**Руководство по выполнению**

**Лабораторная работа № 3. Анализ безопасности приложений.**

 **Безопасность механизмов защиты ПО от копирования.**

**Цель работы:** Получить практические навыки в анализе двоичных приложений, научиться анализировать двоичную реализацию простейших алгоритмов.

**Ход работы:**

1. Проанализировать алгоритм проверки электронной почты и пояснить принцип его работы.
2. Проанализировать алгоритм проверки серийного ключа и пояснить принцип его работы.
3. Произвести верный ввод ключа для одного любого адреса электронной почты.
4. Реализовать программу для автоматической генерации секретного ключа для любого введённого адреса электронной почты.
5. Сформулировать выводы по работе и составить отчёт.

**Краткая теория:**

**1. Сбор общей информации о файле.**

Как только мы получили исполняемый файл, мы ещё толком ничего не знаем о нём, кроме того, что он может выполниться и производить какие-либо действия. Для того чтобы узнать, какие точно действия он производит и каким образом, потребуется провести довольно трудоёмкое исследование, которое начнётся с общего анализа файла.

    Первое что необходимо узнать - платформа для исполнения и разрядность файла. Данную информацию можно получить при анализе файла в шестнадцатеричном редакторе. При анализе заголовка файла, очень просто понять платформу, для которой предназначен данный двоичный файл, также в заголовочной секции можно найти признаки принадлежности к той или иной разрядности. На Рис.1 явно видно, что данный файл предназначен для Linux-подобных ОС и имеет разрядность 32-бита, однако будет запускаться и в 64-битной версии ОС.



Рисунок 1 - Шестнадцатеричный дамп заголовка исполняемого файла.

    После определения платформы и разрядности исполняемого файла, можно производить загрузку данного файла в IDA Pro для дальнейшего исследования. Про функционал дизассемблера IDA Pro было подробно описано в одной из лекций по исследованию безопасности приложений.

После загрузки исполняемого файла в IDA Pro (далее IDA) чаще всего проверяются имеющиеся строки, для анализа некоторых функций очень удобно использовать строки, на которые есть ссылки из этих функций, при отсутствии символьной информации - строки являются незаменимым помощником в распознавании важных функций и обработчиков.

Для получения полного списка строк можно воспользоваться сочетанием клавиш Shift+F12. После нажатия данных клавиш будет отображена отдельная вкладка со списком всех строк и их адресами Рис. 2.



Рисунок 2 - Список строк, используемых в файле.

После анализа полученной информации, касающейся строк, можно переходить непосредственно к восстановлению логики работы приложения.

**2. Статический анализ дизассемблированного кода.**

Существуют различные подходы к решению данного вопроса, при анализе крупных проектов не принято восстанавливать логику работы всей системы, обычно исследуются конечные части, выполняющие несколько функций. При таком анализе обратная разработка начинается с какой-либо конечной функции, действие которой вы однозначно можете определить.

Однако в случае данной лабораторной работы можно начинать обратную разработку прямо с функции main(), так как функционал приложения крайне мал и однообразен, поэтому запутаться в функциях невозможно.

Для удобного исследования кода рекомендуется применять различные типы отображения в IDA. Типы отображения кода можно менять с помощью клавиши “Space”. Режим графового отображения кода позволяет быстро ориентироваться между логическими блоками и быстрее анализировать логические условия Рис. 3.



Рисунок 3 - Блок схема функции.

При исследовании программ часто нужно обращать внимание на обработку вводимых данных, т.к. большая часть уязвимостей и ошибок в работе связана с недостаточно надежной и хорошей обработкой введённых данных.

    При анализе приложений подобных приложению в данной лабораторной работе требуется чётко понимать, когда, как и куда поступают введённые вам данные. Рассмотрим реализацию передачи электронного адреса в функцию проверки в данной лабораторной работе.



Рисунок 4 - Фрагмент кода функции main.

В приведённом фрагменте кода функции main() на Рис. 4, мы видим, что в функцию scanf() передаётся 2 параметра, один из которых - формат вводимых данных, а второй - это адрес буфера на стеке, куда и будут сохранены введённые данные. Передачи параметров в функции осуществляются через стек.

После считывания наших данных происходит коррекция стека, и на стек поступает аргумент для следующей функции, имя для которой мы не имеем, так как она нестандартная и была написана программистом. Однако при анализе аргумента можно заметить, что он является адресом на введённый буфер, хранимый на стеке, также можно заметить, что результат работы данной функции влияет на логическое выражение и разбивает ход исполнения на 2 возможных варианта, одним из которых является вариант с выводом фразы “Email error!”. Данных предположений и наблюдений будет достаточно для формирования вывода, что данная функция является функцией проверки введённого адреса электронной почты.

**3. Портирование алгоритмов с языка низкого уровня на язык программирования высокого уровня.**

После анализа алгоритмов, реализованных в исполняемом файле, часто возникает потребность реализовать их на языке программирования высокого уровня, реализация может быть абсолютно идентичная по функциональности, но иметь другие параметры или носить иной смысл. IDA предоставляет удобные упрощения для анализа алгоритмов на ассемблере, одно из таких упрощений уже было рассмотрено выше - графическое представление функции. Графы могут помочь в определении циклов и условий.



Рисунок 5 - Графическое отображение функции.

На Рис. 5 видно, что перед входом в цикл происходит обнуление 3 локальных переменных, одна из которых является счётчиком цикла. Счётчик цикла помещается в регистр **edx** в следующем блоке, далее прибавляется к адресу, помещенному в **eax**. В **eax** помещается адрес переданного в функцию аргумента, которым является адрес введённой строки. Далее в **eax** помещается очередной символ строки, и проверяется на Null-byte, так как все строки заканчиваются Null-byt’ом и это будет служить сигналом для выхода из цикла. По ветке выхода из цикла можно обнаружить два сравнения локальных переменных с единицей. При успешном прохождении данных сравнений произойдёт переход на ветку, представленную на Рис. 6.



Рисунок 6 - Ветка выхода из функции.

Данная ветка приводит к завершению работы функции с кодом 1, что означает, что проверка электронной почты прошла успешно. То есть для верного прохождения проверки необходимо, чтобы данные локальные переменные были равны 1. Рассмотрим ветку, относящуюся к телу цикла Рис. 7.



Рисунок 7 - Ветка тела цикла.

В первом блоке тела цикла можно заметить те же инструкции, что были перед входом в цикл, соответственно очередное значение строки помещается в младший байт регистра **eax** и сравнивается с константой 40h, которая является ASCII кодом знака “@”, который используется в адресах электронной почты. Если символ строки равен данной константе, то локальная переменная увеличивается на 1. То же самое происходит и в следующем блоке. Так как мы знаем, что для успешного выполнения нашей функции необходимо, чтобы обе локальные переменные были равны 1, значит, что в введенном нами адресе электронной почты должны обязательно присутствовать только по одному символу “.” и “@”. Несложно заметить, что данная функция имеет некую ошибку. Если вы обнаружите её, то опишите в чём состоит её суть.

    После довольно простого анализа функции можно получить готовый алгоритм проверки адреса, что позволит с лёгкостью реализовать его на одном из языков программирования высокого уровня, например, на Python.

Реализация данной функции на ЯП высокого уровня - Python приведена на Рис. 8.



Рисунок 8 - Портированный алгоритм проверки.

**Оформление отчёта:** При оформлении отчёта необходимо предоставить изображения и описания проделанных действий, а также исходный код программы генератора. Сформулировать выводы по проделанной работе.