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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**«Изучение методов защиты программного обеспечения»**

1. по дисциплине «Технологии реверс-инжиниринга программного обеспечения»
2. Выполнил
3. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. Овасапян Т.Д.

<*подпись*>

1. Санкт-Петербург
2. 2022
3. **Цель работы**

Изучение методов защиты программного обеспечения и возможных способов их преодоления.

1. **Задачи**
2. Реализовать программу 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. Выбрать защищаемое ПО. В нем должен быть полезный функционал, который станет доступным в случае успешного прохождения проверки пароля.
4. Осуществить модификацию исполняемого файла (бинарный патчинг) таким образом, чтобы результат проверки пароля всегда был положительным.
5. Реализовать защиту разработанного программного обеспечения от возможности изменения поведения программы при помощи бинарного патчинга со следующими подходами 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.
   7. 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.
6. 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).
   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).
7. Для реализованной программы произвести обход внедренных механизмов защиты путем бинарного патчинга исполняемого кода
8. Произвести упаковку исполняемого файла с использованием одного из распространенных упаковщиков исполняемых файлов (например, UPX).
9. 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.
10. Сравнить параметры оригинального и упакованного файлов (энтропия).
11. **Ход работы**
    1. **Реализация программы для считывания пароля**

Была реализована программа для считывания пароля с консоли. Пароль сверяется с паролем, хранящимся в файле «password.txt». При вводе правильного пароля пользователю выводится сообщение об успешно введенном пароле, после чего в файл «serial.txt» выводится сформированный серийный номер в заданном формате. При вводе некорректного пароля, пользователю предлагается попробовать ввести пароль снова или выйти из программы. Интерфейс разработанной программы представлен на рисунке 1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Интерфейс разработанной программы проверки пароля

Исполняемый файл находится в одной директории с файлами «password.txt» и «serial.txt». Директория представлена на рисунке 2.

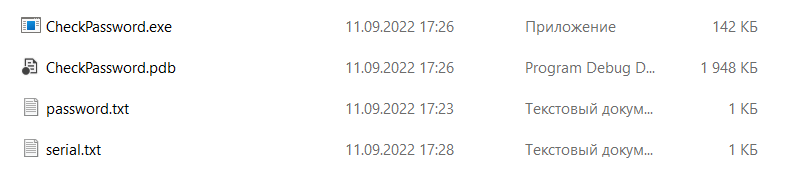


Рисунок 2 – Содержимое директории с исполняемым файлом

После успешного ввода пароля в файл «serial.txt» был записан сформированный серийный номер. Содержимое файла «serial.txt» представлено на рисунке 3.

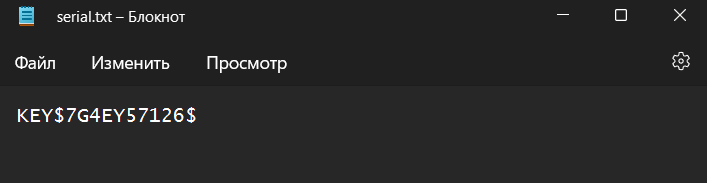


Рисунок 3 – Содержимое файла «serial.txt»

В качестве произвольных символов выбираются либо большие буквы латинского алфавита, либо цифры.

* 1. **Выбор защищаемого ПО**

В качестве защищаемого ПО была выбрана ранее разработанная программа для нахождения обратной матрицы к заданной. Для упрощения тестирования, пользователю предлагается ввести только размер матрицы, без ввода самой матрицы (матрица будет формироваться случайным образом). Исходные коды данной программы были добавлены в проект программы проверки пароля. Далее в программу проверки пароля был добавлен вызов функции для поиска обратной матрице при правильно введенном пароле. На рисунке 4 представлен пример выполнения полезной нагрузки при правильно введенном пароле.

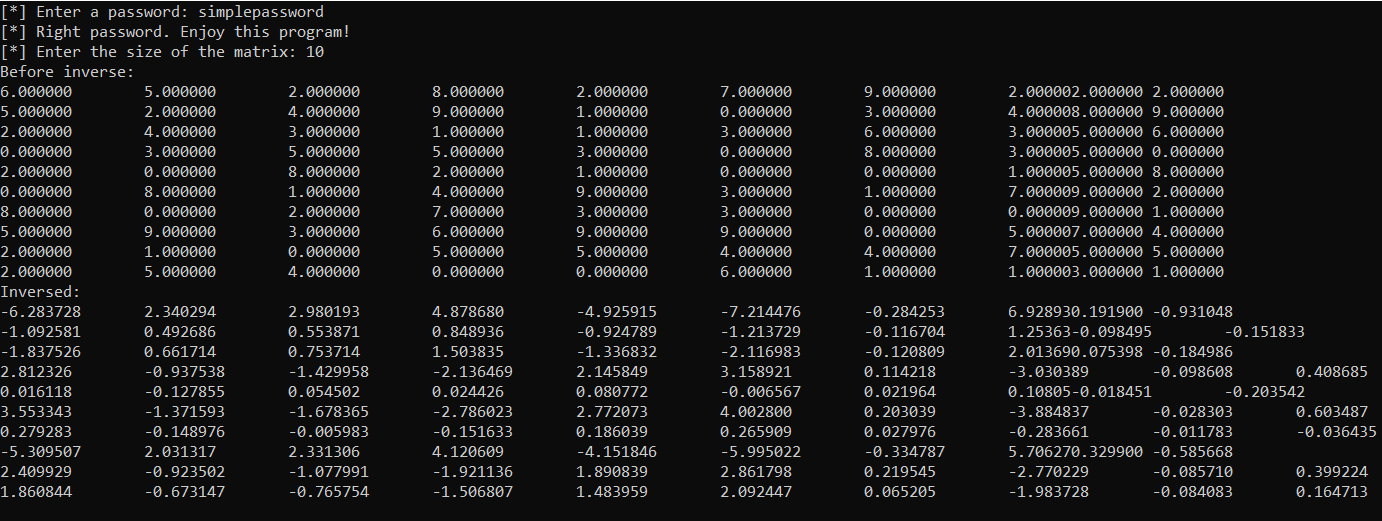


Рисунок 4 – Выполнение полезной нагрузки при правильно введенном пароле

* 1. **Бинарный патчинг**

Исполняемый файл был просмотрен с помощью дизассемблера IDA-Pro. Был произведен поиск слова «password», был найден фрагмент кода, в котором пользователю предлагается ввести пароль, после чего происходит проверка введенного пароля. На рисунке 5 выделен фрагмент проверки введенного пароля, который делает/не делает переход после введения пароля. Этот фрагмент кода и интересен, необходимо изменить логику на противоположную (поменять jz на je), чтобы при вводе неверного пароля программа давала доступ к полезной нагрузке.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Фрагмент дизассемблированного кода с проверкой правильности введенного пароля

Далее с помощь HIEW-редактора была изменена команда условного перехода (рисунок 6) на противоположную (jnz).

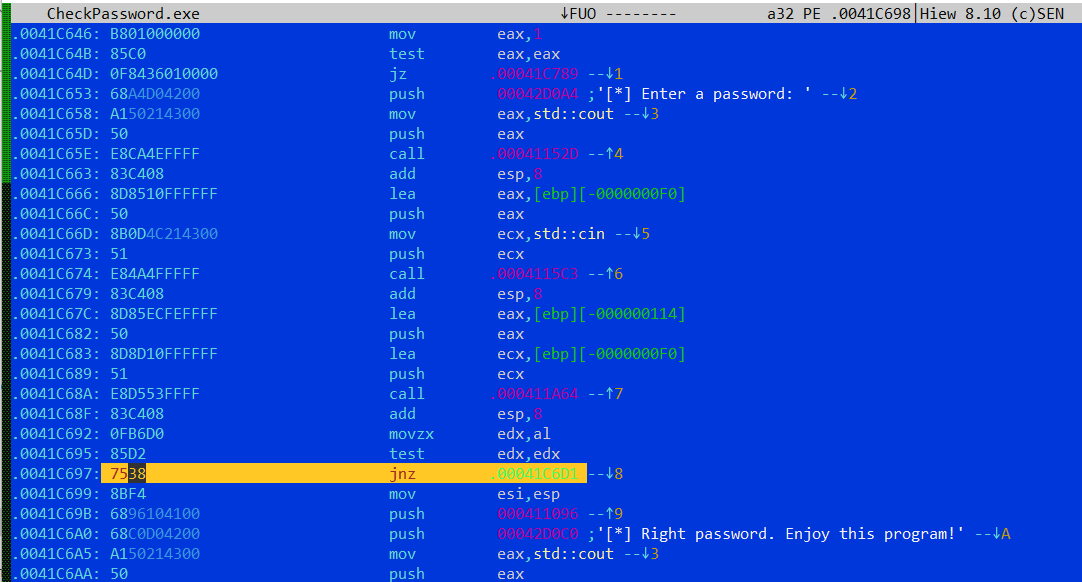


Рисунок 6 – Изменение команды условного перехода в HIEW-редакторе

В результате программа начала принимать все неправильные пароли. Пример поведения программы приведен на рисунке 7.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 – Проверка пароля после бинарного пачтинга

* 1. **Защита разработанного ПО**
     1. **Шифрование строк**

Для разработанного ПО была реализована защита. Все строки были зашифрованы и в открытом виде более не хранились в коде. На рисунке 8 приведен пример хранения и обработки строки перед ее использованием. Функция get\_string выполняет операцию XOR с числом 10 каждого символа строки, после чего выводится сама строка в расшифрованном виде.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – Пример хранения и использования строки

Пароль также более не хранится в открытом виде. Он также проходит шифрование с помощью функции get\_string. На рисунке 9 представлен файл с паролем в зашифрованном виде.

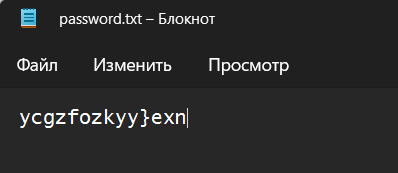


Рисунок 9 – Файл с паролем

Серийный номер также шифруется после ввода корректного пароля. Пример зашифрованного серийного номера представлен на рисунке 10.

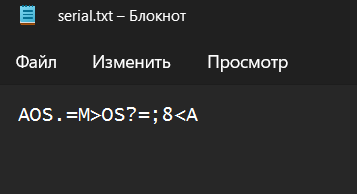


Рисунок 10 – Пример зашифрованного серийного номера

* + 1. **Проверка и вывод информации в разных местах программы**

Проверка и вывод информации после проверки (успешный ввод пароля или ошибка) в дизассемблированном коде идет подряд, из-за чего довольно легко проследить логику работы программы. Для усложнения анализа были написаны функции print\_info и type\_data, которые будут вызываться инструкцией call и усложнять анализ, так как в дизассемблированном коде нововведенные функции будут находиться в новых местах. На рисунке 11 представлен пример кода с применением ввода и вывода информации в разных местах программы.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Пример кода с вводом и выводом информации в разных местах программы

Как можно заметить, помимо выноса ввода и вывода в разные функции, информация теперь выводится как можно позже, благодаря чему после проверки делается много других действий (на рисунке 11 после проверки пароля, если он корректный, будет сначала создаваться серийный номер, а только потом выводиться информация о том, что пароль корректный).

* + 1. **Подсчет CRC исполняемого кода функции проверки пароля**

Для подсчета CRC в дизассемблированном коде был посчитан размер функции. Для этого был посмотрен адрес начала функции и конца, после чего посчитан размер в байтах. Далее был написан код подсчета CRC функции проверки пароля. Листинг функции представлен на рисунке 12. Функция узнает адрес функции проверки пароля, далее суммирует значения 691 байтов (размер функции), после чего сравнивает с числом 66095 (оригинальный CRC функции проверки пароля).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Функция проверки CRC

* + 1. **Наличие нескольких проверок пароля в разных местах программы**

Для проверки пароля в нескольких местах программы была создана глобальная переменная gPassword, где хранится введенный пользователем пароль для возможности проверить его в любом месте программы. Также был вынесен фрагмент проверки пароля в отдельную функцию. Проверка CRC была переделана под новую функцию.

Проверки пароля были добавлены сразу после вывода информации о получении доступа к функционалу. Также после ввода размера матрицы и перед выводом найденной обратной матрицы.

* + 1. **Наличие ложных проверок**

Была добавлена ложная проверка пароля сразу после ввода, которая сверяет пароль с фиксированным и пишет, что пароль верный, если совпадает. Никаких дополнительных действий не делается, благодаря чему логика работы программы не ломается. Единственное, что может произойти, это вывод сообщения «Right password» при неправильно введенном пароле. Выводная строка с паролем специально не шифровалась, чтобы запутать исследователя. На рисунке 13 представлен фрагмент кода с ложной проверкой.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 13 – Пример ложной проверки пароля

Также была добавлена проверка пароля после проверки пароля при неправильно введенном пароле. На рисунке 14 представлен листинг второй проверки.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 – Листинг ложной проверки пароля

Как видно на рисунке 14, выводимая после проверки информация хранится в зашифрованном виде. Это сделано для усложнения анализа, так как такой фрагмент также можно принять за реальную проверку.

* + 1. **Запутывание кода**

Так как исходный код программы не доступен пользователю, то имеет смысл использовать только методы запутывания, которые будут добавлять лишнюю логику работы программы в дизассемблированный код: добавление лишних переменных и лишних действий. На рисунке 15 представлен листинг ввода пароля с добавленной логикой программы.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 – Листинг с запутыванием кода

В данном фрагменте кода введенный пароль преобразуется в формат json с полем «password», далее сохраняется оригинальный введенный пользователем пароль, после чего глобальная переменная для проверки пароля принимает значение преобразованного пароля, после чего происходит проверка пароля – если она корректная, то будет передано управление функции print\_info, которая просто выводит информацию о том, что пароль верный (функция не будет вызываться). После проверки переменная для проверки реального пароля получает оригинальное значение и программа продолжает свое выполнение.

* + 1. **Защита от средств отладки**

Для обнаружения средств отладки использовалась функция IsDebuggerPresent, которая возвратит ненулевое значение, если процесс выполняется в контексте отладчика. На рисунке 16 приведен листинг программы. В случае ненулевого результата программа сразу завершается.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 – Листинг с обнаружением отладчика

Данный метод можно обойти, поместив 0х00 в BeingDebugged в PEB.

Также была добавлена проверка параметров запуска процесса. Листинг проверки представлен на рисунке 17.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 – Листинг проверки параметров запуска процесса

Данный метод можно обойти установкой NtGlobalFlags в 0x00 в PEB до проверки его программой.

Также была написана проверка TF: Trap Flag (рисунок 18). Если этот флаг установлен, то CPU после выполнения каждой инструкции будет генерировать INT 01h. Данный флаг нельзя поменять напрямую – этим и пользуется данный код: в нем делается операция логического или с 0x100 (установка TF), если TF установлен, то ничего не произойдет. Если TF сброшен, то произойдет исключение, так как нельзя менять данный флаг самостоятельно. Таким образом, при сброшенном TF произойдет исключение, которое сообщит о том, что отладчика нет.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 18 – Обнаружение отладчика с помощью флага TF

Данный метод можно обойти, поставив точку останова на pushfd. Точка останова вызывает прерывание INT1, которое сбрасывает флаг TF, благодаря чему будет вызвано исключение и трассировку можно будет продолжить.

Также была добавлена проверка открытых окон на 2 отладчика – OllyDebug и WinDebug (рисунок 19).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 19 – Фрагмент кода проверки открытых окон

Если окно отладчика будет открыто, то оно будет обнаружено и программа завершится. Обойти данную проверку можно используя другой отладчик, или произвести бинарные патчинги проверки.

* + 1. **Противодействие дизассемблированию**

Была написана функция jump\_opcode (рисунок 20).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 20 – Функция jump\_opcode

Суть метода заключается в том, чтобы вставить байт (\_\_emit 0xe9) перед инструкцией mov ebx, eax. Так как дизассемблер будет читать файл побайтово, то он прочитает 0xe9xxxxxxxx – что даст в результате инструкцию jmp, которая и будет выведена.

Далее был написан фрагмент кода, который не делает полезных действий, но противодействует дизассемблированию (рисунок 21).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21 – inc и dec для противодействия дизассемблированию

Суть метода заключается в том, чтобы вставить байт 0xc0 перед инструкцией dec – это второй байт инструкции inc (ff c0). Переход происходит на 1 байт до res, поэтому jmp res-1 будет восприниматься как eb ff, а сам переход будет производиться на ff и срабатывать будет просто inc eax, dec eax. Для дизассемблера increment будет скрыт, вместо него будет находиться лишний байт.

* + 1. **Выявление виртуальных машин**

Выявить виртуальную машину можно с помощью проверки имени супервизора. Для этого была реализована функция IsVM, которая получает эту информацию, после чего сравнивает с именами популярных виртуальных машин. На рисунке 22 представлен фрагмент сравнения полученного имени с популярными именами.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 22 – Сравнение имени супервизора

Для нормальной работы практически все виртуальные машины требуют установки дополнений к гостевой операционной системе, например VBoxGuestAddition для VirtualBox или Parallels Tools для Parallels Workstation. Без этих дополнений работа с виртуальной машиной несколько затруднительна. А эти дополнения оставляют след в виде запущенных процессов. Была написана функция для проверки работы на VMWare.

* + 1. **Самомодифицирующийся код**

Для написания самомодифицирующегося кода использовалась функция WriteProcessMemory, которая позволяет перезаписывать байты процесса. Для примера был реализован бесконечный цикл (рисунок 23), в котором вызвалась функция модификации кода и изменила инструкцию je на jne, что позволило выйти из бесконечного цикла.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 23 – Самомодифицирующийся код

* + 1. **Методы бинарной обфускации и обфускации потока передачи управления**

Была написана функция, в которой выполняется код с делением на 0 для вызова исключения. В исключении будет вызываться функция для проверки контрольной суммы. Таким образом реализуется обфускация потока передачи управления через исключение. Код представлен на рисунке 24.

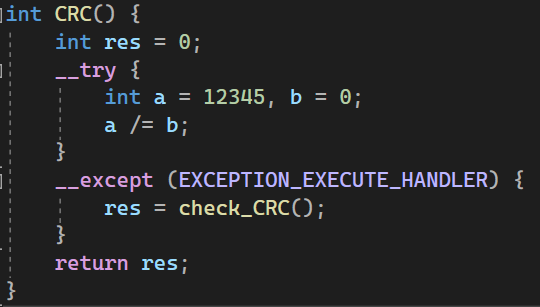


Рисунок 24 – Обфускация потока передачи управления через исключение

Другой метод бинарной обфускации – обфускация указателей функций. Суть метода заключается в том, чтобы не вызывать функции, а использовать их адреса как переменные. На рисунке 25 представлен фрагмент кода с данным методом.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25 – Обфускация с помощью указателя на функцию

В результате инструкции, которые распознает дизассемблер будут другими. В оригинале кода в инструкциях был бы call jump\_opcode. С данным методом адрес функции jump\_opcode будет записан в регистр и будет произведен jmp по этому адресу. А оттуда будет произведен ret в место вызова function\_pointers.

* 1. **Бинарный патчинг**

Консольные программы для взаимодействия с пользователем используют текстовые сообщения, которые выводят на экран. Если у исполняемого файла нет дополнительных файлов с наборами строк, то эти строки хранятся внутри исполняемого файла как константные. Был произведен поиск каки-либо строк в дизассемблированном файле. Рядом с этими строками просматривались инструкции условных переходов или сравнений. Если таковые находились, то код анализировался точнее – смотрелся сформированный код на Си (F5 в IDA Pro), смотрелась блок-схема, которая получается при нажатии на пробел в IDA Pro. Далее менялись условные переходы и смотрелось на результат работы программы.

Изначально было изменено возвращаемое значение функции, которая выводит «[!] Wrong password …». Теперь, при возвращении оттуда через «-end» результат был положительным. Но это не привело к изменениям работы программы – значит, есть дополнительная проверка после этого фрагмента.

Был произведен поиск фразы, состоящей из 38 символов (столько символов в фразе об успехе). Был произведен поиск функции, откуда вызывается функция с выводом этой фразы. Там был найден условный переход – он был изменен. После этого на экране после ввода «-end» появилась фраза об успешно введенном пароле, но программа сразу же завершалась.

После анализа функции с выводом сообщения об успешно введенном пароле, было найдено место условного перехода, который завершал работу почти сразу же после вывода сообщения – был изменен условный переход. После этого программа позволила ввести размер матрицы и вывела сформированную матрицу. Но после этого завершилась, не выводя обратную матрицу. Условный переход основывался на вызове функции – эта же функция вызывалась и ранее – скорее всего, это основная функция проверки пароля.

Данная функция была рассмотрена и произведен поиск вызова этой функции – она вызывалась еще в одном месте, перед выводом результата. Условный переход был изменен и программа сработала. Результат работы программы после бинарного патчинга представлен на рисунке 26.

В результате, скорее всего можно было сделать бинарный патчинг только последней функции, так как она основная, с помощью которой происходит проверка пароля. Для защиты от такого патчинга, можно использовать разные функции проверки пароля в разных местах.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 – Результат работы защищенной программы после бинарного патчинга

* 1. **Упаковка исполняемого файла**

Был установлен UPX и с его помощью был сжат защищенный исполняемый файл. Для выполнения следующего пункта был изучен процесс работы упаковщика.

Упаковщик считывает массив исполняемых байтов, упаковывает их без потерь, далее в соответствии с форматом PE дописывает сжатый исполняемый массив к загрузчику.

Загрузчик – это программа, которая ищет в своем теле сжатый массив, распаковывает его и запускает.

* 1. **Анализ принципов работы используемого средства защиты бинарного кода**

Для поиска оригинальной точки входа в программу необходимо найти адрес начала сжатого массива. Этот адрес ищется загрузчиков в зависимости от реализации, но самый простой вариант – ввести константные значения переменным, в которых будет лежать адрес начала сжатого массива, и потом упаковщиком при имплантировании исполняемого кода в загрузчик найти эти значения и заменить их на адрес массива.

Для поиска оригинальной точки входа программы использовался отладчик OllyDBG. Проверки в самом коде происходят уже в функции main – то есть оригинальной точке входа программы, которую и надо найти.

В OllyDBG был загружен исполняемый файл. Программа изначально остановлена на входной точке. По умолчанию стояли hardware accept на все регистры в правой панели – это означает, что при доступе к регистрам будет вызван breakpoint. На F9 была запущена отладка и первый breakpoint вызвался на изменение регистра esp, в котором хранится указатель на вершину стека. Далее значение записывалось в регистр ebx, после чего происходил вызов функции по адресу ebx. Отладчик был запущен на F7 на каждый шаг и вскоре программа выполнила проверку на наличие отладчика и завершилась (данное действие выполняется уже внутри main). Таким образом момент с вызовом по адресу ebx и есть оригинальной входной точкой программы. Фрагмент найденного кода представлен на картинке 27.

Изображение выглядит как стол

Автоматически созданное описание

Картинка 27 – Фрагмент найденной точки входа программы

* 1. **Сравнение оригинального и упакованного файлов (энтропия)**

Энтропия – показатель случайности распределений байтов в файле. Энтропия высчитывается по следующей формуле:



Где  – найденные частоты.

В скомпилированном файле обычной программы все байты распределены довольно равнозначно, и энтропия будет от 2 до 6.

В модифицированном файле энтропия достигает 7 и выше.

С помощью программы Detect It Easy можно анализировать исполняемые файлы, в том числе посмотреть их энтропию. На рисунке 28 представлена энтропия защищенного файла. На рисунке 29 энтропия защищенного и сжатого файла.

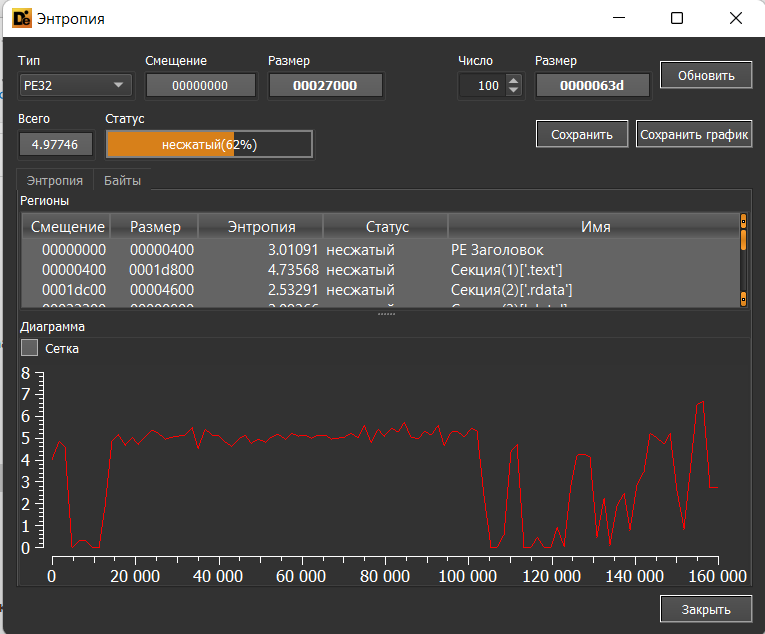


Рисунок 28 – Энтропия защищенного файла

Изображение выглядит как текст, снимок экрана, монитор, черный

Автоматически созданное описание

Рисунок 29 – Энтропия упакованного файла

Как видно из рисунков, энтропия защищенного файла находится в пределах обычного скомпилированного файла. А упакованного в пределах упакованного, что и заметно.

1. **Выводы**

В ходе выполнения данной лабораторной работы были изучены механизмы защиты программного обеспечения и возможные методы обхода защиты. Эти методы были реализованы и проанализированы. Был произведен обход некоторых методов на практике. Также был изучен процесс работы упаковщиков и сравнены упакованный и оригинальный файлы (с помощью энтропии).

**Приложение А**

Листинг «main.cpp» до модификации

#include "main.h"

int main()

{

//ввод размера матрицы

cout << "Enter the size of the matrix: ";

int matrix\_size;

cin >> matrix\_size;

vector<vector<double>> matrix;

vector<vector<double>> identity;

matrix.resize(matrix\_size);

identity.resize(matrix\_size);

//заполняем матрицы

srand(time(NULL));

for (int i = 0; i < matrix\_size; i++)

for (int j = 0; j < matrix\_size; j++)

{

identity[i].push\_back(i == j ? 1.0 : 0.0);

matrix[i].push\_back(rand() % 10);

}

cout << "Before inverse:\n";

print(matrix, matrix\_size);

double time = clock();

//нахождение обратной матрицы

inverse(matrix, identity, matrix\_size);

time = (clock()-time) / CLOCKS\_PER\_SEC;

cout << "Inversed:\n";

print(identity, matrix\_size);

//cout << "Time of work: " << time << endl;

return 0;

}

void print(vector<vector<double>> &matrix, int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

cout <<fixed<< matrix[i][j] << "\t";

cout << endl;

}

}

// Поставить на место row строку с наибольшим ведущим коэффициентом

void pivotize(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int row) {

int max\_index = row;

double max\_value = fabs(matrix[row][row]);

double current\_value;

for (int i = row + 1; i < n; i++)

{

current\_value = fabs(matrix[i][row]);

if (current\_value > max\_value)

{

max\_index = i;

max\_value = current\_value;

}

}

//меняем строки местами

if (row != max\_index)

{

swap(matrix[row], matrix[max\_index]);

swap(identity[row], identity[max\_index]);

}

}

// Разделить строку матрицы на значение

void devide(vector<vector<double>>& matrix, int n, int i, double denominator)

{

for (int j = 0; j < n; j++)

matrix[i][j] /= denominator;

}

// Обнуление элементов под ведущим элементом в строке x

void subtract\_below(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int x) {

double coeff;

for (int i = x + 1; i < n; i++)

{

coeff = matrix[i][x] / matrix[x][x];

for (int j = x; j < n; j++)

matrix[i][j] -= coeff \* matrix[x][j];

for (int j = 0; j < n; j++)

identity[i][j] -= coeff \* identity[x][j];

}

}

// Обнуление элементов над ведущим элементом в строке x

void subtract\_above(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int x)

{

double coeff;

for (int i = x - 1; i >= 0; i--)

{

//обнуляем элементы выде главной диагонали матрицы для левой части расширенной матрицы

coeff = matrix[i][x] / matrix[x][x];

for (int j = x; j >= 0; j--)

matrix[i][j] -= coeff \* matrix[x][j];

//делаем тоже самое для правой матрицы расширенной матрицы

for (int j = 0; j < n; j++)

identity[i][j] -= coeff \* identity[x][j];

}

}

// Обращение матрицы методом Гаусса-Жордана

void inverse(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n)

{

//с помощью элементарных преобразований над матрицами получим верхнюю треугольную матрицу

for (int i = 0; i < n - 1; i++)

{

// Обмен строк

pivotize(matrix, identity, n, i);

// Обнулить элементы ниже ведущего

subtract\_below(matrix, identity, n, i);

}

//с помощью элементарных преобразований получим диагональную матрицу

for (int i = n - 1; i > 0; i--)

// Обнулить элементы выше ведущего

subtract\_above(matrix, identity, n, i);

// Разделить строки на ведущие элементы

for (int i = 0; i < n; i++)

{

devide(identity, n, i, matrix[i][i]);

devide(matrix, n, i, matrix[i][i]);

}

}

**Приложение Б**

Листинг «main.cpp» после модификации

#include "main.h"

#include "password.h"

int main\_act()

{

//ввод размера матрицы

string info1 = "Q W\*Od~ox\*~bo\*ycpo\*el\*~bo\*gk~xcr0\*";

info1 = get\_string(info1);

cout << info1;

int matrix\_size;

cin >> matrix\_size;

if (!get\_password())

return 0;

vector<vector<double>> matrix;

vector<vector<double>> identity;

matrix.resize(matrix\_size);

identity.resize(matrix\_size);

//заполняем матрицы

srand(time(NULL));

for (int i = 0; i < matrix\_size; i++)

for (int j = 0; j < matrix\_size; j++)

{

identity[i].push\_back(i == j ? 1.0 : 0.0);

matrix[i].push\_back(rand() % 10);

}

string info2 = "Q W\*Holexo0";

info2 = get\_string(info2);

cout << info2 << endl;

print(matrix, matrix\_size);

double time = clock();

//нахождение обратной матрицы

inverse(matrix, identity, matrix\_size);

time = (clock()-time) / CLOCKS\_PER\_SEC;

string info3 = "Q W\*Cd|oxyon0";

info3 = get\_string(info3);

if (!get\_password())

return 0;

cout << info3 << endl;

print(identity, matrix\_size);

//cout << "Time of work: " << time << endl;

return 0;

}

void print(vector<vector<double>> &matrix, int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

cout <<fixed<< matrix[i][j] << "\t";

cout << endl;

}

}

// Поставить на место row строку с наибольшим ведущим коэффициентом

void pivotize(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int row) {

int max\_index = row;

double max\_value = fabs(matrix[row][row]);

double current\_value;

for (int i = row + 1; i < n; i++)

{

current\_value = fabs(matrix[i][row]);

if (current\_value > max\_value)

{

max\_index = i;

max\_value = current\_value;

}

}

//меняем строки местами

if (row != max\_index)

{

swap(matrix[row], matrix[max\_index]);

swap(identity[row], identity[max\_index]);

}

}

// Разделить строку матрицы на значение

void devide(vector<vector<double>>& matrix, int n, int i, double denominator)

{

for (int j = 0; j < n; j++)

matrix[i][j] /= denominator;

}

// Обнуление элементов под ведущим элементом в строке x

void subtract\_below(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int x) {

double coeff;

for (int i = x + 1; i < n; i++)

{

coeff = matrix[i][x] / matrix[x][x];

for (int j = x; j < n; j++)

matrix[i][j] -= coeff \* matrix[x][j];

for (int j = 0; j < n; j++)

identity[i][j] -= coeff \* identity[x][j];

}

}

// Обнуление элементов над ведущим элементом в строке x

void subtract\_above(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n, int x)

{

double coeff;

for (int i = x - 1; i >= 0; i--)

{

//обнуляем элементы выде главной диагонали матрицы для левой части расширенной матрицы

coeff = matrix[i][x] / matrix[x][x];

for (int j = x; j >= 0; j--)

matrix[i][j] -= coeff \* matrix[x][j];

//делаем тоже самое для правой матрицы расширенной матрицы

for (int j = 0; j < n; j++)

identity[i][j] -= coeff \* identity[x][j];

}

}

// Обращение матрицы методом Гаусса-Жордана

void inverse(vector<vector<double>>& matrix, vector<vector<double>>& identity, int n)

{

//с помощью элементарных преобразований над матрицами получим верхнюю треугольную матрицу

for (int i = 0; i < n - 1; i++)

{

// Обмен строк

pivotize(matrix, identity, n, i);

// Обнулить элементы ниже ведущего

subtract\_below(matrix, identity, n, i);

}

//с помощью элементарных преобразований получим диагональную матрицу

for (int i = n - 1; i > 0; i--)

// Обнулить элементы выше ведущего

subtract\_above(matrix, identity, n, i);

// Разделить строки на ведущие элементы

for (int i = 0; i < n; i++)

{

devide(identity, n, i, matrix[i][i]);

devide(matrix, n, i, matrix[i][i]);

}

}

**Приложение В**

Листинг «check\_password.cpp»

//#include <debugapi.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include "main.h"

#include "password.h"

#include <string>

#include <intrin.h>

#include <Tlhelp32.h>

int check\_debugged();

void jump\_opcode();

bool IsVM2();

bool IsVM();

int WriteMe(void\* addr, int wb);

int CRC();

void function\_pointers();

int main() {

function\_pointers();

if (IsDebuggerPresent())

return 0;

if (IsVM())

return 0;

\_\_asm {

push 0x75

push offset cycle

call WriteMe

add esp, 8

mov eax, 5

mov ebx, 5

cmp eax, ebx

cycle :

JE short cycle

}

unsigned long NtGlobalFlags = 0;

\_\_asm {

mov eax, fs:[30h]

mov eax, [eax + 68h]

mov NtGlobalFlags, eax

}

/\*

FLG\_HEAP\_ENABLE\_TAIL\_CHECK 0x10

FLG\_HEAP\_ENABLE\_FREE\_CHECK 0x20

FLG\_HEAP\_VALIDATE\_PARAMETERS 0x40

\*/

if (NtGlobalFlags & 0x70) {

return 0;

}

if (check\_debugged())

return 0;

HANDLE checkOllyDbg = FindWindow(TEXT("OLLYDBG"), NULL);

if (checkOllyDbg)

return 0;

HANDLE checkWinDbg = FindWindow(TEXT("WinDbgFrameClass"), NULL);

if (checkWinDbg)

return 0;

\_\_asm {

jmp res - 1

res:

\_\_emit 0xc0

dec eax

}

\_\_asm {

call $ + 5

add [esp], 5

ret

}

if (IsVM2())

return 0;

//if (!CRC())

// return 0;

if (!check\_password())

return 0;

std::string info2 = "Q W\*Xcmb~\*zkyy}exn$\*Od`es\*~bcy\*zxemxkg+";

info2 = get\_string(info2);

print\_info(info2);

if (get\_password())

main\_act();

return 0;

}

int CRC() {

int res = 0;

\_\_try {

int a = 12345, b = 0;

a /= b;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER) {

res = check\_CRC();

}

return res;

}

int check\_debugged() {

BOOL isDebugged = TRUE;

\_\_try {

\_\_asm {

pushfd

or dword ptr[esp], 0x100

popfd

nop

}

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER) {

isDebugged = FALSE;

}

if (isDebugged)

return 1;

return 0;

}

void jump\_opcode() {

\_\_asm {

xor eax, eax

jz label

\_\_emit 0xe9

label:

mov ebx, eax

}

}

void function\_pointers() {

void (\*jump\_opcode\_ptr)(void);

jump\_opcode\_ptr = jump\_opcode;

return jump\_opcode\_ptr();

}

bool IsVM2() {

wchar\_t VMwareProcessName[] = { L"vmtoolsd.exe" };

PROCESSENTRY32 pe;

HANDLE hSnapShot;

hSnapShot = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

ZeroMemory(&pe, sizeof(PROCESSENTRY32W));

pe.dwSize = sizeof(PROCESSENTRY32W);

Process32First(hSnapShot, &pe);

do {

if (memcmp(pe.szExeFile, VMwareProcessName, 24) == 0) // Мы под VMware

return true;

} while (Process32Next(hSnapShot, &pe));

return false;

}

bool IsVM()

{

int cpuInfo[4] = {};

\_\_cpuid(cpuInfo, 1);

if (!(cpuInfo[2] & (1 << 31)))

return false;

const auto queryVendorIdMagic = 0x40000000;

\_\_cpuid(cpuInfo, queryVendorIdMagic);

const int vendorIdLength = 13;

using VendorIdStr = char[vendorIdLength];

VendorIdStr hyperVendorId = {};

memcpy(hyperVendorId + 0, &cpuInfo[1], 4);

memcpy(hyperVendorId + 4, &cpuInfo[2], 4);

memcpy(hyperVendorId + 8, &cpuInfo[3], 4);

hyperVendorId[12] = '\0';

static const VendorIdStr vendors[]{

"KVMKVMKVM\0\0\0", // KVM

"Microsoft Hv", // Microsoft Hyper-V or Windows Virtual PC \*/

"VMwareVMware", // VMware

"XenVMMXenVMM", // Xen

"prl hyperv ", // Parallels

"VBoxVBoxVBox" // VirtualBox

};

for (const auto& vendor : vendors) {

if (!memcmp(vendor, hyperVendorId, vendorIdLength))

return true;

}

return false;

}

int WriteMe(void\* addr, int wb)

{

HANDLE

h = OpenProcess(PROCESS\_VM\_OPERATION |

PROCESS\_VM\_WRITE,

true, GetCurrentProcessId());

return WriteProcessMemory(h, addr, &wb, 1, NULL);

}

**Приложение Г**

Листинг «password.cpp»

#include "password.h"

#include <fstream>

#include <cassert>

#include <iostream>

using namespace std;

string gPassword = "";

void create\_serial\_number() {

string fname = "yoxckf$~r~";

fname = get\_string(fname);

ofstream output(fname);

assert(output);

string serial = "AOS.";

for (int i = 0; i < 10; i++) {

int act = rand();

if (act % 2 == 0)

serial += (char)(65 + rand() % 26) ^ CONST;

else

serial += (char)(48 + rand() % 10) ^ CONST;

}

serial += "A";

output << serial;

}

string get\_string(string str) {

for (int i = 0; i < str.length(); i++)

str[i] ^= CONST;

return str;

}

string type\_data() {

char data[31];

cin.getline(data, 30, '\n');

string str(data);

return str;

}

void print\_info(string info) {

cout << info << endl;

}

int check\_CRC() {

unsigned char \*ptr = (unsigned char\*)get\_password;

long long crc = 0;

for (int i = 0; i < 691; i++) {

crc += (long long)(\*((unsigned char\*)ptr));

ptr++;

}

//cout << crc;

if (crc == 66095)

return 1;

else return 0;

}

int check\_password() {

string password, realPassword;

string info5 = "Xcmb~\*zkyy}exn";

info5 = get\_string(info5);

while (1) {

string info1 = "Q W\*Od~ox\*k\*zkyy}exn0\*";

info1 = get\_string(info1);

print\_info(info1);

gPassword = type\_data();

string cpygPassword = "{\"password\":\"" + gPassword + "\"}";

string realgPassword = gPassword;

gPassword = cpygPassword;

if (get\_password())

print\_info(info5);

gPassword = realgPassword;

if (gPassword == "1qaz@WSX3edc$RFV")

cout << "Right password";

if (!get\_password()) {

string info3 = "Q+W\*]xedm\*zkyy}exn&\*~szo\*('odn(\*~e\*orc~\*ex\*kds\*e~box\*~e\*~xs\*kmkcd0\*";

info3 = get\_string(info3);

print\_info(info3);

string end;

end = type\_data();

string info4 = "'odn";

info4 = get\_string(info4);

if (end == info4)

return 0;

if (gPassword == "I will be back!@#123") {

print\_info(info5);

}

}

else

return 1;

}

}

//00422C90 - начало 4336784 - в десятичной\

00422F43 - конец 4337475 - в десятичной\

691 - количество байт для функции

int get\_password() {

string fname = "zkyy}exn$~r~";

fname = get\_string(fname);

ifstream input(fname);

string realPassword;

assert(input);

input >> realPassword;

realPassword = get\_string(realPassword);

if (gPassword == realPassword) {

create\_serial\_number();

return 1;

}

return 0;

}