в отладчике оригинальную точку входа в программу после упаковки файла.
6. Сравнить параметры оригинального и упакованного файлов (энтропия).
7. В отчете необходимо привести следующую информацию:

* исходный код программы до внесения в нее механизмов защиты;
* описание полезной функциональности программы;
* описание процесса модификации программы без механизмов защиты;
* перечень механизмов защиты и фрагменты исходного кода, отвечающие за их реализацию;
* описание принципа действия используемых механизмов защиты;
* исходный код программы после добавления механизмов защиты;
* описание процесса обхода реализованных механизмов защиты;
* описание программы-упаковщика;
* описание алгоритма упаковки исполняемого файла;
* фрагмент кода упакованной программы, отвечающий за самораспаковку исполняемого файла (загрузчик), полученный с помощью дизассемблера;
* схема структуры упакованного файла;
* параметры исполняемых файлов до и после упаковки (размер файла, энтропия и т.п.).

**Список источников**

1. Michael Sikorski, Andrew Honig. Practical Malware Analysis

<https://doc.lagout.org/security/Malware%20%26%20Forensics/Practical%20Malware%20Analysis.pdf>

1. Binary Deobfuscation. <https://calwa.re/reversing/obfuscation/binary-deobfuscation-preface#indirect-branches>
2. Анти-дизассемблерные приемы

<https://1malware1.medium.com/anti-disassembly-techniques-e012338f2ae0>

1. ANTI-DISASSEMBLY

<http://staff.ustc.edu.cn/~bjhua/courses/security/2014/readings/anti-disas.pdf>

1. Примеры используемых антиотладочных приемов в ВПО

<https://github.com/rrbranco/blackhat2012/blob/master/blackhat2012-paper.pdf>

<https://github.com/rrbranco/blackhat2012>

1. Introduction Into Windows Anti-Debugging

<http://www.codeproject.com/Articles/29469/Introduction-Into-Windows-Anti-Debugging>

1. An Anti-Reverse Engineering Guide

<http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide>

1. Anti-Debug Protection Techniques: Implementation and Neutralization

<http://www.codeproject.com/Articles/1090943/Anti-Debug-Protection-Techniques-Implementation-an>

1. OpenRCE Anti Reverse Engineering Techniques Database

<http://www.openrce.org/reference_library/anti_reversing>

1. Anti Reverse Engineering Protection Techniques to Use Before Releasing Software

<https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software>

1. The art of unpacking

<https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf>

1. Anti-Memory Dumping Techniques

<http://resources.infosecinstitute.com/anti-memory-dumping-techniques/>

1. Об упаковщиках в последний раз: Часть первая - теоретическая

<http://uinc.ru/articles/41/>

1. Разработка своего PE-упаковщика

<https://habrahabr.ru/company/xakep/blog/139138/>

1. Empirical identification, comparison and circumvention of current Antivirus detection techniques

<https://www.blackhat.com/docs/us-14/materials/us-14-Mesbahi-One-Packer-To-Rule-Them-All-WP.pdf>

1. Strings Obfuscation System

<http://www.codeproject.com/Articles/502283/Strings-Obfuscation-System>

1. Binary Obfuscation

<https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-sean_taylor-binary_obfuscation.pdf>