МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. Ульянова (ЛЕНИНА)
Кафедра информационной безопасности

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Модели безопасности компьютерных систем»
Тема: Информационный поток по памяти

Студентки гр. 1361 _____ Галунина Е.С.
_____ Токарева У.В.
Преподаватели _____ Шкляр Е.В.
_____ Шульженко А.Д.

Санкт-Петербург
2024

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Студентки: Галунина Е.С., Токарева У.В.

Группа 1361

Тема: Информационный поток по памяти

Задачи:

1. Написать программу, реализующую следующий функционал:

1.1. Ввод и сохранение строки текста в файл в приватную папку (создание ценного объекта);

1.2. Копирование по запросу пользователя данных из файла приватной папки в файл общедоступной папки.

2. Написать программу, реализующую следующий функционал:

2.1. Определение факта появления в общедоступной папке нового файла с информацией;

2.2. Чтение данных из файла в буфер обмена;

2.3. Сохранение считанных данных в свою папку (объект доступный нарушителю).

Дата сдачи отчета: 28.02.2024

Дата защиты отчета: 28.02.2024

Студентки	_____	Галунина Е.С.
	_____	Токарева У.В.
Преподаватели	_____	Шкляр Е.В.
	_____	Шульженко А.Д.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Сущность в произвольный момент времени может быть однозначно представлена словом некоторого языка (набором данных), которое может рассматриваться как состояние сущности.

Субъект – сущность, инициирующая выполнение операций над сущностями.

Объект – сущность, содержащая или получающая информацию, и над которой субъекты выполняют операции.

Для выполнения операций над сущностями субъекты осуществляют доступы к ним.

Основные виды доступов:

1) read – на чтение из сущности;
2) write – на запись в сущность;
3) append – на запись в конец слова, описывающего состояние сущности;
4) execute – на активацию субъекта из сущности.

Информационный поток по памяти – информационный поток, при реализации которого фактор времени не является существенным.

Дискреционная политика управления доступом – политика, соответствующая следующим требованиям:

- Все сущности идентифицированы;
- Задана матрица доступов, каждая строка которой соответствует субъекту, а столбец – сущности КС, ячейка содержит список прав доступа субъекта к сущности;
- Субъект обладает правом доступа к сущности КС тогда и только тогда, когда в соответствующей ячейке матрицы доступов содержится данное право доступа.

Мандатная политика управления доступом – политика, соответствующая следующим требованиям:

- Все сущности идентифицированы;
- Задана решетка уровней конфиденциальности информации;

- Каждой сущности присвоен уровень конфиденциальности, задающий установленные ограничения на доступ к данной сущности;
- Каждому субъекту присвоен уровень доступа, задающий уровень полномочий данного субъекта в КС;
- Субъект обладает правом доступа к сущности КС тогда, когда уровень доступа субъекта позволяет предоставить ему данный доступ к сущности с заданным уровнем конфиденциальности, и реализация доступа не приведет к возникновению информационных потоков от сущностей с высоким уровнем конфиденциальности к сущностям с низким уровнем.

Политика безопасности информационных потоков основана на разделении всех возможных информационных потоков между сущностями КС на два непересекающихся множества: множество разрешенных ИП и множество запрещенных ИП.

ХОД РАБОТЫ

1. В первую очередь был реализован интерфейс пользователя, содержащий три раздела. Первый раздел предназначен для создания нового файла в папке пользователя посредством описания заголовка и содержимого файла. Второй раздел предназначен для выбора файла из папки пользователя и копирования его в общую папку. Третий раздел предназначен для выбора файла из общей папки, записи нового содержимого этого файла и сохранения изменений. На рисунке 1 представлено описание интерфейса пользователя.

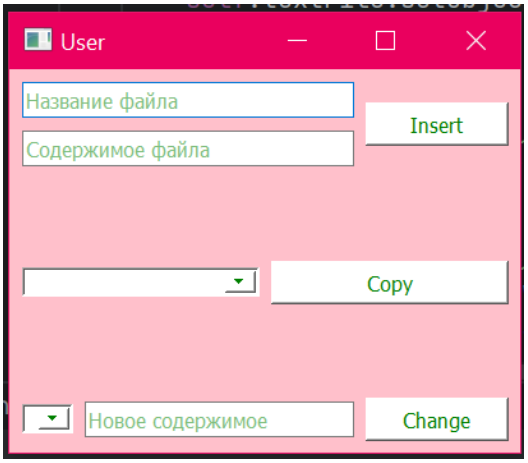

Рисунок 1 – Описание интерфейса пользователя

2. Далее была написана программа пользователя. Данная программа предназначена для создания новых файлов внутри пользовательской папки, копирования файлов из пользовательской папки в общую папку и изменения содержимого файлов пользовательской папки. На рисунке 2 представлен запуск программы пользователя.

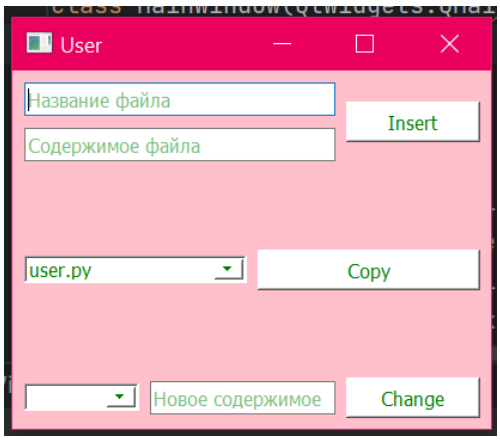

Рисунок 2 – Запуск программы пользователя

2.1. Реализуем создание файла «One» с содержимым «111». Процесс реализации создания представлен на рисунке 3.

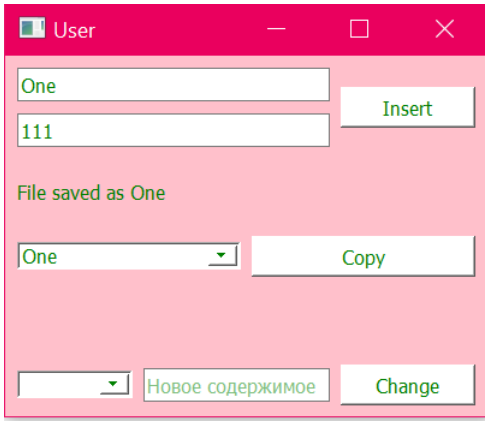

Рисунок 3 – Процесс реализации создания файла

2.2. На рисунке 4 представлены изменения внутри папки пользователя.

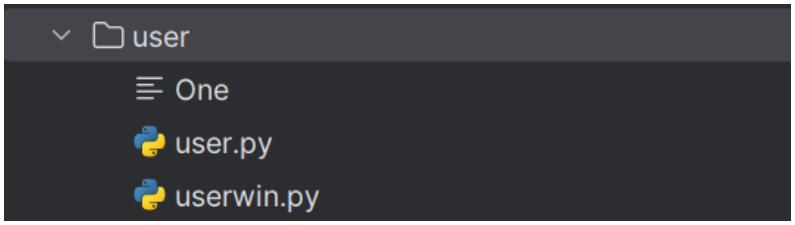

Рисунок 4 – Изменения внутри папки пользователя

2.3. Реализуем создание файла «Two» с содержимым «222» и копирование файлов «One» и «Two» в общую папку. На рисунке 5 представлен процесс копирования файла.

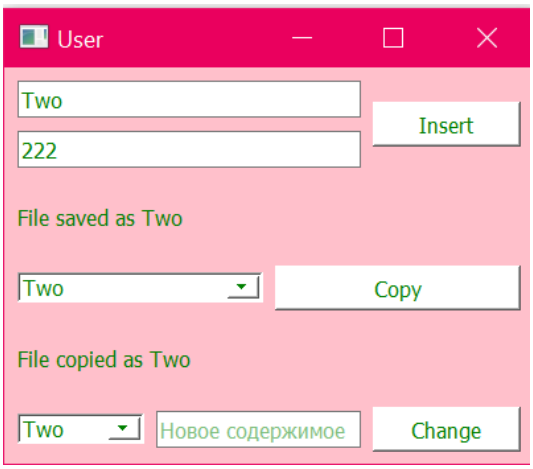

Рисунок 5 – Процесс копирования файла

2.4. На рисунке 6 представлены изменения внутри папок.

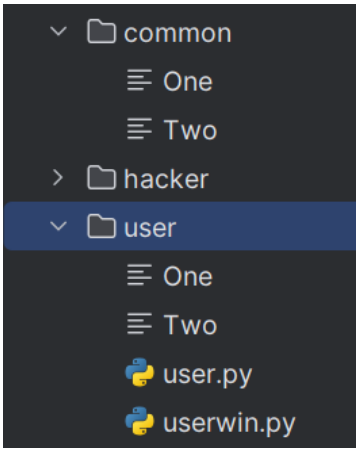

Рисунок 6 – Изменения внутри папок

2.5. Реализуем изменение содержимого файла «One» в общей папке. На рисунке 7 представлено изменение содержимого файла.

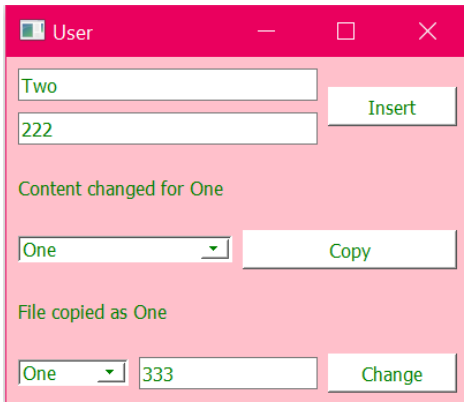

Рисунок 7 – Изменение содержимого файла

2.6. На рисунке 8 представлено новое содержимое файла «One».

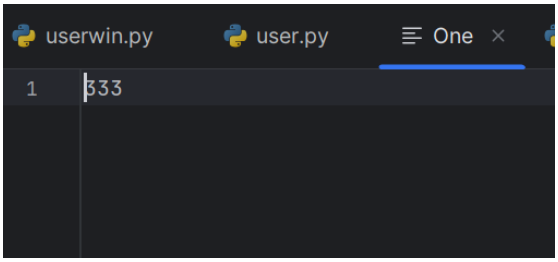

Рисунок 8 – Новое содержимое файла «One»

3. Далее описываем интерфейс для программы злоумышленника. Интерфейс должен содержать кнопку «Старт» для начала процесса отслеживания и кнопку «Стоп» для окончания процесса отслеживания. На рисунке 9 представлен интерфейс злоумышленника.

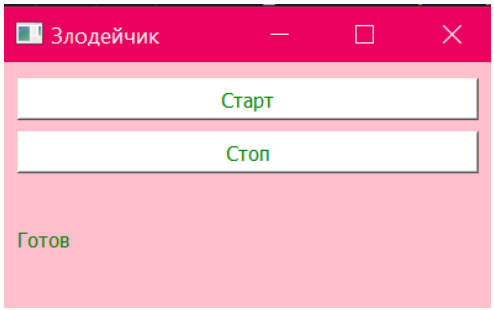

Рисунок 9 – Интерфейс злоумышленника

4. Далее реализуем программу злоумышленника. Программа злоумышленника отслеживает любые изменения в общей папке. Программа злоумышленника проверяет, есть ли файлы, находящиеся в общей папке в папке злоумышленника и если нет, то копирует их к себе. Так же программа злоумышленника в режиме реального времени проверяет добавление новых файлов в общую папку и изменение содержимого файлов в общей папке и копирует файлы в свою папку.

5. Рассмотрим работу программы злоумышленника.

5.1. На рисунке 10 представлен запуск отслеживания.

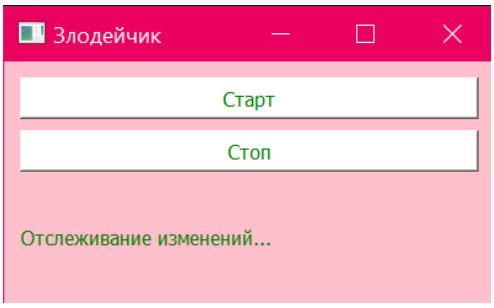

Рисунок 10 – Запуск отслеживания

5.2. На рисунке 11 представлено изменение содержимого файла «Two».

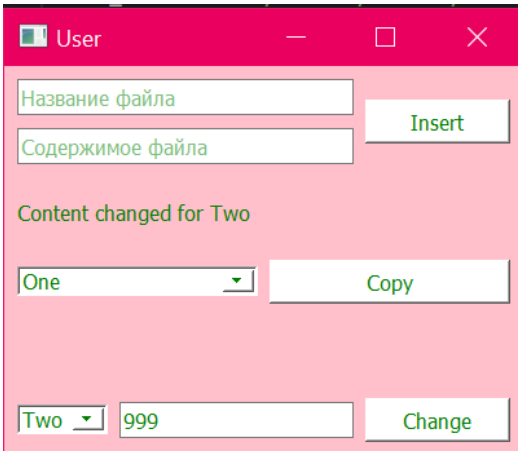

Рисунок 11 – Изменение содержимого файла «Two»

5.3. На рисунке 12 представлено создание и копирование файла «Three».

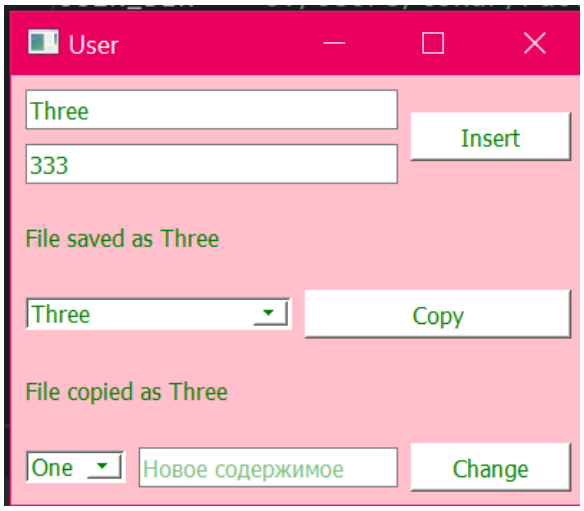

Рисунок 12 – Создание и копирование файла «Three»

5.4. На рисунке 13 представлено окончание отслеживания.

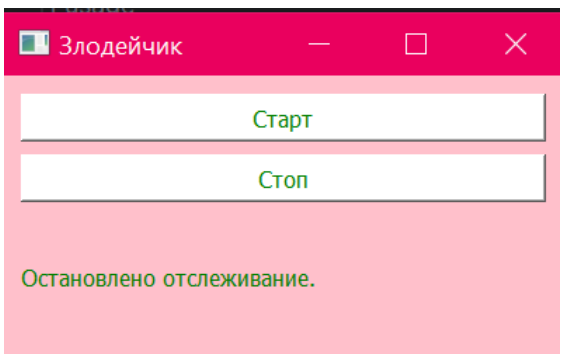

Рисунок 13 – Окончание отслеживания

5.5. На рисунке 14 представлен состав папок.

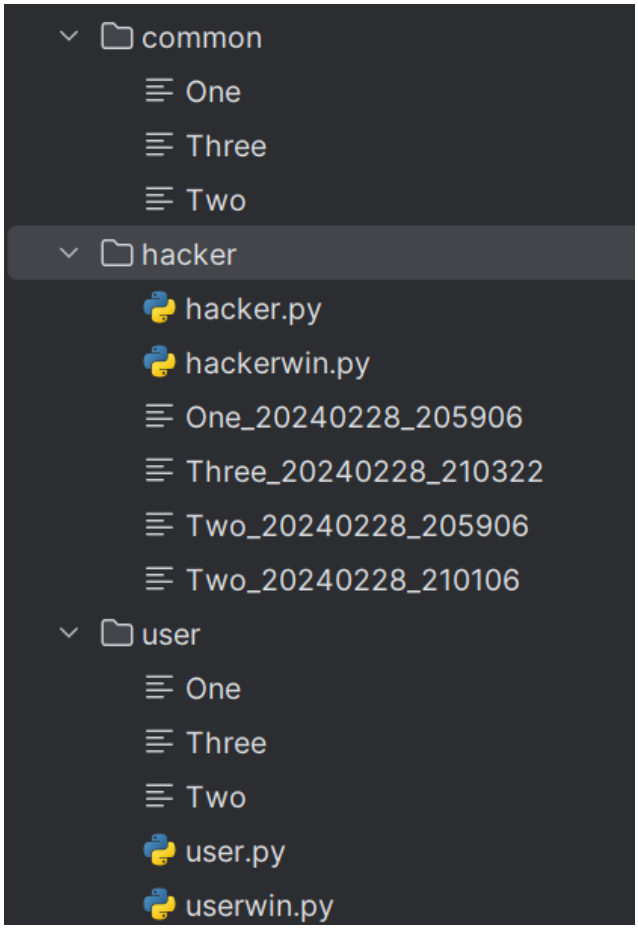

Рисунок 14 – Состав папок

ВЫВОД

В результате выполнения данной лабораторной работы было реализовано два различных программных средства:

1. Программное средство, позволяющее уязвимость путем создания и копирования содержимого файла в директорию, позволяющую субъекту, не обладающему доступом на чтение, прочитать его содержимое и использовать в угодных себе целях.
2. Программное средство, эксплуатирующее уязвимость по копированию содержимого приватного файла, содержащего в себе секретную информацию.

Обе программы показывают важность распределения доступа к файлам для различных групп субъектов и ограничения доступа информационного потока по памяти.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
# Импортируем необходимые модули из PyQt5 для работы с графическим интерфейсом
from PyQt5 import QtCore, QtGui, QtWidgets
import os # Модуль для работы с файловой системой
from datetime import datetime # Модуль для работы с датами и временем
from userwin import U_MainWindow # Импортируем класс U_MainWindow из файла userwin

# Определяем пути к директориям пользователя и общей директории
USER_DIR = "C:/Users/tokar/Рабочий стол/MBKS1/user"
COMMON_DIR = "C:/Users/tokar/Рабочий стол/MBKS1/common"

# Определяем главное окно приложения
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__() # Вызываем конструктор родительского класса
        self.u = U_MainWindow() # Создаем экземпляр класса U_MainWindow
        self.u.setupU(self) # Инициализируем пользовательский интерфейс
        self.populateUserFileList() # Заполняем список файлов из директории пользователя
        self.populateCommonFileList() # Заполняем список файлов из общей директории
        # Связываем события нажатия кнопок с соответствующими методами
        self.u.insertButton.clicked.connect(self.generateFile)
        self.u.copyButton.clicked.connect(self.transferFile)
        self.u.changeButton.clicked.connect(self.changeFileContent)

    # Метод для заполнения списка файлов из директории пользователя
    def populateUserFileList(self):
        self.u.dirFileList.clear() # Очищаем список
        files = [f for f in os.listdir(USER_DIR) if os.path.isfile(os.path.join(USER_DIR, f))] # Получаем список файлов
        self.u.dirFileList.addItems(files) # Добавляем файлы в список

    # Метод для заполнения списка файлов из общей директории
```

```
    def populateCommonFileList(self):
        self.u.allFileList.clear() # Очищаем список
        files = [f for f in os.listdir(COMMON_DIR) if
os.path.isfile(os.path.join(COMMON_DIR, f))] # Получаем список файлов
        self.u.allFileList.addItems(files) # Добавляем файлы в список

    # Метод для копирования файла из директории пользователя в общую
директорию
    def transferFile(self):
        selectedFile = self.u.dirFileList.currentText() # Получаем
выбранный файл
        sourcePath = os.path.join(USER_DIR, selectedFile) # Полный
путь к исходному файлу
        destPath = os.path.join(COMMON_DIR, selectedFile) # Полный
путь к целевому файлу

        if not os.path.isfile(destPath): # Если файл еще не существует
в общей директории
            with open(sourcePath, "rb") as src, open(destPath, "wb")
as dst: # Открываем файлы для чтения и записи
                dst.write(src.read()) # Копируем содержимое
                self.u.copyLabel.setText(f"File copied as
{os.path.basename(destPath)}") # Обновляем метку статуса
        else: # Если файл уже существует, добавляем к имени временную
метку
            timestamp = datetime.now().strftime("_%M%S") # Создаем
временную метку
            newPath = destPath + timestamp # Создаем новый путь с
временной меткой
            with open(sourcePath, "rb") as src, open(newPath, "wb") as
dst: # Открываем файлы для чтения и записи
                dst.write(src.read()) # Копируем содержимое
                self.u.copyLabel.setText(f"File copied as
{os.path.basename(newPath)}") # Обновляем метку статуса
        self.populateCommonFileList() # Обновляем список файлов в
общей директории

    # Метод для создания нового файла в директории пользователя
    def generateFile(self):
```

```
        filename = self.u.filenameLabel.text() # Получаем имя файла из
метки
        fullPath = os.path.join(USER_DIR, filename) # Полный путь к
файлу

        if not os.path.isfile(fullPath): # Если файл еще не существует
            with open(fullPath, "w") as file: # Открываем файл для
записи
                file.write(self.u.textToFile.text()) # Записываем
текст из текстового поля
                self.u.createLabel.setText(f"File saved as
{os.path.basename(fullPath)}") # Обновляем метку статуса
        else: # Если файл уже существует, добавляем к имени временную
метку
            timestamp = datetime.now().strftime("_%M%S") # Создаем
временную метку
            newPath = fullPath + timestamp # Создаем новый путь с
временной меткой
            with open(newPath, "w") as file: # Открываем файл для
записи
                file.write(self.u.textToFile.text()) # Записываем
текст из текстового поля
                self.u.createLabel.setText(f"File saved as
{os.path.basename(newPath)}") # Обновляем метку статуса
        self.populateUserFileList() # Обновляем список файлов в
директории пользователя

    # Метод для изменения содержимого файла в общей директории
    def changeFileContent(self):
        selectedFile = self.u.allFileList.currentText() # Получаем
выбранный файл
        fullPath = os.path.join(COMMON_DIR, selectedFile) # Полный
путь к файлу
        newContent = self.u.textFile.text() # Получаем новое
содержимое из текстового поля

        if os.path.isfile(fullPath): # Если файл существует
            with open(fullPath, "w") as file: # Открываем файл для
записи
```

```
                file.write(newContent) # Записываем новое содержимое
                self.u.createLabel.setText(f"Content changed for
{os.path.basename(fullPath)}") # Обновляем метку статуса
        else: # Если файл не существует
            self.u.createLabel.setText(f"File
{os.path.basename(fullPath)} does not exist.") # Обновляем метку
статуса

# Главная функция программы
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv) # Создаем экземпляр
приложения
    MainWindow = MainWindow() # Создаем экземпляр главного окна
    MainWindow.show() # Показываем главное окно
    sys.exit(app.exec()) # Запускаем основной цикл приложения
```