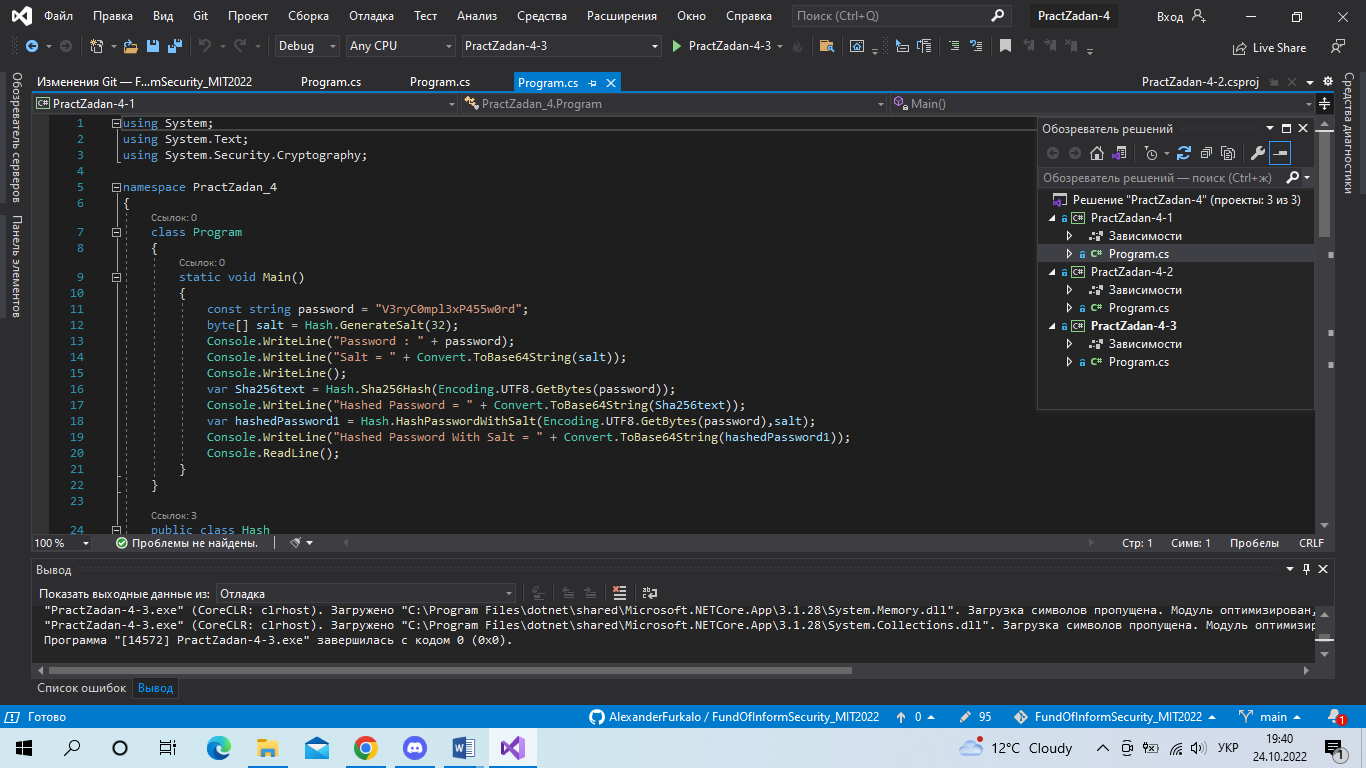
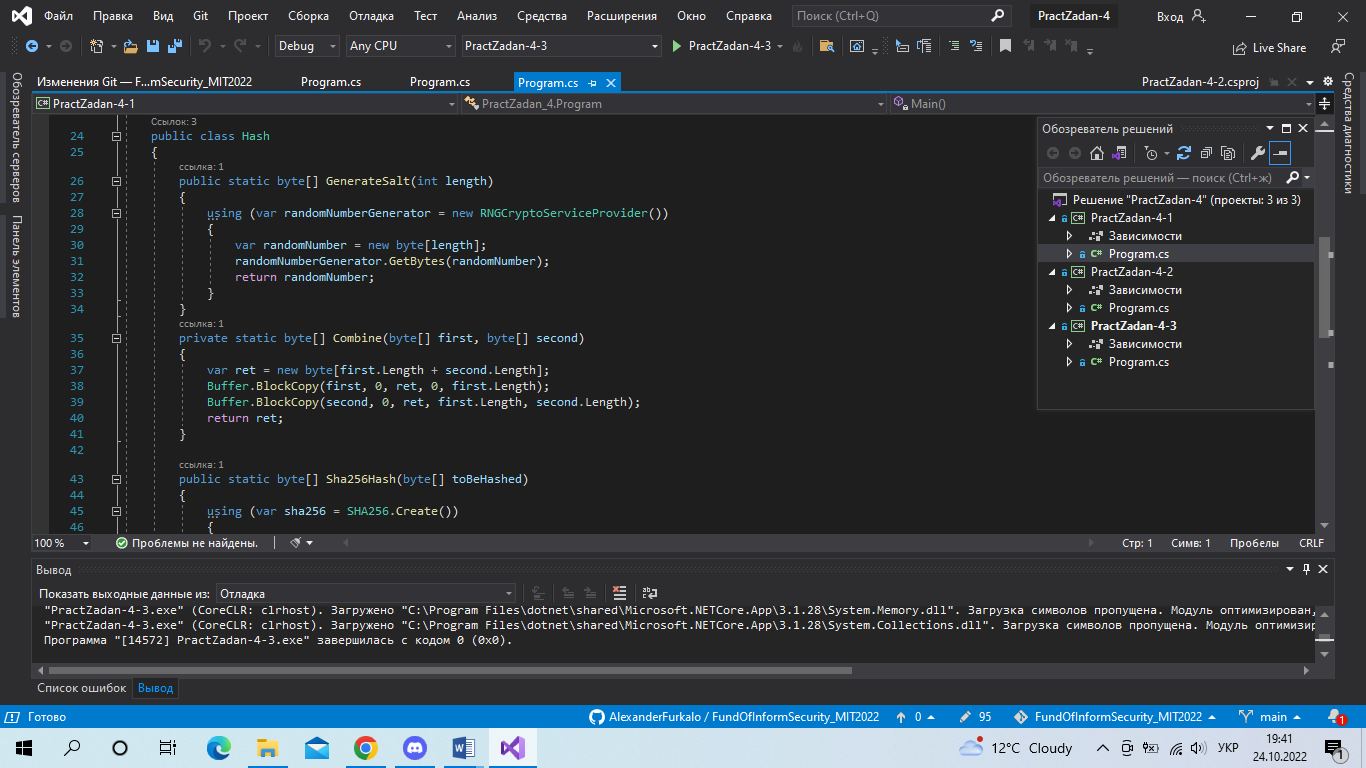
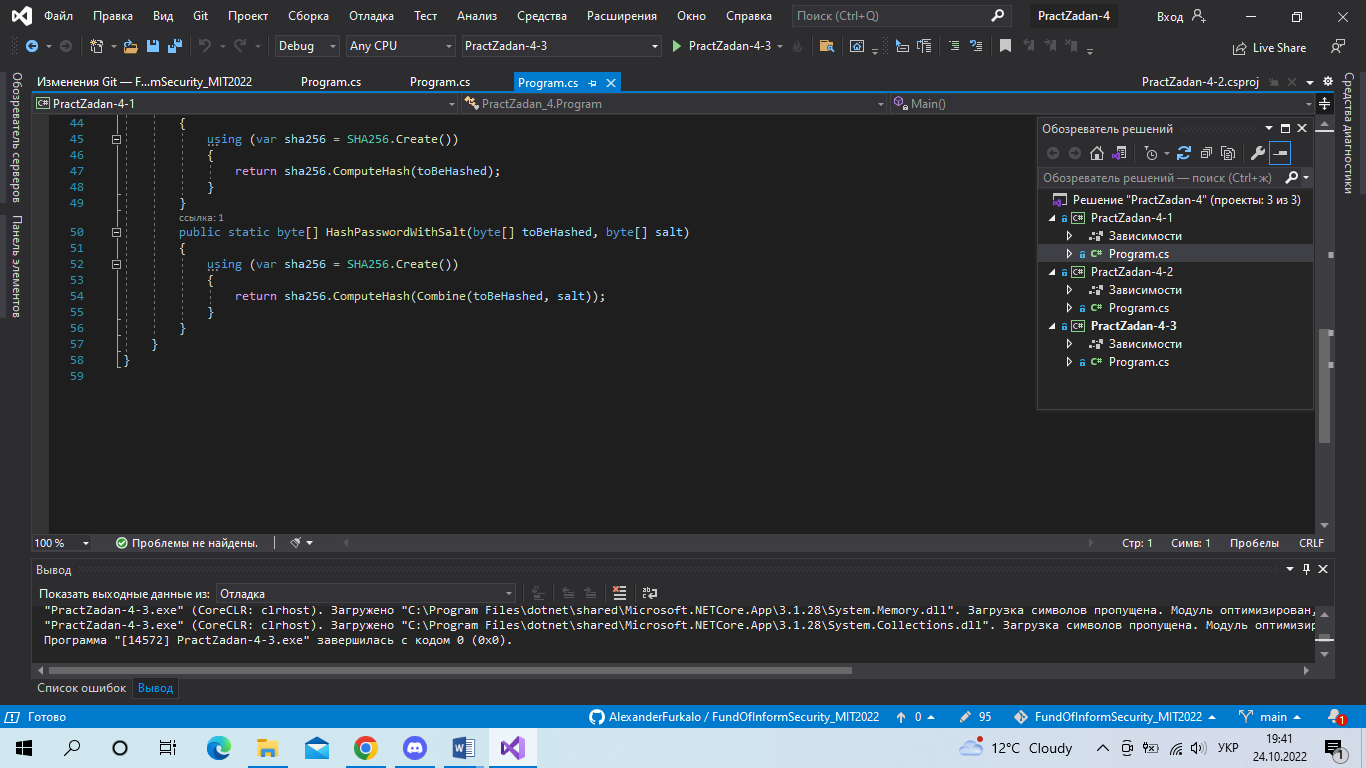
ПРАКТИЧНА РОБОТА №4  
З ОСНОВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ  
СТУДЕНТА КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКО  
ФАКУЛЬТЕТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРИ МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ  
ДРУГОГО КУРСУ, ДРУГОЇ ПІДГРУПИ  
ОЛЕКСАНДРА ОЛЕКСАНДРОВИЧА ФУРКАЛА  
ЗА ТЕМОЮ “БЕЗПЕЧНЕ ЗБЕРІГАННЯ ПАРОЛІВ”  
ЗВІТ  
24.10.2022

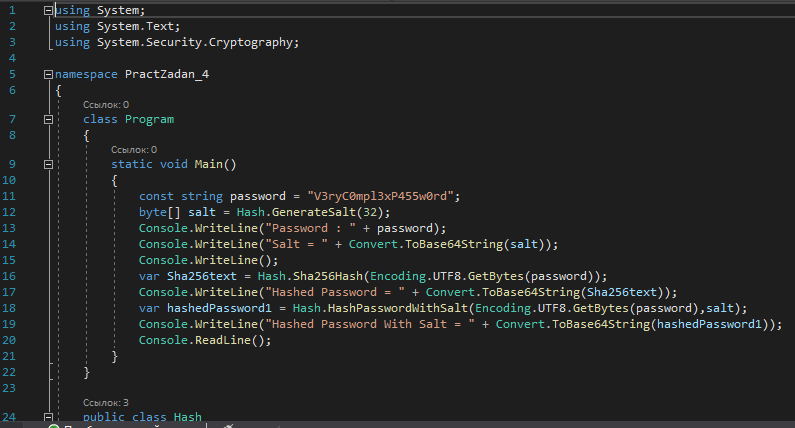
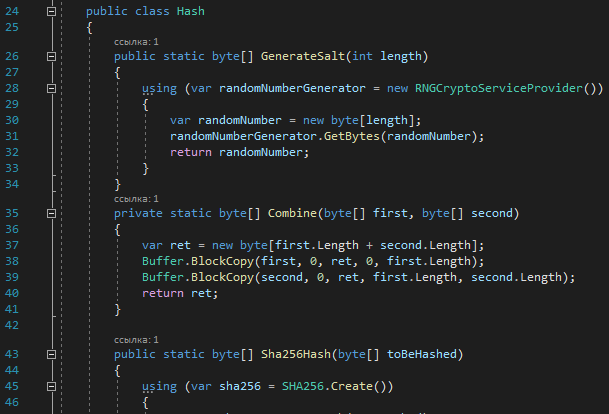
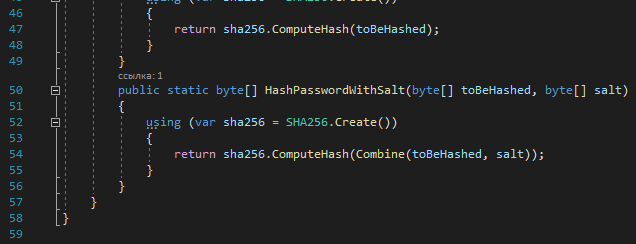
Завдання цієї практичної роботи були такі:  
1. Опрацювати розділ 5 літературного джерела Haunts Stephen. Applied Cryptography in .NET and Azure Key Vault.  
2. Для кожного способу зберігання паролів, описаного у розділі 5 (Safely Storing Passwords), законспектувати основні переваги і недоліки. Зробити висновок про доцільність застосування даного способу зберігання паролів.  
3. Розробити клас SaltedHash, що реалізує хешування паролів із додаванням додаткової ентропії. Продемонструвати роботу класу, обчислюючи хеш для заданого пароля та "солі".  
4. Розробити клас PBKDF2, що має наступну функціональність: генерує "сіль", задає алгоритм хешування (MD5, SHA1, SHA256, SHA384, SHA512) та обчислює хеш для заданого числа ітерацій. Створити програму, що обчислює час, витрачений на обчислення хешу для різного числа ітерацій (10 значень із кроком 50'000; перше значення = номер варіанта \* 10'000 ). Побудувати графік залежності витраченого часу від числа ітерацій.  
5. Написати програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та "сіль" у пам'яті. Реалізувати можливість автентифікації за логіном і паролем. Число ітерацій = номер варіанта \* 10'000.  
6. Оформити звіт.

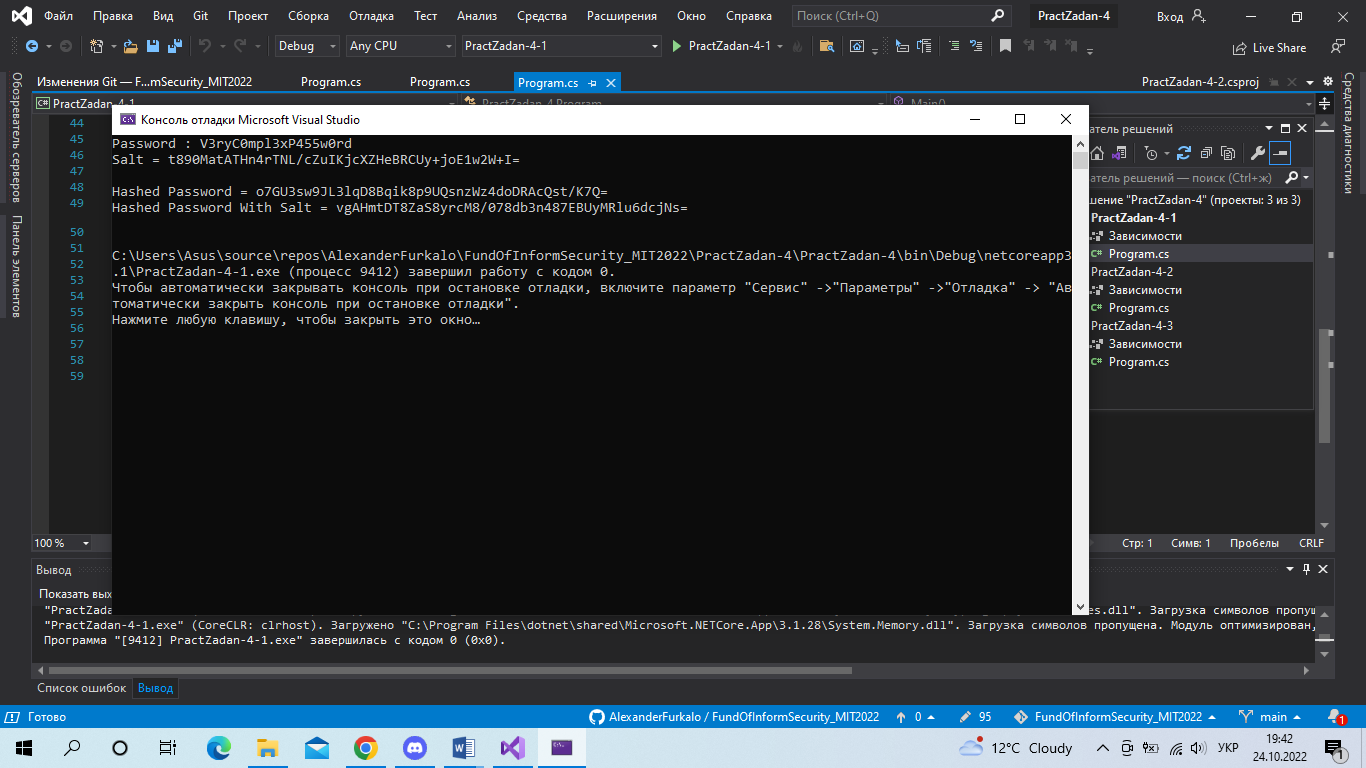
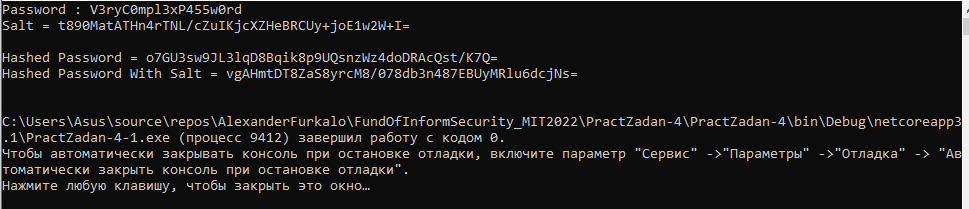
Номер варіанта: 23.

Усього є декілька способів зберігання паролів:

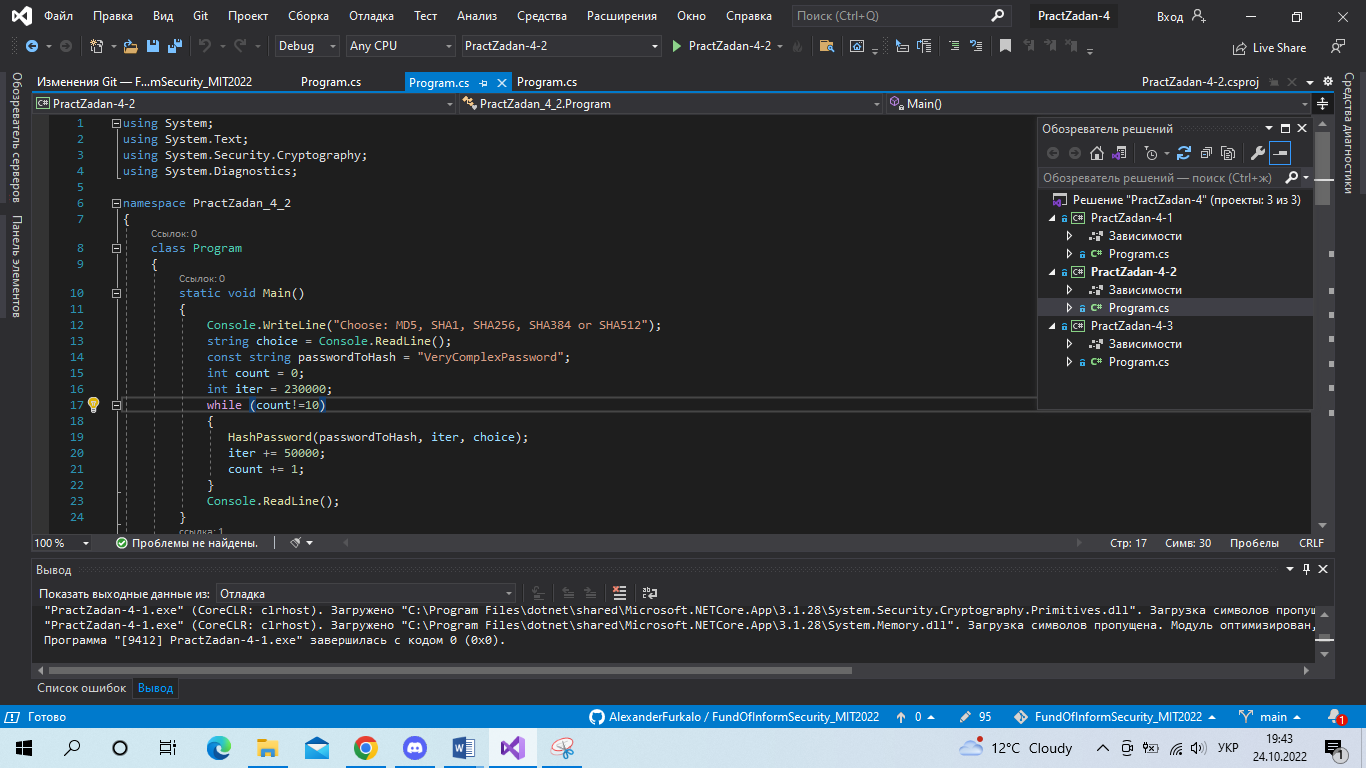
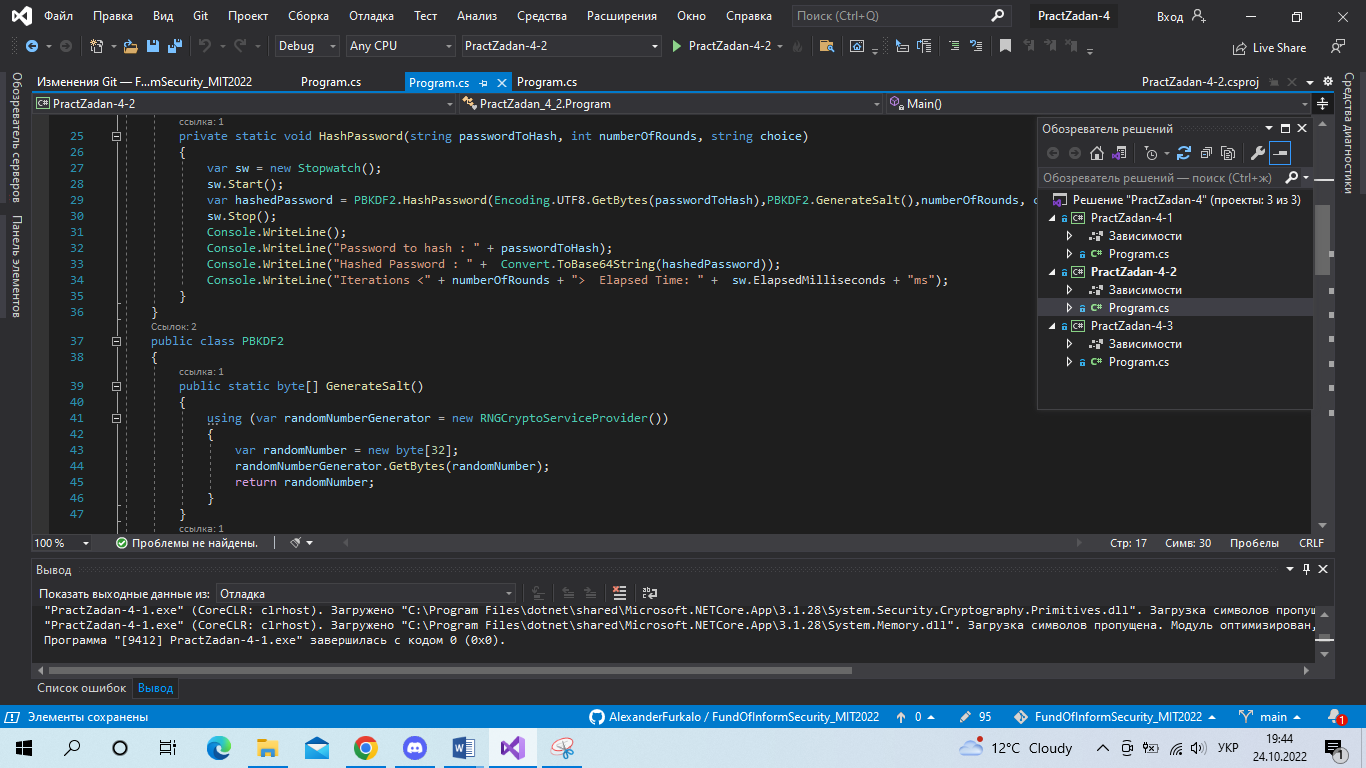
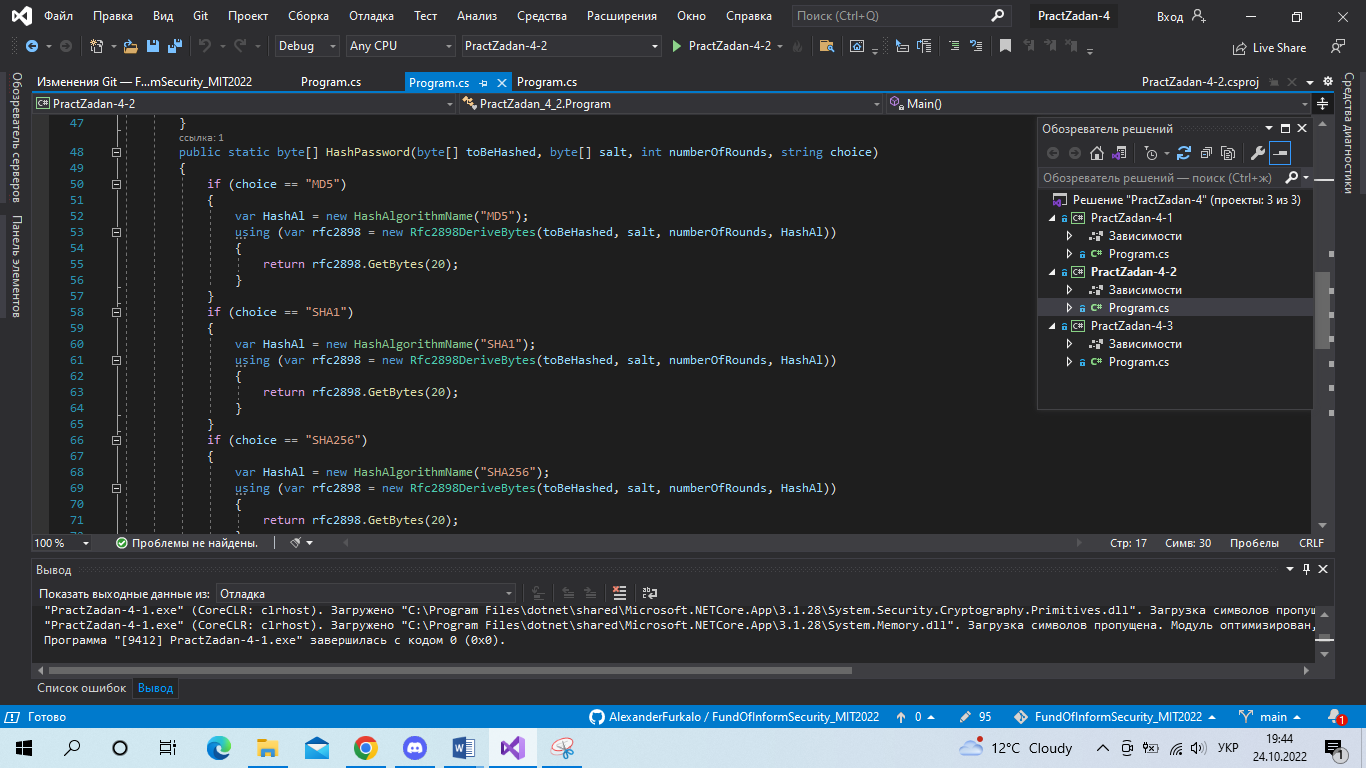
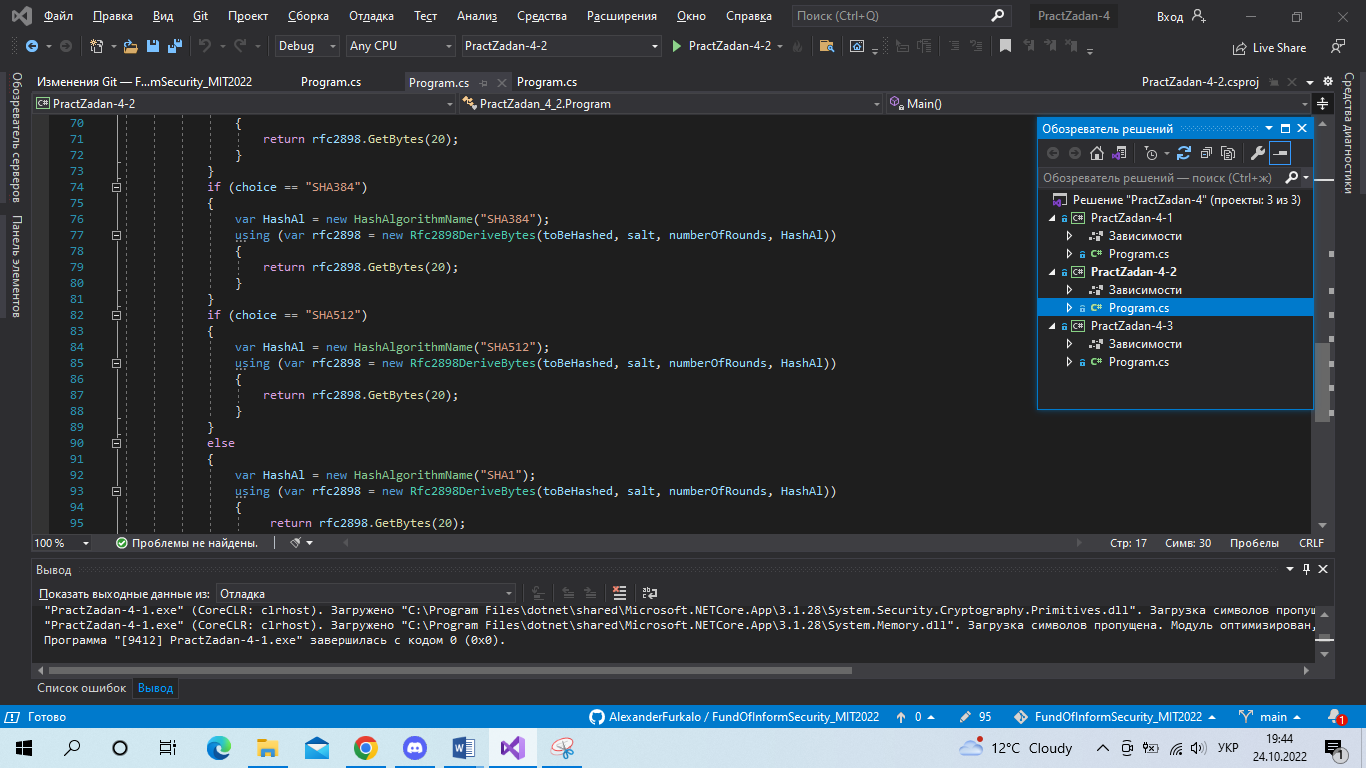
- У відкритому вигляді. Хоча цей спосіб і виконує свою функцію збереження паролів, він також не пропонує жодного захисту. Системі хоч і потрібно мати можливість верифікувати користувача, їй зовсім необов'язково знати його пароль. Якщо база даних буде зламана, і таблиця паролів буде викрадено, зловмиснику не доведеться докладати жодних інших зусиль для отримання доступу до профілів  
- У зашифрованому виді. Цей спосіб хоч і дає якийсь захист, у ньому все ж таки є проблема - існування ключа дозволяє не тільки зашифровувати, але і розшифровувати дані. Тому, з'являється проблема в тому, як саме потрібно зберігати ключ - адже отримавши його, злодій отримує фактичний доступ і до інших даних теж. Системі необов'язково потрібно вміти розшифровувати дані для перевірки правильності введення, тому є інший спосіб  
- Хешування паролів. Легко обчислити хеш-значення для будь-якого повідомлення, неможливість створити повідомлення з заданим хешем, неможливість змінити повідомлення без зміни хешу, неможливість знайти два однакових повідомлення з одним хешем – ці властивості дозволяють нам використовувати метод хешування для зберігання даних. Для покращення цього методу збільшують ентропію пароля (це міра надійності – наскільки легко його вгадати та відновити) – додавши до пароля декілька випадкових літер та цифр, його буде складніше підібрати. Рядок з випадкових символів називають “сіллю”.  
Про ще один метод зберігання даних (через PBKDF2) написано пізніше у цій роботі.

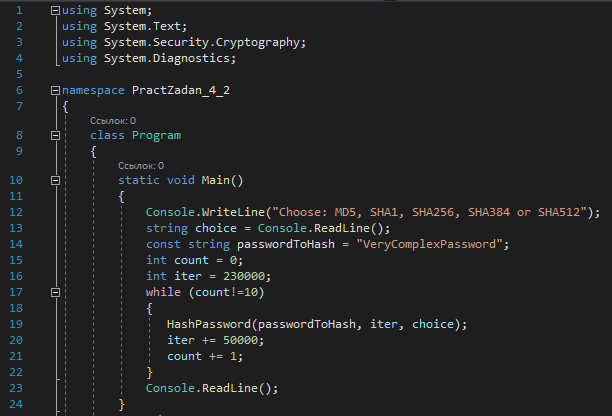
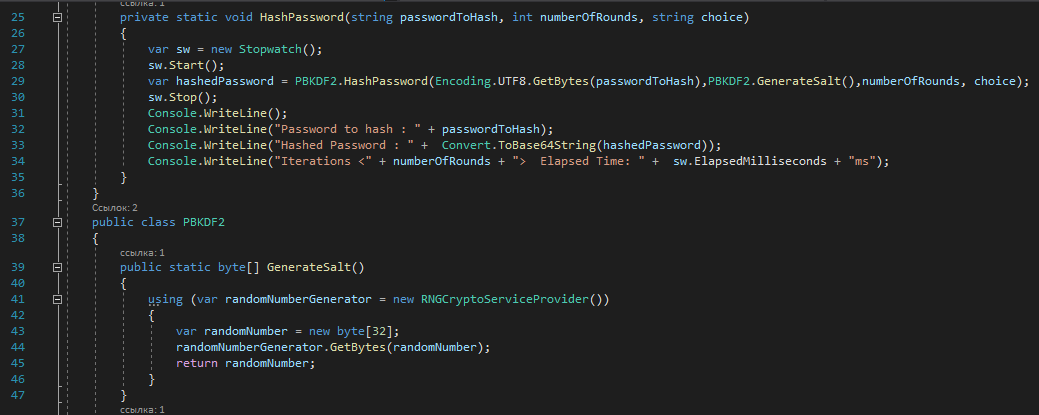
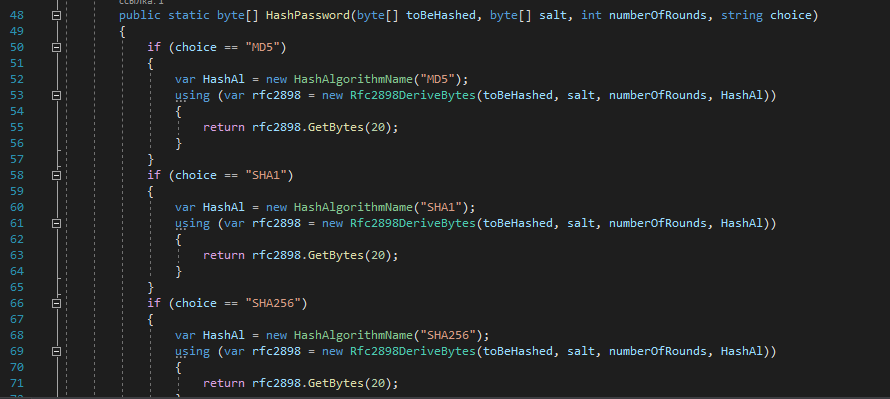
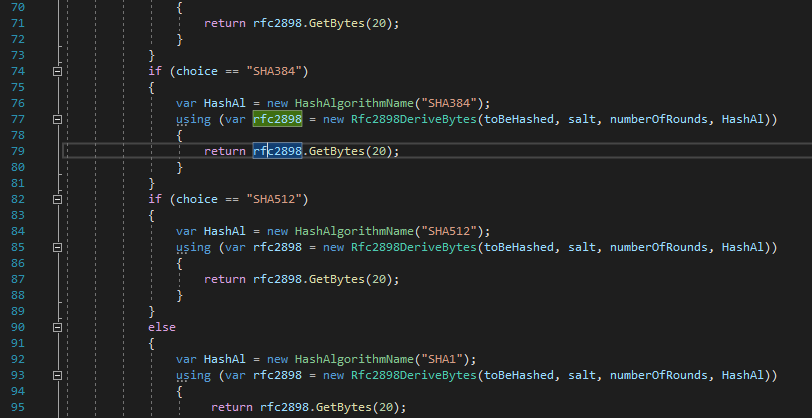
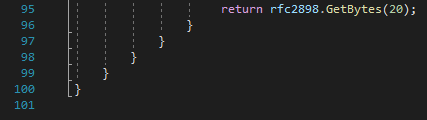
Перейдемо до першої з трьох програм:  
  
  


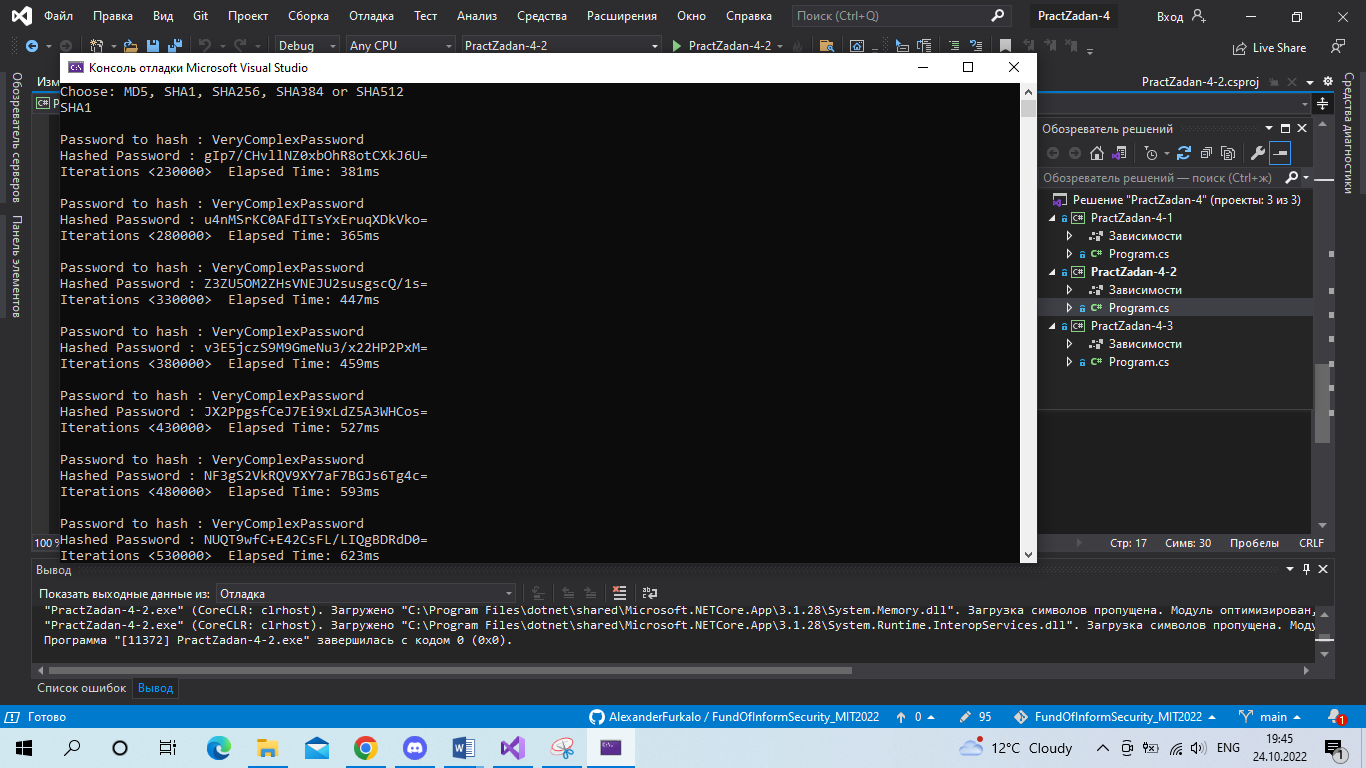
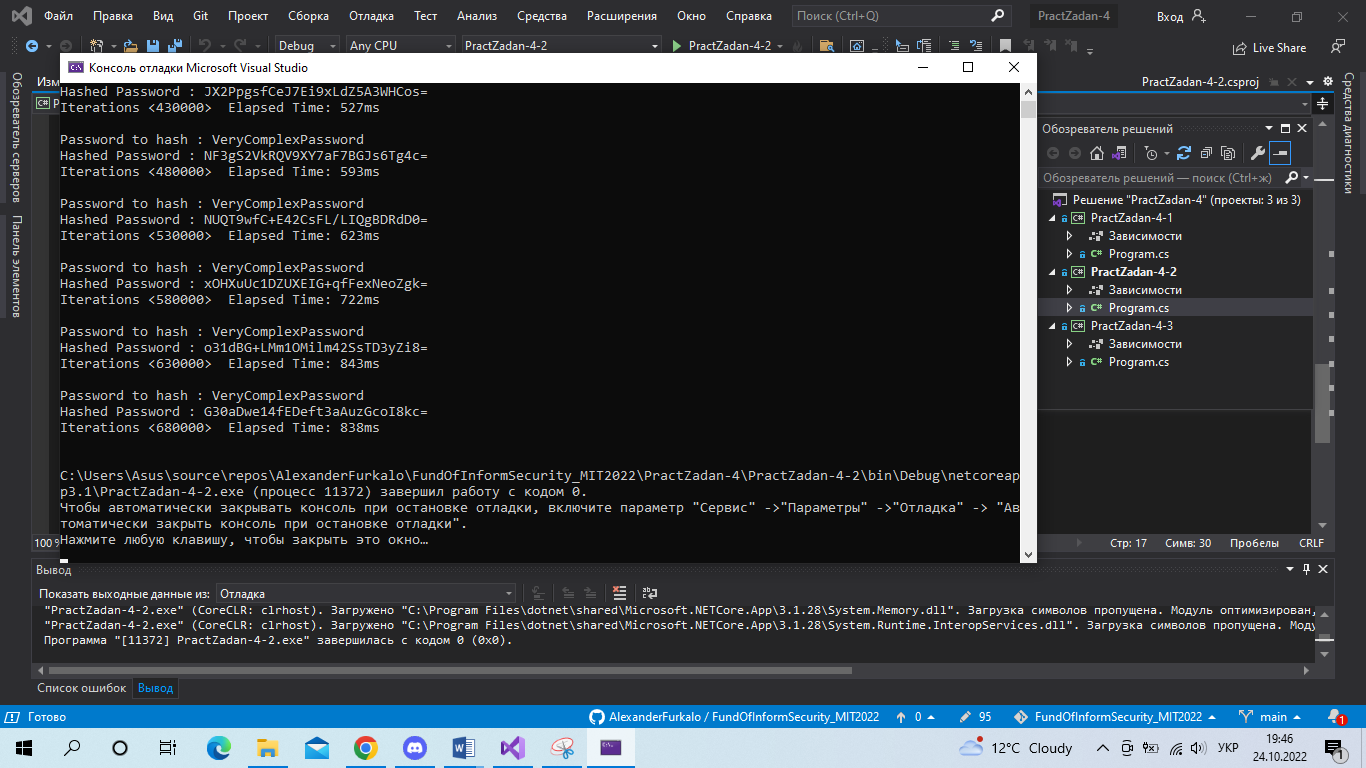
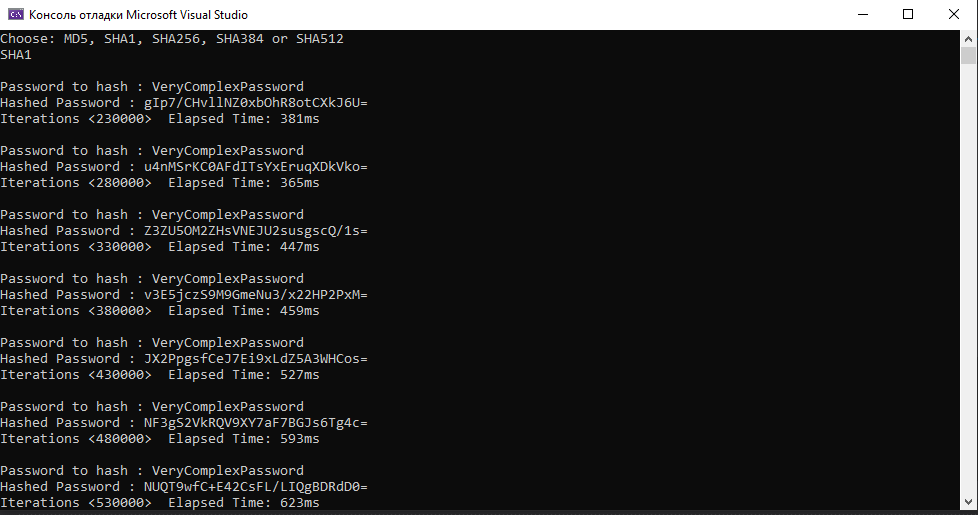
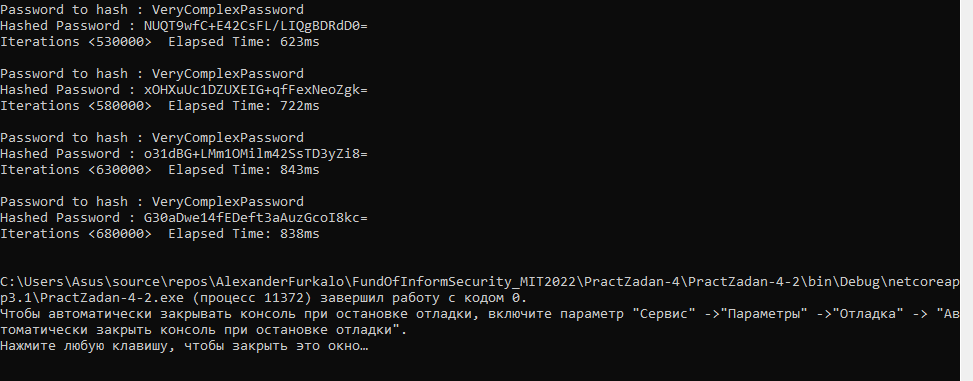
Трохи наблизивши:  
  
  


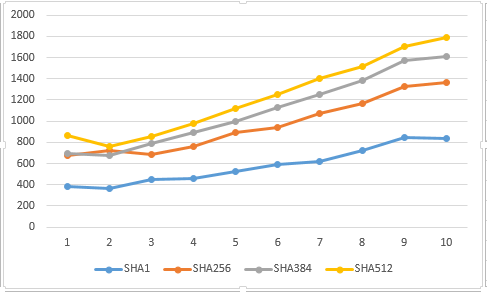
Результати роботи:  
  


У цьому коді надзвичайним є метод Buffer.BlockCopy. Розберемо його:  
Buffer.BlockCopy(Array src, int srcOffset, Array dst, int dstOffset, int count– це метод, який копіює вказане число байтів з вихідного масиву, починаючи з певного зміщення, у кінцевий масив, починаючи з певного зміщення. Параметри такі:  
- Array src: Вихідний масив  
- int srcOffset: Зміщення байтів у вихідному масиві  
- Array dst: Кінцевий масив  
- int dstOffset: Зміщення байтів у кінцевому масиві  
- int count: Кількість байтів для копіювання   
Цей метод використовується в коді для комбінування пароля з сіллю - спочатку копіюється число байтів із пароля, а потім, з урахуванням уже введених байтів, копіюється число байтів із солі.

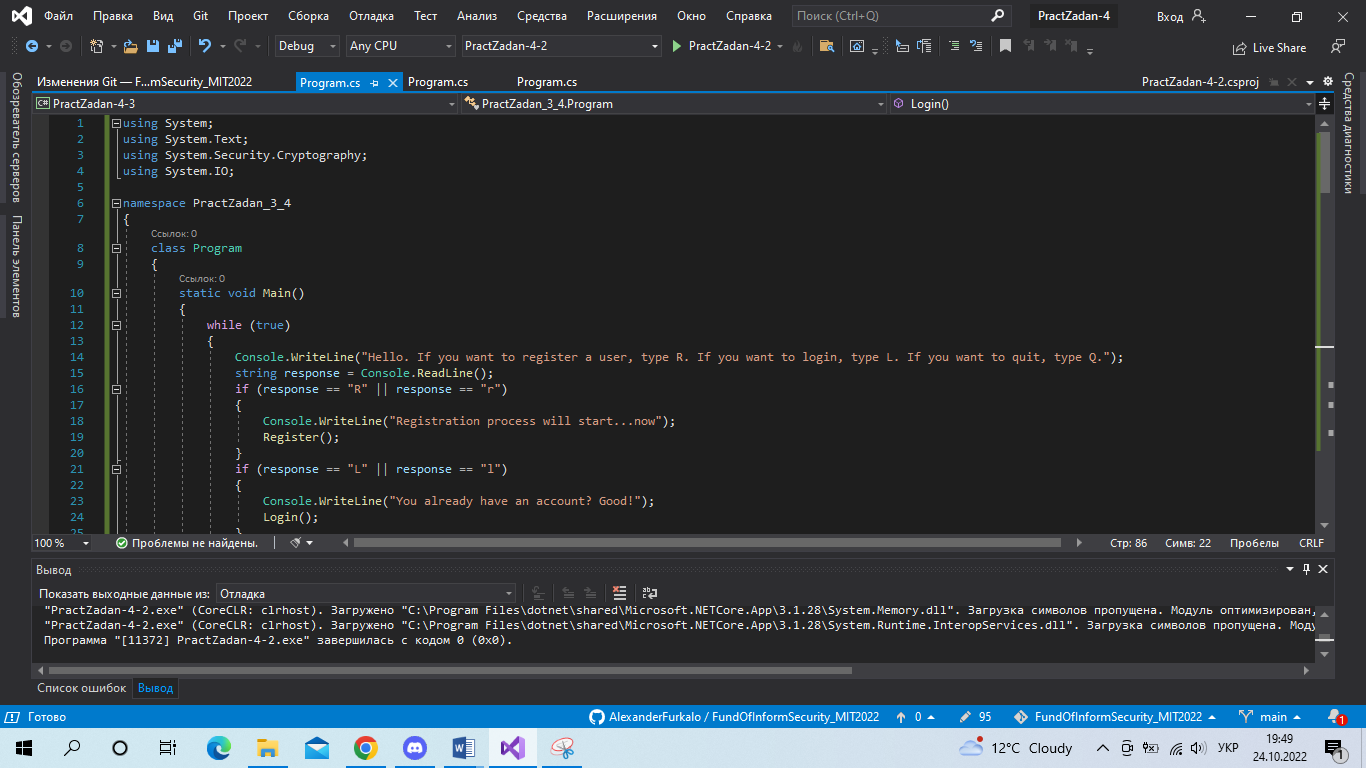
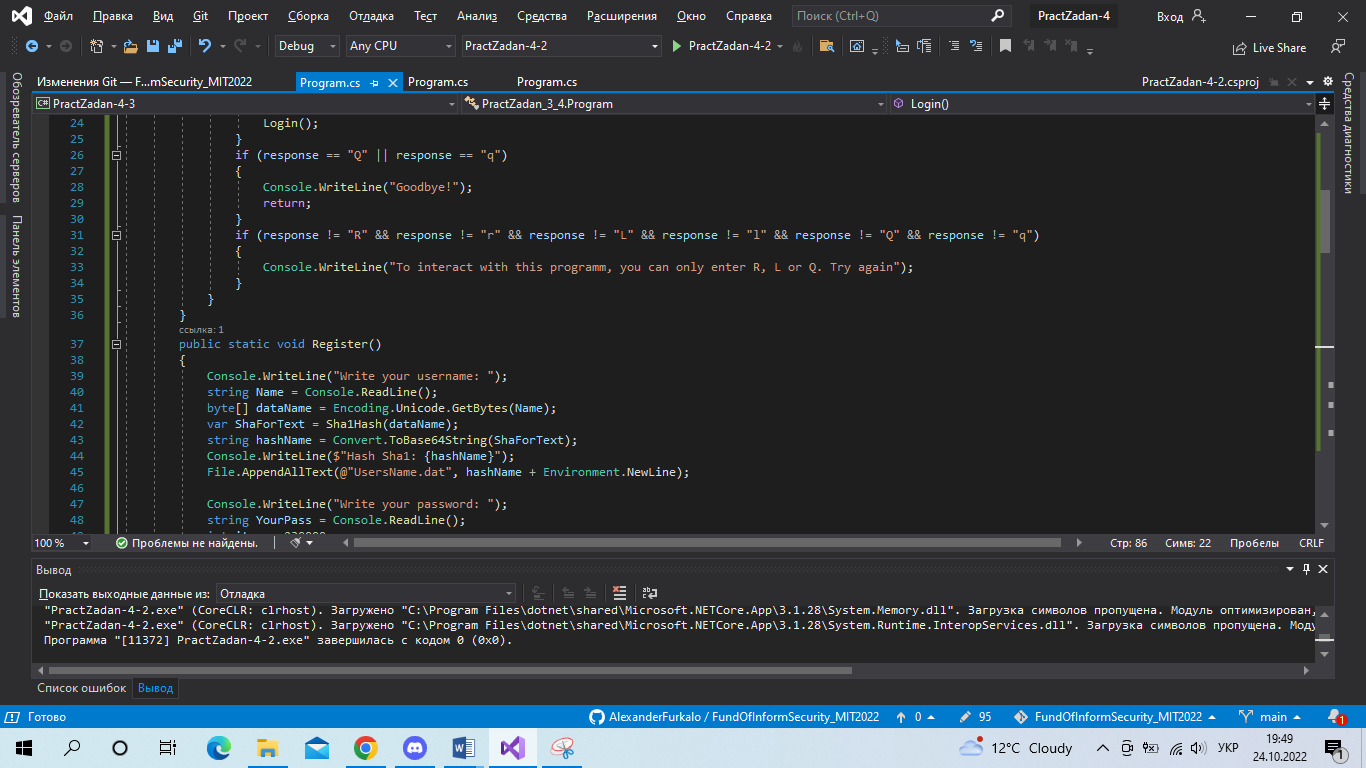
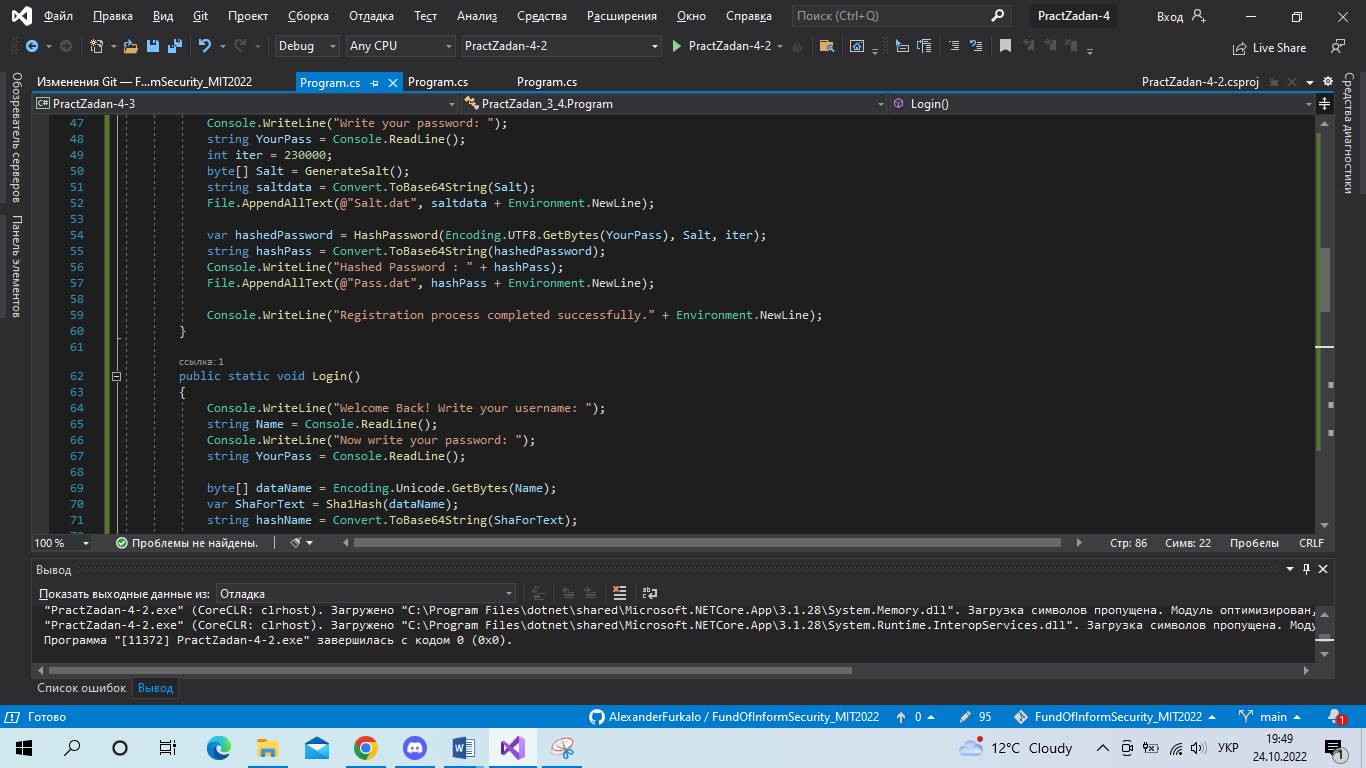
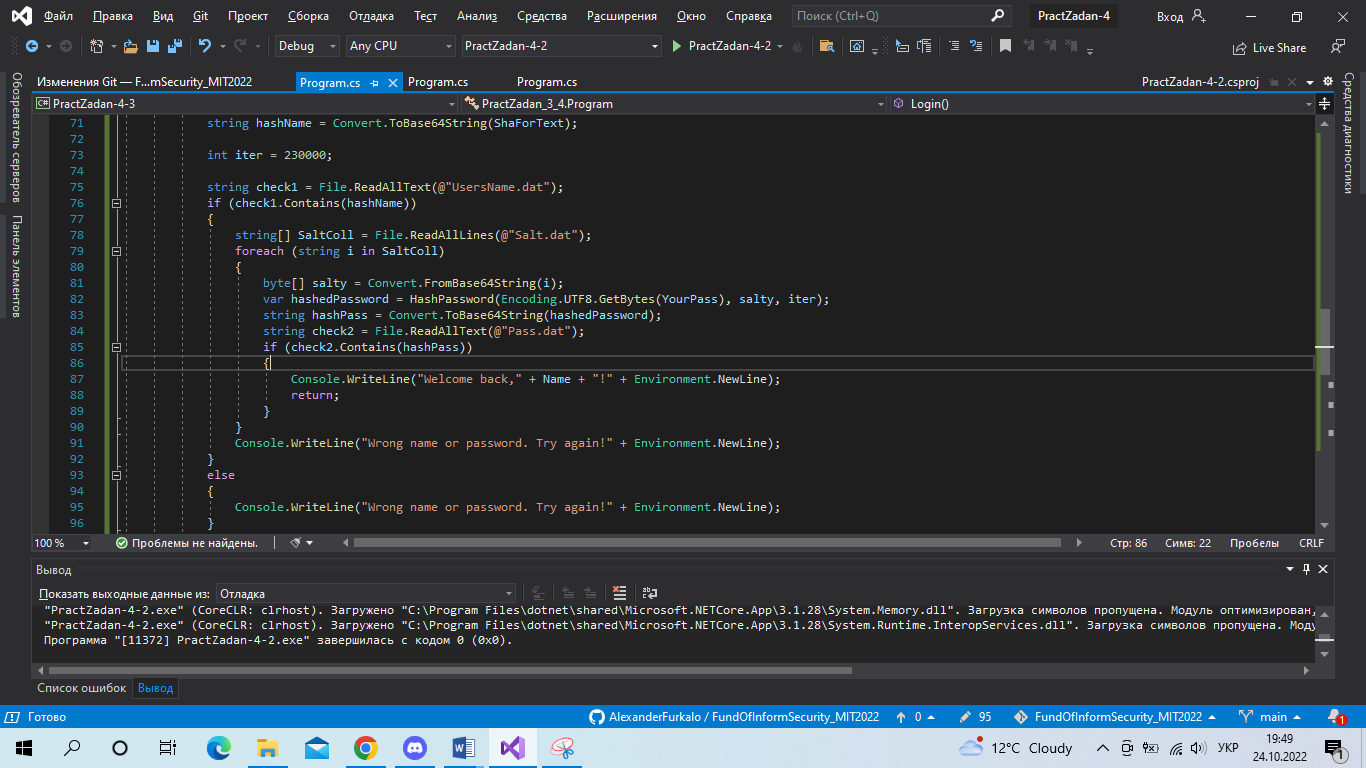
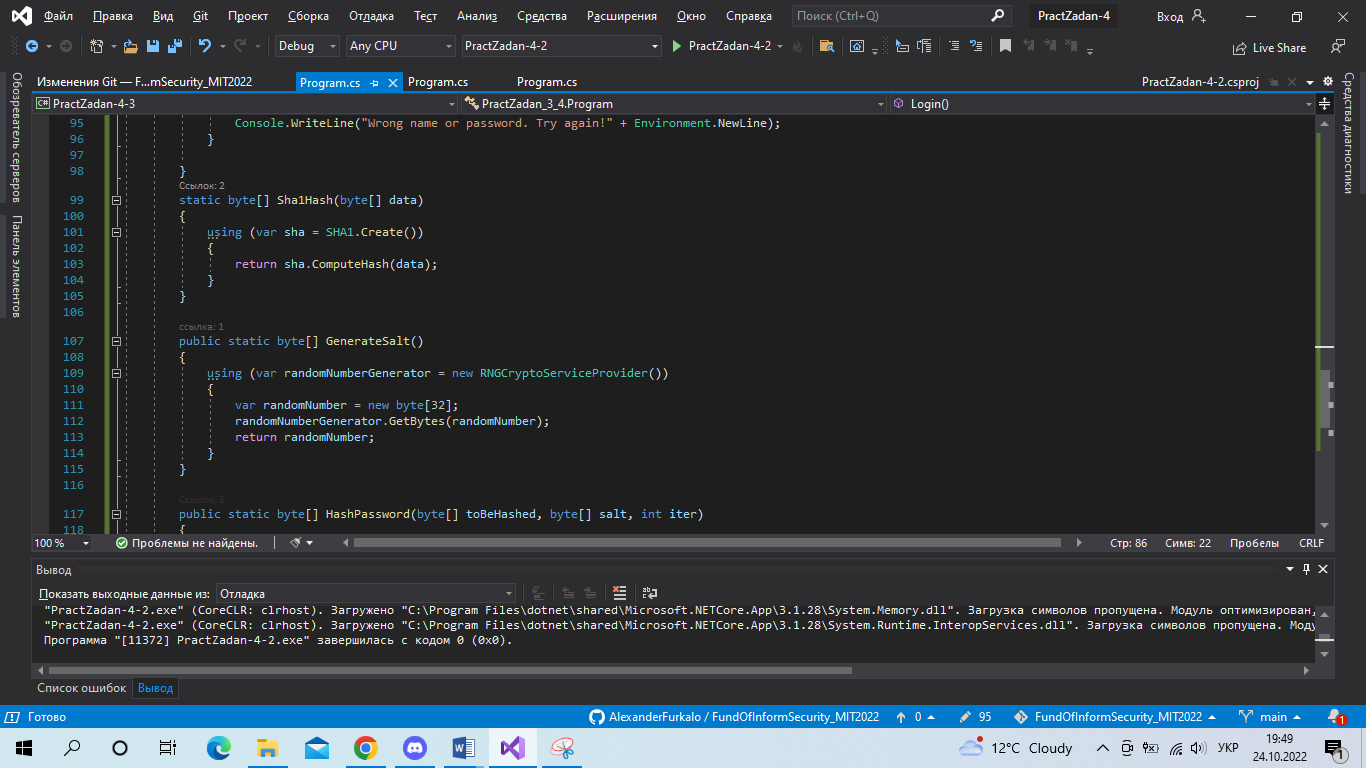
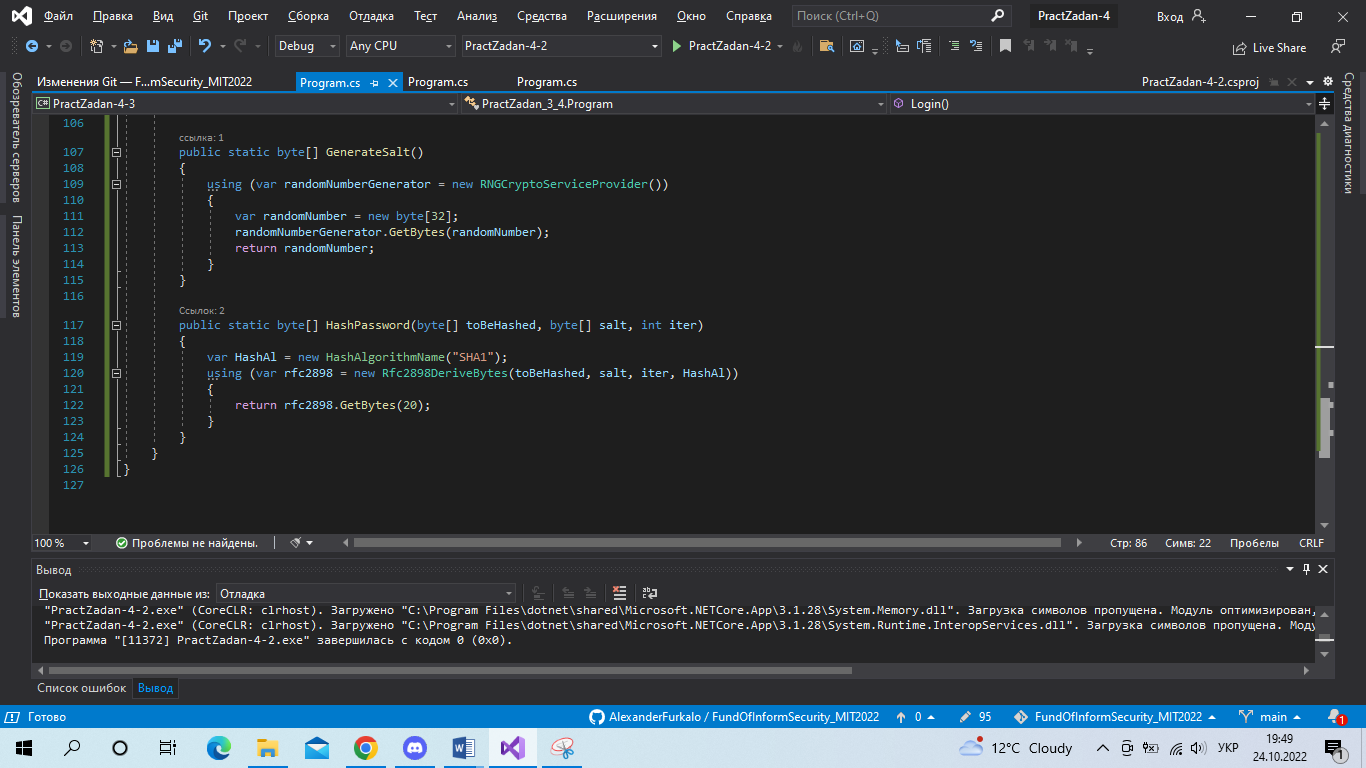
Наступне завдання:  
  
  
  


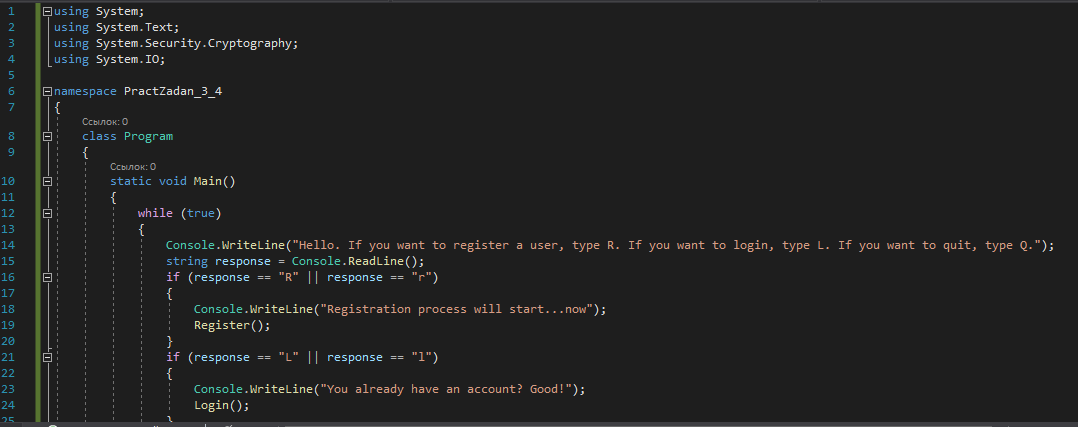
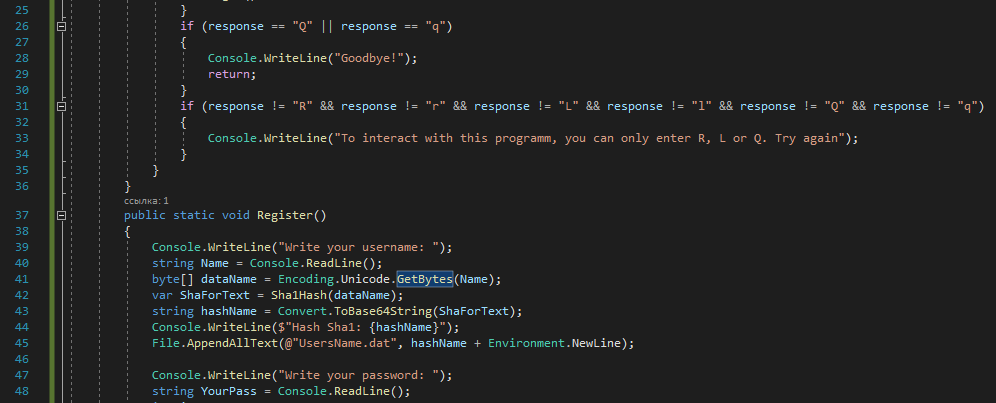
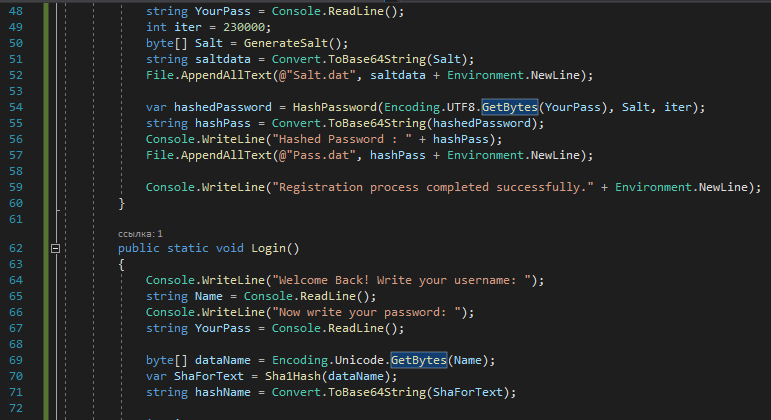
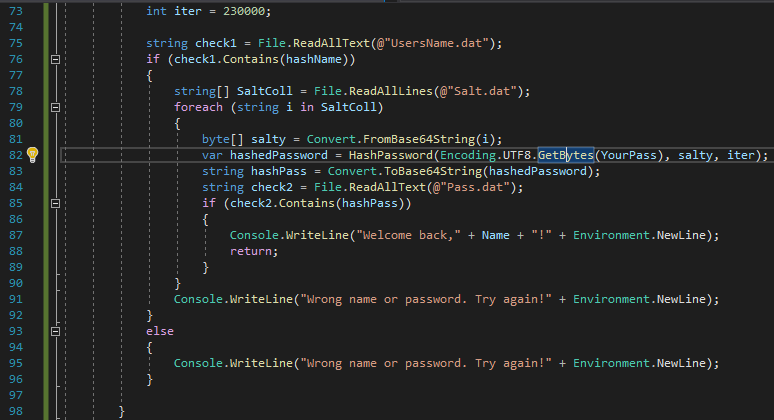
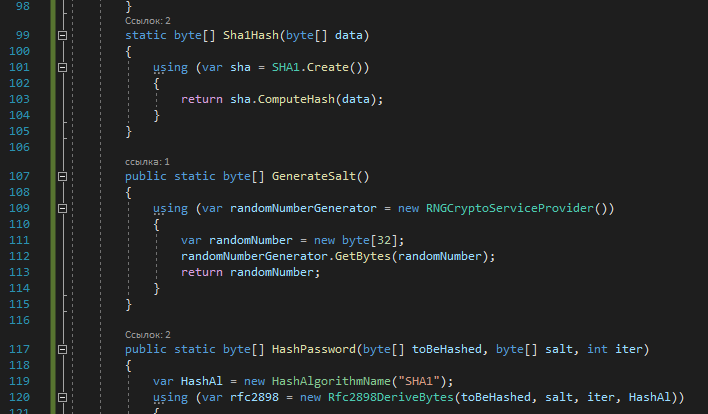
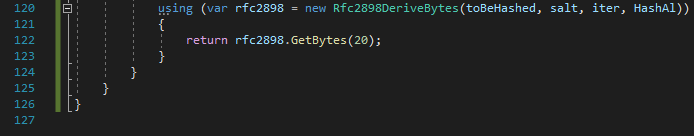
Трохи наблизивши:  
  
  
  
  


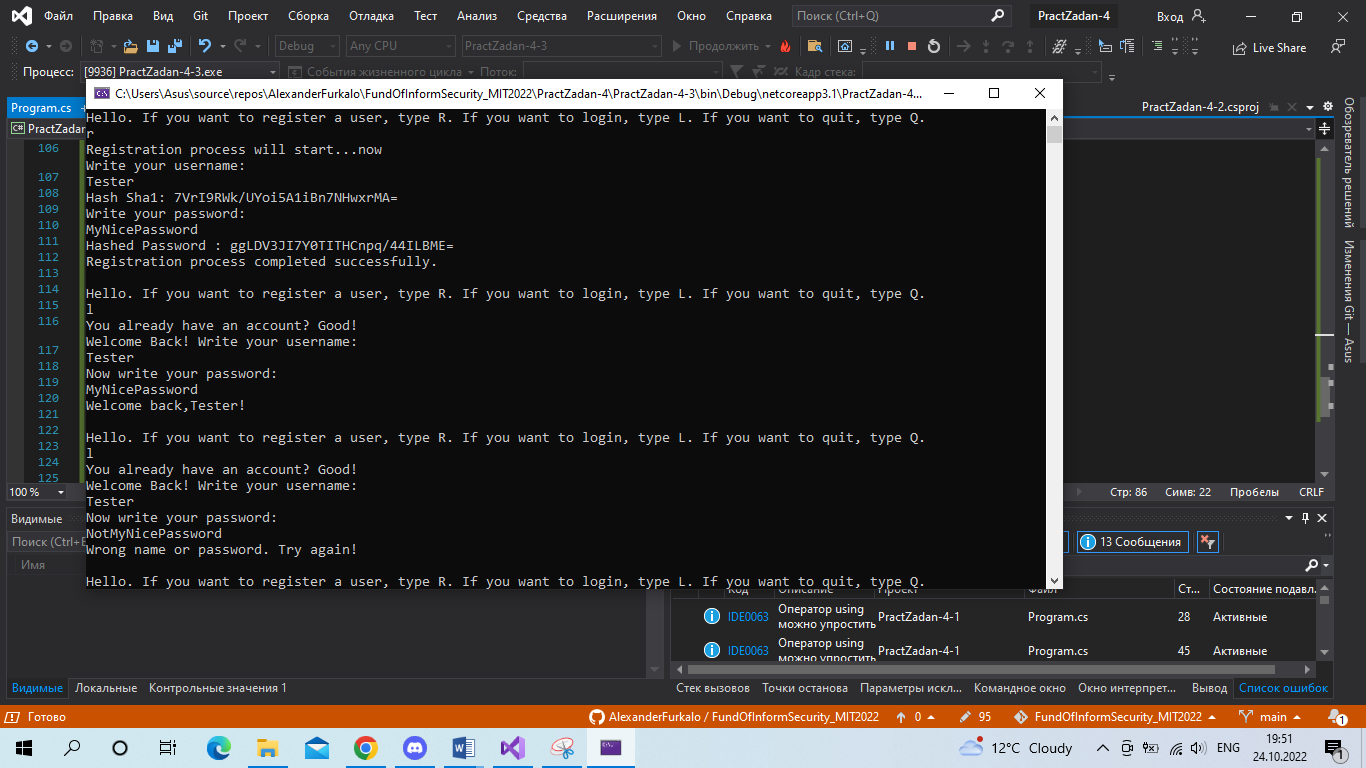
Результати роботи:  
  
  
  


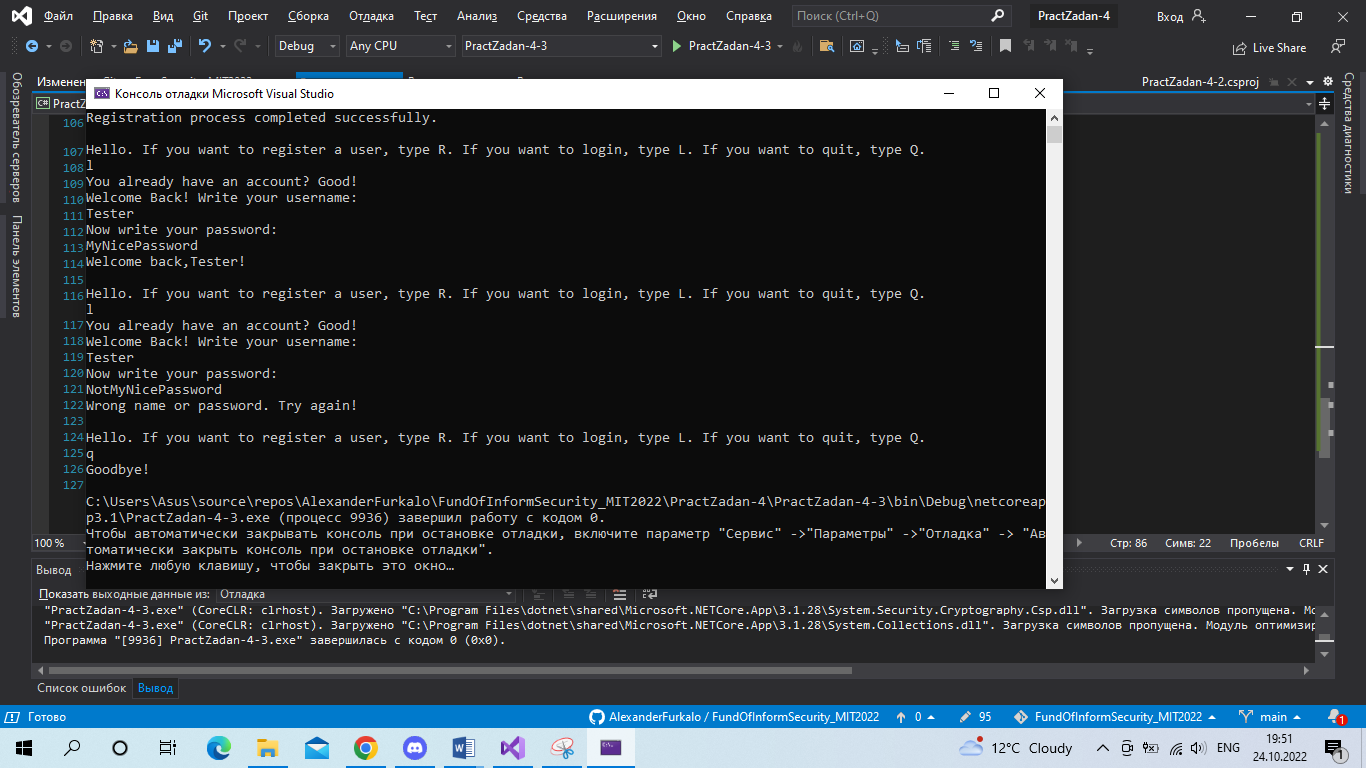
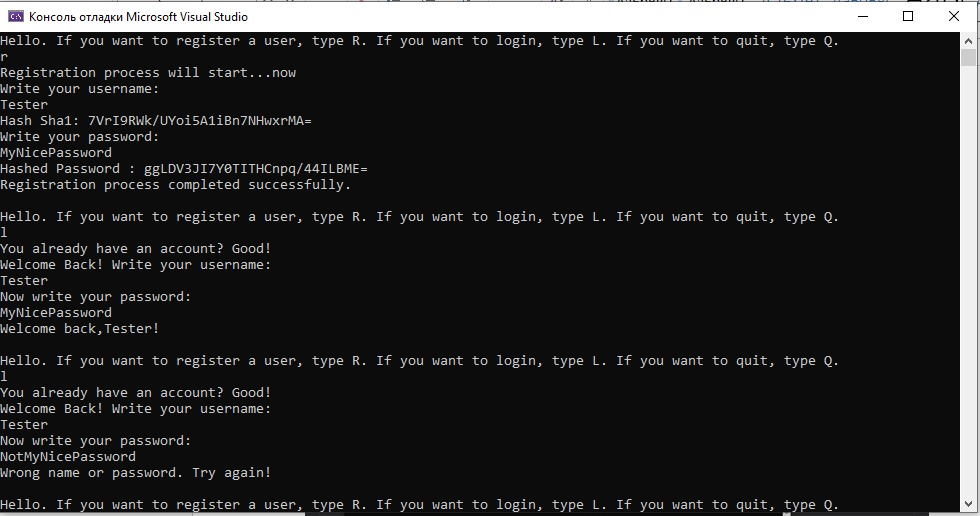
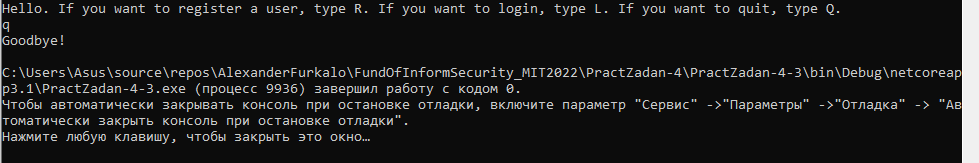
З якихось причин, алгоритм MD5 не працює під час застосування операції Rfc2898DeriveBytes, на відміну від інших протоколів. Тому розглянути ми можемо лише SHA1, SHA256, SHA384 та SHA512  


Проблема із збереженням паролів і їх хешуванням (навіть зі значенням солі) полягає в тому, що як мікропроцесори та графічні процесори стають швидшими, ми ризикуємо тим, що ті паролі, які зараз вважаємо безпечними, будуть скомпрометовані, оскільки процесори можуть швидше виконувати атаки грубої сили та атаки веселкової таблиці. Щоб вирішити це, використовують PBKDF2 (password-based key derivation function, або ж функція виведення ключа на основі пароля) - приймає пароль, сіль (щоб додати додаткову ентропію до пароля) і значення «кількість ітерацій». Значення кількості ітерацій повторює операцію хешування над паролем кілька разів, щоб створити похідний ключ для пароля, який можна зберігати в базі даних.

Наступне завдання:  
  
  
  
  
  


Трохи приблизивши:  
  
  
  
  
  


Результати роботи:  


Програма дуже схожа на те, що використовувалось у завданні минулої практичної роботи. Різниця в тому, що замість HMAC ми використовуємо інші методи:  
- Логін, сіль та комбінація пароля з сіллю зберігаються у трьох різних файлах (при цьому логін і соль - у вигляді хэш-кода). Це потім використовується для автентифікації користувача - цей процес відбувається спочатку з перебору наявного в документі імені (і якщо нічого не підходить, повертається назад в меню), а потім до введеного для перевірки пароля з ним комбінуються всі існуючі варіанти солі з другого файлу, і якщо жоден з отриманих результатів не дорівнює готовій комбінації пароля та солі з третього файлу, також повертається в меню. Якщо логін та пароль правильні, система вітає користувача.

Висновки

У цій практичній роботі, ми розглянули кілька способів зберігання даних та виявили їх плюси та мінуси. Розробили клас SaltedHash, що реалізує хешування паролів із додаванням додаткової ентропії та продемонстрували його роботу. Розробили клас PBKDF2, що має наступну функціональність: генерує "сіль", задає алгоритм хешування (MD5, SHA1, SHA256, SHA384, SHA512) та обчислює хеш для заданого числа ітерацій; створили програму, що обчислює час, витрачений на обчислення хешу для різного числа ітерацій (10 значень із кроком 50'000; перше значення = номер варіанта, тобто 23, \* 10'000 ), побудували графік залежності витраченого часу від числа ітерацій. Написали програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та "сіль" у пам'яті, реалізували можливість автентифікації за логіном і паролем.