Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Высшая школа кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА № 1

«Изучение методов защиты программного обеспечения»

1. по дисциплине «Технологии реверс-инжиниринга программного обеспечения»
2. Выполнил
3. студент гр. 5151004/10101 Веремейчук Я.Ю.

<подпись>

1. Преподаватель
2. асс. преподавателя Овасапян Т.Д.

<подпись>

1. Санкт-Петербург
2. 2023
3. Цель работы

Изучение методов защиты программного обеспечения и возможных способов их преодоления.

1. Ход работы

Перед выполнением работы было выбрано защищаемое программное обеспечение, а именно – игра «Крестики-нолики на большом поле». Код отличается множеством вызываемых функций и переменных.

Изображение выглядит как снимок экрана, шаблон, ткань, шов

Автоматически созданное описание

Рисунок 1 – Ожидаемый результат работы программы

Для начала в код была добавлена проверка пароля. Пароль, введенный пользователем, проверяется на корректность символов и на размер. Пароль был предварительно зашифрован с помощью хеш-функции и записан в файл, из которого он будет читаться и сверяться с хешем, полученным путем шифрования пароля, введенного пользователем. Так как в стандартной библиотеке Си отсутствуют хеш-функции, алгоритм хеширования sha256 был прописан явно.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Фрагмент кода проверки пароля

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 3 – Хеш пароля

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 4 – Вывод серийного номера

Далее был произведен бинарный патчинг программы. Для этого с помощью ПО IDA Pro был найден вызов функции проверки пароля. Были произведены действия, чтобы эта функция игнорировалась, и программа сразу переходила к выполнению основного кода.

Изображение выглядит как текст, снимок экрана, диаграмма, программное обеспечение

Автоматически созданное описание

Рисунок 5 – Предполагаемая схема после патчинга

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 6 – Патчинг

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 7 – Игнорирование функции проверки пароля

Далее к исходному коды применялись различные методы защиты.

Для начала происходит вызов ложной функции проверки пароля (названия функций были изменены после выполнения работы, дабы не раскрыть злоумышленнику их функционал).

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 8 – Тела функций ложной проверки пароля

Также была создана еще одна функция проверки пароля, имеющая схожий алгоритм с первой. Но верный пароль (его хеш) для выполнения программы теперь также зашит в код, дабы избежать махинаций с файлом password.txt и его содержимым.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 9 – Функция дополнительной проверки пароля

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – Проверка пароля в ходе выполнения основной программы

Также местами были добавлены ассемблерный вставки, которые ничего не делают, но могут запутать злоумышленника. Между такими вставками можно прятать функции проверки пароля.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 11 – Пример ассемблерной вставки

Затем все строки, которые выводятся в программе, были зашифрованы с помощью алгоритма Цезаря. Ключ, над значением которого заранее был произведен сдвиг, хранится в глобальной переменной. Перед дешифрованием строк над значением ключа снова производится сдвиг. Таким образом ключ хранится в несколько скрытом виде.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 12 – Функции шифрования

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 13 – Функции сдвига

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 14 – Пример зашифрованной строки

Была добавлена проверка на то, что переменные, заданные в начале программы (размер поля, первый игрок, длина выигрышной последовательности, уровень сложности игры), не изменяются. При наличии изменений программа не завершается, а выводит некорректное значение, что также может запутать злоумышленника.

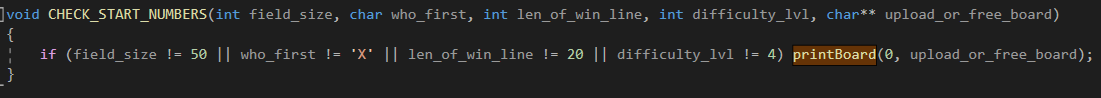


Рисунок 15 – Проверка входных значений переменных

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 16 – Пример некорректного вывода результата работы программы (при изменении входных переменных)

Далее был добавлен алгоритм нахождения контрольной суммы CRC-32Q. Контрольная сумма считается в начале программы и в конце.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 17 – Фрагмент механизма подсчета контрольной суммы

Изображение выглядит как текст, снимок экрана, Шрифт, клавиатура

Автоматически созданное описание

Рисунок 18 – Пример вывода значений CRC в начале и перед завершением программы (для отладки)

Далее были добавлены методы обнаружения отладки в количестве 6 штук.

Таблица 1 – Использованные методы антиотладки

|  |  |
| --- | --- |
| Метод обнаружения отладки | Описание метода |
| IsDebuggerPresent() | Является частью Windows API и предназначена для определения, выполняется ли программа в отладчике. Эта функция возвращает TRUE (ненулевое значение) в случае, если процесс выполняется в отладчике, и FALSE (нулевое значение) в противном случае. |
| NtGlobalFlags | Данный метод антиотладки использует чтение глобальных флагов отладки в Windows (через переменную NtGlobalFlags) и проверяет определенные биты (0x70), чтобы выявить, выполняется ли программа в отладчике. |
| CheckRemoteDebuggerPresent() | Функция CheckRemoteDebuggerPresent используется для проверки наличия отладчика в текущем процессе. Она принимает два аргумента:  GetCurrentProcess(): Эта функция получает дескриптор текущего процесса.  &isDebuggerPresent: Это указатель на переменную isDebuggerPresent, в которую будет записан результат проверки. |
| Отладочные регистры | Всего отладочных регистров восемь, DR0–DR7. Первые четыре регистра DR0–DR3 содержат информацию о точках останова, регистры DR4–DR5 — зарезервированные, регистр DR6 заполняется, когда сработал брейк-пойнт отладчика, и содержит информацию об этом событии. Регистр DR7 содержит биты управления отладкой. В данной работе были рассмотрены первые 4 регистра. |
| SeDebugPrivilege | Позволяет проверить наличие привилегии SeDebugPrivilege в токене текущего процесса и информирует пользователя о результате этой проверки. SeDebugPrivilege – это одна из привилегий в операционной системе Windows, которая предоставляет программам возможность получать доступ к процессам и потокам других пользователей. Эта привилегия позволяет, например, отлаживать или мониторить другие процессы. |
| TLS Callbacks | Метод заключается в том, что антиотладочные приемы встраиваются в TLS Callbacks, которые выполняются до входной точки программы. Внутри самого приложения могут быть установлены точки останова, да и внимание будет сконцентрировано на основном коде приложения, но этот прием завершит отладку, даже толком ее не начав. |

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 19 – Методы обнаружения отладки

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 20 – Проверка наличия привилегии SeDebugPrivilege

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 21 – Метод антиотладки TLS Callbacks

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 22 – Завершение программы при работе отладчиков WinDBG и OllyDBG

Далее в код были добавлены методы противодействия дизассемблированию. Один из методов – условные переходы на одну и ту же функцию. Дизассемблер не распознает этого как безусловный переход, так как он дизассемблирует код по одной инструкции за раз. Это может ввести в заблуждение анализатора кода. Еще один метод заключается в использовании условного перехода с постоянным условием. Еще один метод, затрудняющий дизассемблирование – цикл с пустым телом.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 23 – Условные переходы на одну и ту же функцию

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 24 – Условный переход с постоянным условием

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 25 – Пустой цикл на 10 итераций

Далее применялись методы для обнаружения виртуальных машин. Один из них – попытка вызова исключения. Некоторые виртуальные машины могут запрещать доступ к определенным портам I/O или генерировать исключения при таких попытках. Еще один метод – отправка специфические команды ввода/вывода (I/O) на порт, который обычно используется VMware.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 26 – Вызов исключения

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 27 – Обнаружение VMware

Далее в программу был добавлен самомодифицирующийся код. Структура кода изменяется в зависимости от случайного числа, что усложняет анализ дизассемблированного кода.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 28 – Самомодифицирующийся код

Был внедрен код, где одна функция вызывает другую и в конце цепочки происходит проверка CRC. Такая ветвь, которая вначале кажется лишь мертвым кодом, может скрыть проверку целостности программы.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 29 – Фрагмент запутывающего кода

Затем была проведена простейшая обфускация кода: изменены названия функций, добавлены лишние переменные и мертвый код. Также был применен метод Constant Unfolding – большое развертывание констант. Еще был применен метод Control Flow Flattening, который делает поток выполнения программы менее предсказуемым и усложняет его анализ. Например, при использовании if, не всегда используется else.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 30 – Обфускация: «не говорящие» имена функций



Рисунок 31 – Обфускация: Constant Unfolding

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 32 – Обфускация: Control Flow Flattening

Далее полученный исполняемый файл был открыт с помощью IDA Pro. На рисунках ниже можно увидеть, как изменилась функция main с момента применения методов защиты информации. Стоит заметить, что проект содержит 4 сишных файла и 3 файла-заголовков.

Изображение выглядит как диаграмма, План, Технический чертеж, линия

Автоматически созданное описание

Рисунок 33 – Функция main до модификации

Изображение выглядит как диаграмма, План, Технический чертеж, схематичный

Автоматически созданное описание

Рисунок 34 – Функция main после изменений

Далее путем патчинга файла функция main обошла несколько проверок и попала в блок кода с основным функционалом программы.

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 35 – Патчинг

Затем был установлен упаковщик UPX и исполняемый файл crackme.exe был упакован. Размер файла стал в 3,6 раза меньше.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 36 – Упаковка исполняемого файла

После этого получившийся файл был открыт с помощью IDA Pro. Структура программы изменилась, появилось множество маленьких отдельных блоков.

Изображение выглядит как текст, диаграмма, снимок экрана, План

Автоматически созданное описание

Рисунок 37 – Изменение упакованного файла

Затем с помощью утилиты PeiD были найдены точки вхождения упакованной и неупакованной программ.

Изображение выглядит как текст, Шрифт, число, снимок экрана

Автоматически созданное описание

Рисунок 38 – Энтропия неупакованного исполняемого файла

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 39 – Энтропия неупакованного исполняемого файла

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 40 – Энтропия упакованного исполняемого файла

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 41 – Энтропия упакованного исполняемого файла

1. Выводы

В ходе работы были изучены методы защиты программного обеспечения и возможные способы их преодоления.