Фінальний проект до курсу «Програмування на Java»

Студента 3-го курсу спеціальності «111 Математика» групи «Комп'ютерна математика 2»

Нікітчина Назарія

Постановка задачі:

Розробити клієнт-серверний застосунок, який дозволить зберігати на сервері зашифровані документи клієнта та надавати йому можливість скачати їх за авторизованим доступом.

Вирішення задачі:

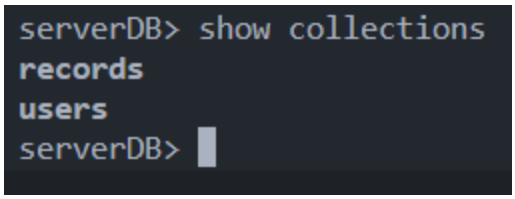
1. Для реалізації рішення нам зручно буде працювати в рамках однієї надійної сесії між сервером і клієнтом, отже доцільно використати протокол TCP.

2. Оскільки ми повинні мати авторизований доступ до даних на сервері, отже необхідно забезпечити надійне передавання паролю з клієнта на сервер і захист від атаки Man in the middle. Цю проблему можна вирішити , наприклад, такими двома шляхами:

   - використати Interlock protocol - Wikipedia оснований на асиметричній криптографії і який не потребує попередньої «підготовки» для з'єднання, проте в нашій ситуації такий підхід може бути ризикований, адже цей протокол може бути скомпрометований, коли зловмисник буде знати схему передачі інформації (контекст), що актуально в нашому випадку, бо спілкування між клієнтом і сервером відбувається за певним алгоритмом;

   - використовувати захищене з'єднання за протоколом TLS і SSL сертифікатами. Для уникнення спроб зловмисником підмінити SSL сертифікат кожен клієнт матиме при встановленні свій довірений SSL сертифікат яким зможе перевірити сертифікат сервера і переконатись, що сервер той за кого себе видає.
   Я скористався другим способом.

3. Для зберігання інформації на сервері потрібна база даних, яка міститиме інформацію про клієнтів та їх записи. Я використав NoSql базу даних MongoDB. Оскільки максимальний розмір одного документа в MongoDB 16 мегабайт, то я побудував архітектуру таким чином, що в одній колекції users знаходиться інформація по користувачу та масив з посиланнями на документи в колекції records в яких власне і зберігаються його записи.



4. Не можна зберігати паролі в базі даних у відкритому вигляді, щоб запобігти компрометації облікових записів користувачів як на нашому сервері , так і на інших ресурсах (багато людей використовують однакові паролі для різних систем, якими користуються) у разі витоку інформації з бази даних. Для цього я використав алгоритм хешування SHA на довжині 512 бітів, а для того, щоб зробити повністю неможливими спроби «брутфорсу» до кожного паролю я додаватиму випадково згенеровану «сіль» довжиною 128 бітів.

5. На стороні клієнта його запис шифруватиметься симетричним алгоритмом AES з довжиною ключа 128 бітів, що є оптимальним для нашого часу, адже навіть при невеликій довжині ключа(у асиметричних алгоритмах ми б потребували хоча б довжини в 4048біт) користувач матиме дуже надійний криптографічний захист своєї інформації. Для розшифрування запису користувач повинен знати не тільки ключ а ще й Initialization Vektor(IV) – він потрібен для того щоб потенційно інформації з однаковими ключами не мала однакового зашифрованого вигляду. Ключ та IV генеруються алгоритмом на стороні клієнта випадковим чином під час створення нового документа.

6. Для управління залежностями використаю систему збірки Maven.

Для розробки використовую текстовий редактор Visual Studio Code з розширенням Remote Development з конектом до Windows Subsystem for Linux.

Тест роботи javax.net.ssl.SSLSocket

на простій програмі, де сервер чекає повідомлення від клієнта а потім повідомляє, що він його отримав.

1. Бачимо, що сервер хоститься на ::: порт 8888 під протоколом TCP для IPv6(тобто за дефолтом навіть не на localhost, а так щоб можна було отримати з’єднання з іншого пристрою)

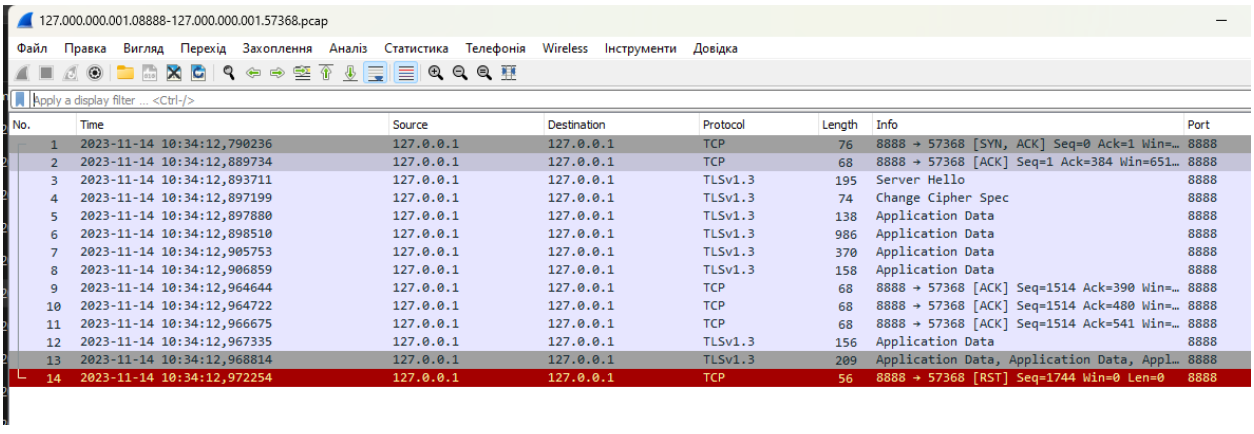
```
% sudo netstat -tnlp
[sudo] password for naza_ua:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
  PID/Program name
tcp        0      0 127.0.0.1:40839         0.0.0.0:*               LISTEN
  411/node
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
  104/systemd-resolve
tcp6       0      0 :::8888                 :::*                    LISTEN
  6654/java
tcp6       0      0 :::42685                :::*                    LISTEN
  8069/java
tcp6       0      0 :::39307                :::*                    LISTEN
  709/java
naza_ua@nazaua ~/Documents/Java/ClientServerEncrypted/Server/server_mongodb
```

2. Слухаємо порт 8888 та записуємо результат в .pcap файл для змоги потім зручно переглянути його у Wireshark.

```
naza_ua@nazaua ~/Documents/Java/ClientServerEncrypted/Server/server_mongodb
 % sudo tcpflow -i any -C -K port 8888


tcpflow[53214]: ensuring pcap core
reportfilename: ./report.xml
tcpflow: listening on any
^Ctcpflow: terminating orderly
naza_ua@nazaua ~/Documents/Java/ClientServerEncrypted/Server/server_mongodb
```

3. Спостерігаємо, що спочатку проходить стандартний TCP-handshake, а далі бачимо, що з’єднання вже захищене протоколом TLS



Демонстрація записів з бази даних після роботи програми

```
  {
    _id: ObjectId("65589b2bae89d2659b8eeeac"),
    username: 'naza_ua',
    salt: 'tKFDq1BdaDbrIDzr8RnKiw',
    password: 'bae762146b2fdf0ed8512be0ad334a69dfdba4cf005eaf82746b51efa3bb7c48433f5054a699338e8750bf66191dc39ee0cb6cee0859
e102b43290b14620bb4c',
    records: [ ObjectId("65589ef0780cdf5c3b94f0ce") ]
  }
]
serverDB>
```

```
serverDB> db.records.find().pretty()
[
  {
    _id: ObjectId("6557e2e0538f0b10db660159"),
    record: 'test',
    title: 'test'
  },
  {
    _id: ObjectId("65589ef0780cdf5c3b94f0ce"),
    record: 'PgxymN2OaxJTXIFho8SCG98vXlTTL9vaWh9l/1LNTyoEzYurj6puCmg2SxxRAuwv4hGDHGpz7vEBvXhE8A==',
    title: 'Test_Document'
  }
]
serverDB>
```

Задачі які будуть реалізовані в майбутньому

1. Можливість зберігати файли більші ніж 16 мегабайт: GridFS — MongoDB Manual.
2. Захист від NoSql ін’єкцій.