**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.
7. В отчете привести описание осуществленных действий для нахождения места проверки пароля и модификации файла.
8. Необходимо реализовать защиту разработанного программного обеспечения от возможности изменения поведения программы при помощи бинарного патчинга.
9. Реализованная программа должна использовать следующие общие подходы к защите ПО:
   1. Хранение в зашифрованном виде всех строк, в т.ч. выводимых на экран.
   2. Проверка и вывод информации в разных местах программы.
   3. Контроль целостности участков кода, ответственных за проверку «пароля» (подсчет CRC исполняемого кода функции проверки пароля **в оперативной памяти** во время выполнения программы).
   4. Наличие нескольких проверок пароля в разных местах программы.
   5. Наличие ложных проверок (в т.ч. сразу после считывания «пароля»).
   6. Использование методов запутывания кода для усложнения анализа кода программы.
10. 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.
11. 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].
12. Для реализованной программы произвести обход внедренных механизмов защиты путем бинарного патчинга исполняемого кода.
13. Произвести упаковку исполняемого файла с использованием одного из распространенных упаковщиков исполняемых файлов (например, UPX).
14. 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.
15. Сравнить параметры оригинального и упакованного файлов (энтропия).
16. 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.).

Коротко — да: **нужно связать вашу «crackme»-функцию с выбранным защищаемым ПО так, чтобы полезный функционал разрешался (выполнялся) только при корректном пароле**. Ниже — чёткое, практическое руководство: что делать, какие варианты интеграции возможны, пример кода и рекомендации по оформлению отчёта и подготовке к ЛР-2 (обмену бинарниками и бинарный патчинг).

# Что именно требуется по заданию (важно)

* Программа-crackme должна проверять пароль и при успехе записывать серийник KEY$xxxxxxxxxx$ в serial.txt и выводить сообщение об успехе.
* В программе должен быть **«полезный функционал»**, который **не выполняется при неправильном пароле**. То есть проверка пароля должна контролировать запуск этого функционала.
* В ЛР-2 вы будете решать crackme одногруппника — поэтому вы должны подготовить исполняемый файл (без исходников) и обменяться ими.

# Варианты связки crackme с ПО

Выберите один из подходов (в порядке простоты):

1. **Интеграция в одну программу (рекомендуется для учебной работы).**  
   Добавьте проверку пароля в начало main()/в точку входа: если пароль правильный — выполнить полезную работу; если нет — вывести ошибку и выйти. Это самый прямой и однозначный способ, который выполняет требование «полезный функционал блокируется».
2. **Обёртка (launcher).**  
   Сделайте отдельный исполняемый файл-launcher, который делает проверку, затем при успехе запускает основной модуль (внутри той же директории) — либо через динамическую загрузку (dlopen/LoadLibrary), либо просто через exec/CreateProcess. Полезно, если вы не хотите вносить изменения в исходники защищаемого ПО.
3. **Плагин / библиотека.**  
   Перенесите проверку пароля в библиотеку, которую вызывает защищаемая программа перед выполнением ключевых функций. Подходит, если у вас есть доступ к исходникам ПО и вы хотите более модульную архитектуру.
4. **Инструментальное встраивание (hook).**  
   Менее тривиально: вставить вызов проверки в конкретную точку исполняемого файла (patch), но для ЛР это, как правило, не нужно — достаточно первого или второго варианта.

# Пример «правильной» интеграции (C++ — минимально)

Это пример программы, где полезный функционал выполняется только при успешной проверке пароля. Код компактный — вставьте вашу логику вместо useful\_function().

#include <iostream>

#include <fstream>

#include <string>

// функция проверки пароля (читает password.txt и при совпадении пишет serial.txt)

bool check\_password\_and\_write\_serial() {

const std::string PASSWORD = "september2025";

std::ifstream fin("password.txt");

if (!fin) return false;

std::string s;

if (!std::getline(fin, s)) return false;

// trim newline/spaces

while(!s.empty() && (s.back()=='\r' || s.back()=='\n' || s.back()==' ' || s.back()=='\t')) s.pop\_back();

if (s != PASSWORD) return false;

// генерируем простой серийник (для примера)

std::ofstream fout("serial.txt");

fout << "KEY$1234567890$";

return true;

}

void useful\_function() {

std::cout << "Выполняется полезный функционал программы.\n";

// здесь — основная работа вашего ПО

}

int main(){

if (check\_password\_and\_write\_serial()) {

std::cout << "Right password\n";

useful\_function();

} else {

std::cout << "Wrong password\n";

// полезный функционал НЕ выполняется

}

return 0;

}

# Практические советы по реализации и тестированию

* **Все файлы** (exe, password.txt, serial.txt) — в одной директории, как требует задание.
* Тщательно обрабатывайте перевод строки в password.txt (trim), иначе сравнение может не совпасть.
* Сделать «полезный функционал» — это может быть любая заметная операция (запись выходного файла, расчёт, простая GUI/консольная утилита), чтобы в ЛР-2 было что тестировать.

# Подготовка к пунктам 6–7 (бинарный патчинг и отчёт)

В ЛР-1 вам нужно также подготовиться к пунктам лабораторной (6 и 7) — позже вы будете делать бинарный патчинг так, чтобы проверка пароля всегда проходила. Для отчёта полезно заранее подготовить следующее:

1. **Соберите исходную и скомпилированную версию** (exe).
2. **Опишите, где происходит проверка пароля**: файл/функция, строка, последовательность вызовов.
3. **Инструменты, которыми будете пользоваться** — перечислите (напр., strings, objdump/nm/readelf, IDA/Ghidra, x64dbg/WinDbg, Hex editor).
4. **При бинарном патчинге** обычно делают одно из:
   * заменить условный переход (jne/je) так, чтобы условие всегда было истинным/ложным,
   * заменить код, возвращающий false, на возврат true,
   * поставить NOP-ы поверх проверочного блока и вставить фиксированный mov eax,1 (в зависимости от архитектуры).  
     В отчёте укажите **адрес/смещение**, **старые байты** и **новые байты** (hex), а также почему именно этих байт достаточно.
5. **Документируйте**: снимки экрана с местом в дизассемблере, вывод strings, шаги в отладчике (breakpoint → значение регистра → изменение памяти), финальная проверка — что теперь полезный функционал запускается без правильного пароля.
6. **Подготовьте два бинарника**: оригинал и запатченный (так вы сможете обмениваться/тестировать и в ЛР-2).

В отчёте важно не только показать, что вы смогли «взломать», но и описать методику: как вы нашли проверку, какие следы искали (строки, сравнения, вызовы strcmp/memcmp), и почему ваш патч меняет поведение.

# Что включить в отчёт по ЛР-1 (рекомендуемая структура)

1. Цель и формулировка задания.
2. Описание выбранного ПО (если использовали ранее разработанное приложение) или краткое описание вашей новой программы.
3. Исходный код (или ссылка/приложение).
4. Инструкция сборки и запуска.
5. Описание реализации проверки пароля и «полезного функционала» (со скриншотами).
6. План действий для ЛР-2 (с кем обменяетесь; что будет предоставлено — только exe).
7. Список инструментов и методика бинарного патчинга (пункты 6–7 задания).
8. Приложения: password.txt (пример), serial.txt (пример), оригинальный exe, запатченный exe (по требованию преподавателя), снимки экрана.

# Дополнительные рекомендации / частые ошибки

* Не забывайте про права доступа к файлам (чтобы программа могла писать serial.txt).
* Для Windows: убедитесь, что антивирус не блокирует исполняемый файл при обмене бинарниками.
* Чтобы ЛР-2 было честно: не включайте в exe подсказки (комментарии) о расположении проверки; но в отчёте опишите, как вы это потом находили.

Если хочешь, могу прямо сейчас:

* помочь встроить проверку в конкретную вашу программу (вставить пример кода в её main()/пункт входа) — пришли исходник/описание функционала;
* или подготовить пример отчёта и чек-лист действий для бинарного патчинга (список команд objdump, strings, какие места смотреть) — дам пошагово без «эксплойтов», только учебный реверс-анализ и доказательство.

Что предпочитаете: интеграция в вашу существующую программу (пришлите краткий код/функции), или шаблон отчёта + чек-лист для поиска и патчинга?

**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>