**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«ЛОКАЛЬНАЯ АУТЕНТИФИКАЦИЯ SALTING PASSWORDS»**

Пояснительная записка

|  |  |  |
| --- | --- | --- |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | к.т.н. доцент кафедры ИЗИ Д.В. Мишин |
| Исполнитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | студент гр. ИСБ-119 Д. А. Журавлев |
|  |  |  |

**Владимир 2023**

**ЛР №1 - локальная аутентификация salting passwords**

**Задание:**

Программа должна моделировать процессы регистрации и идентификации/аутентификации пользователя с salting passwords.

Программа должна демонстрировать значения UserID, пароль, хеш-значение пароля с солью и без, соль и т.д. на различных этапах регистрации и идентификации/аутентификации.

**Краткие сведения:**

Соль (также модификатор входа хэш-функции) — строка данных, которая передаётся хеш-функции вместе с входным массивом данных (прообразом) для вычисления хэша (образа).

Используется для усложнения определения прообраза хэш-функции методом перебора по словарю возможных входных значений (прообразов), включая атаки с использованием радужных таблиц. Позволяет скрыть факт использования одинаковых прообразов при использовании для них разной соли. Различают статическую соль (одна и та же для всех входных значений) и динамическую (генерируется для каждого входного значения персонально).

**Описание алгоритма:**

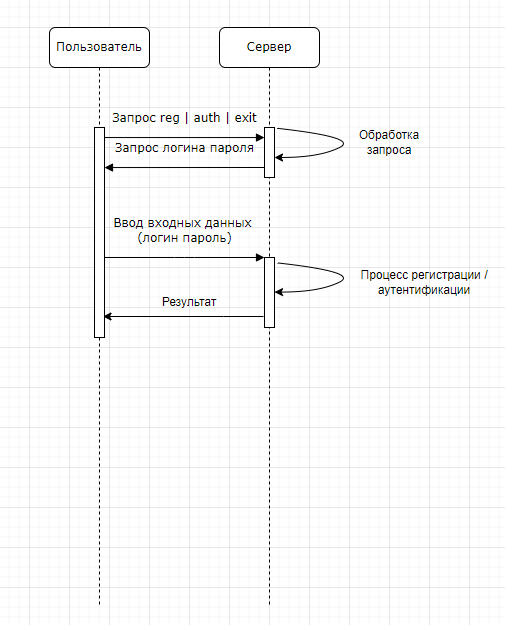
**Регистрация:**

1. Ввод пользователем входных данных для регистрации (логин и пароль).
2. Проверка корректности введённых данных.
3. Генерация персональной соли для пользователя. Генерируется 32 байтная соль.
4. Ключ = Хеширования пароля + соль.
5. Запись логин, пароль, ключ, соль в json файл.

**Аутентификация:**

1. Ввод пользователем входных данных для аутентификации (логин и пароль).
2. Проверка корректности введённых данных.
3. Ключ’ = Хеширования пароля + соль.
4. Проверка Ключа’ == Ключ пользователя
5. Предоставления доступа или отказ.

**Алгоритм работы:**



**Программа:**

Для выполнения поставленного задания была написана программа lab1\_ZIS. На вход данной программе поступают: вид операции от пользователя: регистрация (reg), аутентификация (auth), информация о пользователе (info), выход (exit); а так же логин и пароль при регистрации и аутентификации.

Запись о пользователе хранится в json файле (рисунок 2). В нём поле login – логин пользователя, password – пароль без хеширования, key – результат работы функции хеширования sha1 над паролем и персональной солью, salt – персональная соль. Исходный код программы находится в приложении 1.

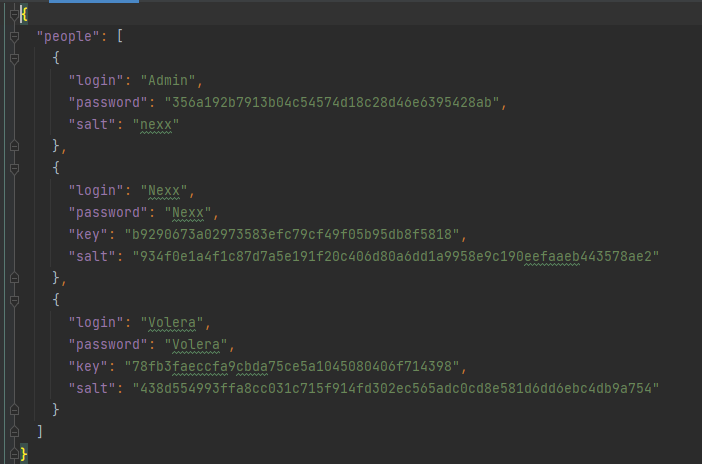
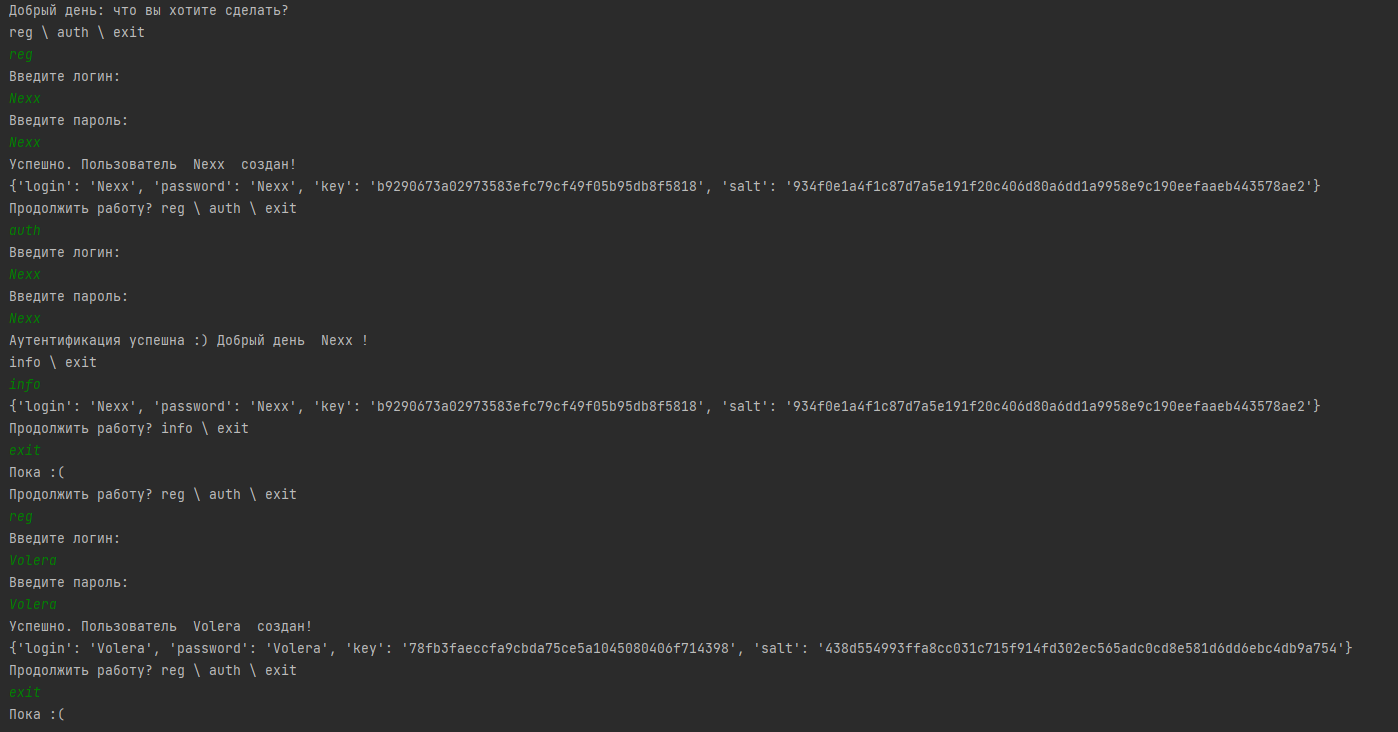


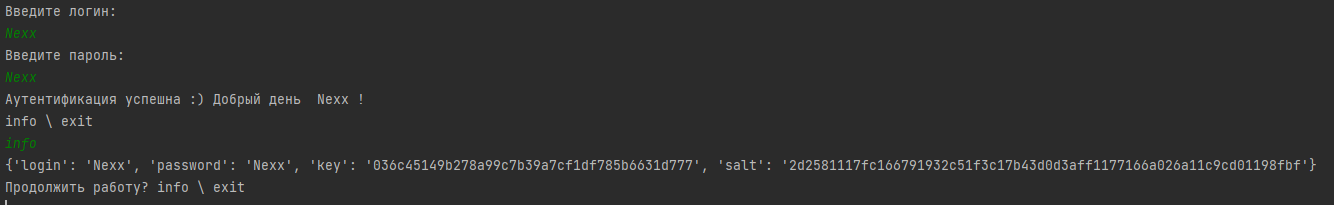
Рисунок 2 – json файл с клиентами

**Пример работы программы:**

****

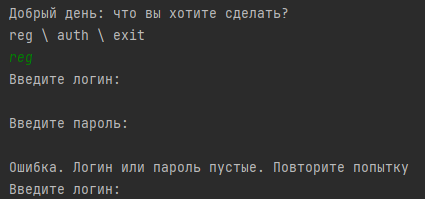
В данном примере был создан пользователь Nexx с паролем Nexx. Далее аутентификация пользователя Nexx. Далее создания пользователя Volera.

Для отображения данных о пользователе существует функция info, в ней наглядно отображается логин, пароль, ключ и соль пользователя:

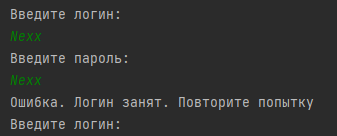


В программе имеется обработка ошибок:

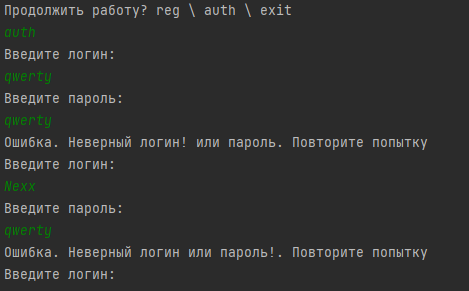
* неверный ввод регистрационных данных



* регистрация существующего логина



* неверный ввод логина и пароля (символ ! появляется в месте не верного ввода)



**Вывод:**

В процессе работы была реализована программа, которая имитируется работу протокола с salting password. Программа выполняет поставленную задачу. Так же в работе требуется дать ответ на вопрос:

*Пояснить, как salting passwords защищает от атак полного перебора (брутфорс) или атак с помощью предварительно построенных радужных таблиц?*

Представим простую аутентификацию. От пользователя к нам приходит связка значений логин/пароль, мы получаем хеш пароля и сравниваем данную связку с данными, хранящимися в базе.

key = Fmd5(password);

В данном случае, если у пользователя пароль **qwerty**, мы получим следующий хеш: **d8578edf8458ce06fbc5bb76a58c5ca4**. Если злоумышленник получит доступ к нашей базе, для подбора паролей он может воспользоваться уже готовыми сервисами, в которых уже есть значения, дающие данный хеш, либо сбрутить самому.  
Для защиты от уже готовых таблиц хешей с значениями, можно использовать статическую соль:

key = Fmd5(password . "MyUniqueSault");

Сейчас при том же пароле **qwerty** мы получим совершенно другой хеш **bdadb0330124cda0e8499c9cd118f7bd**. Готовые таблицы уже не помогут злоумышленнику, ему придется использовать брутфорс. Вот здесь и кроется минус статической соли: злоумышленник сможет сгенерировать свою таблицу хешей со статической солью и получить значения большинства паролей из базы. Для устранения этого минуса используется уникальная соль к каждому хешу:

salt = Fgen

password = Fmd5(password, salt);

Т.е. теперь помимо логина/хеша пароля в базе необходимо будет хранить значение сгенерированной соли для каждого пользователя. Разберем пример: у нас два пользователя: user1 и user2. Оба используют пароль **qwerty**. Но у первого была сгенерирована соль **zxcv** а у второго **asdf**. В итоге у пользователей при одинаковом пароле будут различные хеши: **1d8f3272b013387bbebcbedb4758586d** и **a192862aa3bf46dffb57b12bdcc4c199**.

Теперь нельзя будет сгененерировать одну таблицу хешей, для нахождения значения хеша с динамической солью придется генерировать заново. Все это направлено на увеличение времени подбора значений в случае утечки базы.

Приложение 1 – Исходный код программы

import os

import json

import hashlib

def reg(users):

i = 0

while(True):

print('Введите логин:')

new\_name = input()

print('Введите пароль:')

new\_pass = input()

if i == 3:

print('Ошибка. Неверный ввод 3 раза.')

return users

i+=1

if new\_pass != '' and new\_name != '':

if not any(u['login'] == new\_name for u in users):

break

else:

print('Ошибка. Логин занят. Повторите попытку')

else:

print('Ошибка. Логин или пароль пустые. Повторите попытку')

if new\_pass == '' or new\_name == '':

return users

salt = os.urandom(32).hex() # Новая соль для данного пользователя

key = hashlib.sha1((new\_pass+salt).encode('utf-8')).hexdigest()

print('Успешно. Пользователь ', new\_name, ' создан!')

newUser = {'login': new\_name, 'password': new\_pass, 'key':key, 'salt':salt}

print(newUser)

users.append(newUser)

return users

def auth(users):

i = 0

while (True):

print('Введите логин:')

new\_name = input()

print('Введите пароль:')

new\_pass = input()

if i == 3:

print('Ошибка. Неверный ввод 3 раза.')

return

i += 1

if new\_pass != '' and new\_name != '':

if not any(u['login'] == new\_name for u in users):

print('Ошибка. Неверный логин! или пароль. Повторите попытку')

continue

for u in users:

if u['login'] == new\_name:

salt = u['salt']

if (u['key'] == hashlib.sha1((new\_pass+salt).encode('utf-8')).hexdigest()):

print('Аутентификация успешна :) Добрый день ', u['login'], '!')

print('info \\ exit')

while (True):

k = input()

if k == 'info':

print(u)

print('Продолжить работу? info \\ exit')

elif k == 'exit':

print('Пока :( ')

return

else:

print('Неверная команда: info \\ exit')

else:

print('Ошибка. Неверный логин или пароль!. Повторите попытку')

else:

print('Ошибка. Логин или пароль пустые. Повторите попытку')

file\_path = 'bd.json'

if not os.path.exists(file\_path):

with open(file\_path, "w+") as f:

data = {'people': [{'login': 'Admin', 'password': hashlib.sha1('1'.encode('utf-8')).hexdigest(), 'salt':'nexx'}]}

json.dump(data, f, indent=2)

with open(file\_path) as f:

file\_content = f.read()

items = json.loads(file\_content)

print(items)

print('Добрый день: что вы хотите сделать?')

print('reg \\ auth \\ exit')

users = items['people']

while(True):

k = input()

if k == 'reg':

users = reg(users)

with open(file\_path, "w") as f:

json.dump({'people': users}, f, indent=2)

print('Продолжить работу? reg \\ auth \\ exit')

elif k == 'auth':

auth(users)

# with open(file\_path, "w") as f:

# json.dump({'people': users}, f, indent=2)

print('Продолжить работу? reg \\ auth \\ exit')

elif k == 'exit':

print('Пока :( ')

break

else:

print('Неверная команда: reg \\ auth \\ exit')