Текст доклада на защиту ВКР

Шкалин Кирилл Павлович

3 июня 2023 года

1 Вступление

Слайд 2

В связи с тем, что в современном мире многие общественные процессы приобретают цифровую форму проблема информационной безопасности является крайне актуальной. Наиболее мощным и распространенным Инструментом взлома является программа отладчик.

Целью данной работы является разработка метода обеспечения защиты программы от взлома отладчиком. Этот метод должен как обеспечивать обнаружение факта работы программы под отладчиком, так и препятствовать самому процессу отладки.

В связи с этим были поставлены следующие **задачи**:

1. изучить алгоритм действия отладчика;
2. разработать алгоритм защиты от отладчика;
3. реализовать полученный алгоритм, обеспечив при этом достаточный уровень скрытности;
4. провести тестирование полученной системы защиты.

2 Принцип работы отладчика

Слайд 3

Рассмотрим принцип действия отладчика. Отладчик может самостоятельно запустить отлаживаемый процесс или подключиться к уже работающему процессу. Как правило, отладчик открывает процесс с доступом на чтение и запись в виртуальную память процесса. Затем отладчик в цикле обрабатывает события отладки.

Слайд 4

Отладчик предоставляет две основные возможности: поставить точку останова, дойдя до которой выполнение программы прервется, или трассировать программу, то есть выполнять ее по шагам. Чтобы поставить точку останова отладчик должен заменить один байт в коде программы на инструкцию `CC`. А чтобы продолжить выполнение программы, ему необходимо восстановить замененный байт.

Отладчик позволяет изменить исходный код программы во время ее выполнения, после чего сохранить изменения в файл на диске. Таким образом получают взломанную версию программы.

Исходя из ранее сказанного, было решено реализовать механизм защиты, который будет препятствовать изменению исходного кода программы. Такой подход затруднит использование точек останова при отладке, а также защитит исходный код программы от модификации.

3 Алгоритм защиты от отладчика

Слайд 5

Разработанный алгоритм находит контрольную сумму критической секции программы, после чего сравнивает ее с заранее известным значением. Если эти значения не совпадают, значит код программы подвергался модификации.

Метод реализованный в данной работе находит контрольную сумму всей секции кода программы. Для того, чтобы подсчитать контрольную сумму необходимо знать адреса начала и конца критического участка.

Информация о смещениях и размерах различных секций извлекается из полей PE-формата. PE-формат — формат исполняемых файлов, используемый в 32- и 64-разрядных версиях операционной системы Microsoft Windows. Формат PE представляет собой структуру данных, содержащую всю информацию, необходимую PE-загрузчику для отображения файла в память. Защищаемая программа во время выполнения будет анализировать свой собственный PE-заголовок, чтобы получить адрес начала и размер интересующей секции.

4 Проблема ASLR

Слайд 6

При реализации данного алгоритма пришлось столкнуться с проблемой вызванной ASLR. ASLR (address space load randomization) — это метод компьютерной безопасности, предназначенный для предотвращения использования уязвимостей, связанных с повреждением памяти. ASLR случайным образом упорядочивает позиции адресного пространства ключевых областей данных процесса, включая базовый адрес загрузки, позиции стека, кучи и загружаемых библиотек.

Из этого следует, что адреса конкретных функций и переменных невозможно определить до запуска программы. А значит адреса указанные в параметрах инструкций, как например `near jmp`, в момент запуска программы меняются. Следовательно, необходим механизм корректирования адресов в различных секциях программы.

Слайд 7

В системе Windows для этого в исполняемом файле, есть таблица базовых релокаций. Таблица базовых релокаций содержит записи для всех исправлений в образе программы. Защищаемая программа в процессе работы будет подсчитывать контрольную сумму с учетом того, что некоторые адреса изменяются в момент запуска программы.

5 Итоговый алгоритм

Слайд 8

Алгоритм нахождения контрольной суммы должен быть быстрым, чтобы по задержке нельзя было определить место в программе, где осуществляется проверка. Также алгоритм не должен хранить в памяти большое количество промежуточных данных, так как это тоже упрощает нахождение блока защиты.

Итоговый алгоритм состоит из главного цикла, в котором суммируется каждый байт секции кода, кроме тех случаев, когда адрес проверяемого байта есть в таблице релокаций. В этом случае этот байт и три следующих за ним игнорируются. Полученный алгоритм имеет линейную сложность, а также использует всего 16 байт в стеке процесса.

6 Реализация алгоритма

Слайд 9

Составленный алгоритм реализован на языке ассемблера. Сам ассемблерный код заключен в макрос. Преимуществом такого подхода является то, что при вызове функции-макроса происходит не вызов функции, а подстановка кода функции на место вызова. Использование макроса предоставляет еще одну полезную возможность, а именно указать при вызове разный порядок регистров процессора. Таким образом, вызывая макрос с разным порядком регистров, будут получаться разные участки кода. Если злоумышленник найдет и исправит один блок защиты, то найти остальные путем поиска повторяющихся элементов памяти у него не получится.

7 Классы для работы с секциями

Слайд 10

Помимо макроса для нахождения контрольной суммы в ходе работы были реализованы модули, обеспечивающие нахождение контрольной суммы различных секций программ, загруженных в память и расположенных на диске.

Данные модули следует использовать не для организации защиты, а для отладки. В них реализованы:

- алгоритмы нахождения контрольной суммы для любой секции программы;
- система логирования, настраиваемая при помощи директив препроцессора;
- возможность записи найденной контрольной суммы в исполняемый файл расположенный на диске.

На слайде представлена программа, которая подсчитывает контрольную сумму своей секции кода. Причем подсчитывает сумму в своей виртуальной памяти, а также в exe файле расположенном на диске. Полученные значения совпадают.

8 Тестирование

Слайд 11

Для проведения тестов была написана демонстрационная программа на языке C с использованием только WinAPI. В случае, если введенный пользователем ключ не совпадает с тем, который сгенерировала программа, выводится окно сообщения, уведомляющее пользователя о неудаче.

Слайд 12

При помощи отладчика OllyDBG был проведен взлом данной программы так, чтобы любой введенный пароль считался корректным.

Слайд 13

После этого в код программы был помещен разработанный механизм защиты. При помощи отладчика были проведены аналогичные действия для взлома программы, но программа распознала изменение в исходном коде осталась в заблокированном состоянии.

Слайд 14

Как видно на слайде, защищенная программа детектировала изменение в своем исходном коде и осталась в заблокированном состоянии.

Слайд 15

Помимо этого в ходе тестирования было подтверждено, что если установить в отладчике точку останова программа детектирует изменение исходного кода и изменит свое поведение, что сильно затруднит процесс отладки.

9 Выводы

Слайд 16

В рамках ВКР разработан метод защиты программы от изменения ее исходного кода, а также препятствующий процессу отладки. Разработан алгоритм нахождения контрольной суммы кода защищаемой секции. Составленный алгоритм реализован и протестирован. Результаты тестирования показали, что разработанный метод обеспечивает защиту на достойном уровне.

Алгоритм реализован на языке ассемблера. Ассемблерный код был помещен в макрос языка C. Написаны программные модули на языке C++ для работы с секциями PE-файла.