

**Міністерство освіти і науки України Національний
технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського» Факультет
інформатики та обчислювальної техніки Кафедра
обчислювальної техніки**

Розрахункова графічна робота
з дисципліни
«Безпека програмного забезпечення»

Виконав:

студент групи ІП-05

Гапій Денис Едуардович

Номер залікової: 0504

Перевірів:

доц. Волокита А. М.

Київ 2023

Тема: «Системи безпеки програм і даних.»

Завдання:

Базовий варіант. Реалізувати імітацію TLS \ SSL “рукоштовування” (10 балів):

1. Ініціювання клієнтом:

- Клієнт ініціює рукоштовування, надсилаючи повідомлення "привіт" (рандомно згенероване) на сервер.

2. Відповідь сервера:

- Сервер відповідає повідомленням "привіт сервера" (рандомно згенероване), що містить також SSL-сертифікат (.509). (Або для базового варіанту - сервер генерує public \ private ключі та відправляє public клієнту).

3. Автентифікація:

- Клієнт перевіряє SSL-сертифікат сервера в центрі сертифікації для підтвердження ідентичності сервера. (У випадку базового варіанту - пропустіть)

4. Обмін секретними рядками:

- Клієнт надсилає секрет premaster, який шифрується відкритим ключем сервера.

• Сервер розшифровує секрет premaster. 5. Генерація ключів сеансу:

- Клієнт і сервер генерують ключі сеансу з клієнтського та серверного випадкових рядків і секрету premaster. 6. Готовність клієнта та сервера:

- Клієнт та сервер надсилають повідомлення "готовий", зашифроване сеансовим ключем.

7. Завершення рукоштовування:

- Здійснюється безпечне симетричне шифрування, і рукоштовування завершується. • Зв'язок продовжується за допомогою ключів сеансу. Реалізувати передачу даних по захищеному каналу (наприклад, повідомлення чату, текстовий файл).

Виконання:

Лістинг server.js:

```
const tls = require('tls');  
  
const fs = require('fs');  
  
const crypto = require('crypto');  
  
const options = {  
  key: fs.readFileSync('server-key.pem'),  
  cert: fs.readFileSync('server-cert.pem'),
```

```

});

const server = tls.createServer(options, (cleartextStream) => {

  console.log('Server: Client connected');

  cleartextStream.on('data', (data) => {

    console.log('Server received data:', data.toString());

    if (data.toString().includes('hello')) {

      const serverHelloMessage = 'Hello from the server!';

      cleartextStream.write(serverHelloMessage);

      // Generate a premaster secret for demonstration purposes (replace
with your logic)

      const premasterSecret = 'ThisIsAPremasterSecret';

      // Encrypt the premaster secret using the client's public key with
PKCS#1 v1.5 padding

      const encryptedPremasterSecret = crypto.publicEncrypt(

        {

          key: fs.readFileSync('client-cert.pem'), // Client's
public key

          padding: crypto.constants.RSA_PKCS1_PADDING,

        },

        Buffer.from(premasterSecret, 'utf8') // Ensure proper
encoding

      );

      // Send the encrypted premaster secret to the client

      cleartextStream.write(encryptedPremasterSecret);

    }

  });

  cleartextStream.on('end', () => {

    console.log('Server: Client disconnected');

  });
});

```

```
});

server.listen(8080, () => {

    console.log('Server listening on port 8080');

});
```

Лістинг client.js:

```
const tls = require('tls');
const fs = require('fs');
const crypto = require('crypto');
const uuid = require('uuid');
const options = {
    key: fs.readFileSync('client-key.pem'),
    cert: fs.readFileSync('client-cert.pem'),
};
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";
const socket = tls.connect(8080, 'localhost', options, () => {
    console.log('Client: Connected to server');

    const randomUuid = uuid.v4();

    // Send the "hello" message with the random UUID to the server
    socket.write(`hello-${randomUuid}`);

    // Generate a premaster secret (for demonstration purposes, using a simple
    string)
    const premasterSecret = 'ThisIsAPremasterSecret';

    console.log('Client: Encrypting premaster secret:', premasterSecret);

    // Encrypt the premaster secret using the server's public key
    const encryptedPremasterSecret = crypto.publicEncrypt(
        {
```

```

    key: fs.readFileSync('server-cert.pem'), // Server's public key
    padding: crypto.constants.RSA_PKCS1_PADDING,
  },
  Buffer.from(premasterSecret)
);
socket.write(encryptedPremasterSecret);
socket.on('data', (data) => {
  console.log('Client received data:', data.toString());
});
socket.on('end', () => {
  console.log('Client: Connection closed');
});
});

```

Генерація серверного / клієнтського SSL-сертифікатів за допомоги openssl:

```
C:\Windows\System32\cmd.exe
```

```
S:\Dev\Studying\KPI-Studying\7th semester\Software Security\ngr>openssl req -x509 -nodes -newkey rsa:2048 -keyout server-key.pem -out server-cert.pem  
.....  
-----  
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:  
State or Province Name (full name) [Some-State]:  
Locality Name (eg, city) []:  
Organization Name (eg, company) [Internet Widgets Pty Ltd]:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:  
Email Address []:  
  
S:\Dev\Studying\KPI-Studying\7th semester\Software Security\ngr>openssl req -x509 -nodes -newkey rsa:2048 -keyout client-key.pem -out client-cert.pem  
.....  
-----  
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.
```

Результат запуску серверу:

```
PS S:\Dev\Studying\KPI-Studying\7th semester\Software Security\rgr> node server.js
Server listening on port 8080
Server: Client connected
Server received data: hello-d8c81179-284f-4e3b-a83e-77900f519a89
Server received data: 300HJnfqQ+llioBS;N%;0z?J*{Q;뵆P
궡XL[vC+VIE-(n!LrreS(On. <aalL[5
d'k}eyY[
Server: Client connected
Server received data: hello-89e4a398-d86d-422d-9acf-4561b7f30041
Server received data: !l
%y'd'E+F[uzh'i'Wk<fK:BQ+-j9?a?Xf=G00qfd6'c   FK
```

Результат запуску клієнту 1:

```
PS C:\Dev\studying\KPI-studying\7th semester\Software Security\rgr> node client.js
(node:13884) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use "node --trace-warnings ..." to show where the warning was created)
Client: Connected to server
Client: Encrypting premaster secret: ThisIsAPremasterSecret
Client received data: Hello from the server!
Client received data: w'05oN+000000Pw0000zZJDLT&000jy000000]?0H+000al000{00SA0000000000qz0Y00000Q<000C*000000000U000pE000is00000t000?+xtf:000E000e>0#00F00000000K00000000005u;0000lal000-00000//M00D0Kj0000e0++0gk<<0
```

Результат запуску клієнту 2:

```
PS S:\Dev\Studying\KPI-Studying\7th semester\Software Security\rngr> node client.js  

(node:21124) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.  

(Use `node --trace-warnings ...` to show where the warning was created)  

Client: Connected to server  

Client: Encrypting premaster secret: ThisIsAPremasterSecret  

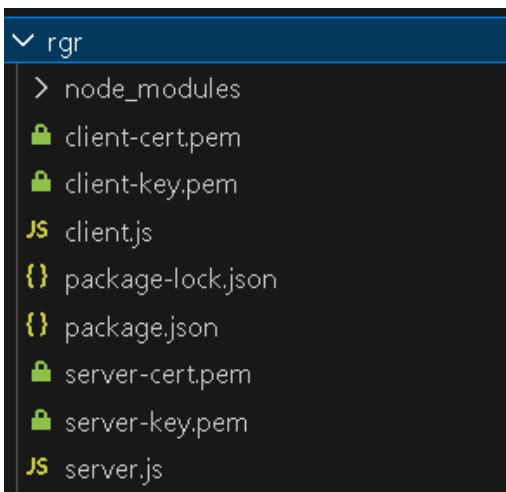
Client received data: Hello from the server!  

Client received data: T4XI...  

o[...]  

...<H..9
```

Структура проекту:



Теоретична частина:

1. Ініціювання клієнтом:

Теорія:

Рукописання TLS/SSL розпочинається з ініціювання клієнтом підключення.

Реалізація:

У файлі `client.js` клієнт встановлює TLS-з'єднання із сервером за допомогою методу `tls.connect`.

2. Відповідь сервера:

Теорія:

Сервер відповідає на ініціювання клієнта з'єднанням.

Реалізація:

У файлі `server.js` сервер слухає вхідні підключення за допомогою `tls.createServer`.

3. Повідомлення "Привіт":

Теорія:

Клієнт та сервер обмінюються повідомленнями "привіт" у рамках процесу рукоштовування.

Реалізація:

Клієнт ініціює рукоштовування, надсилаючи повідомлення "привіт" із випадковим UUID.

Сервер відповідає повідомленням "привіт".

4. Обмін секретом premaster:

Теорія:

Клієнт генерує секрет premaster і шифрує його за допомогою публічного ключа сервера.

Сервер розшифровує секрет premaster за допомогою свого приватного ключа.

Реалізація:

Клієнт генерує секрет premaster і шифрує його у файлі `client.js`.

Сервер розшифровує секрет premaster у файлі `server.js` за допомогою приватного ключа сервера.

5. Генерація ключа сеансу:

Теорія:

Сервер та клієнт використовують розшифрований секрет premaster для створення ключа сеансу.

Реалізація:

У файлі `server.js` сервер використовує розшифрований секрет premaster для генерації ключа сеансу.

6. Обмін зашифрованими повідомленнями:

Теорія:

Після рукоштовування клієнт та сервер можуть обмінюватися зашифрованими повідомленнями за допомогою отриманого ключа сеансу.

Реалізація:

У файлах `client.js` та `server.js` обмінюються зашифрованими повідомленнями, використовуючи отриманий ключ сеансу.

7. Завершення та захищена комунікація:

Теорія:

Рукоштовування завершується, і подальша комунікація захищена за допомогою ключа сеансу.

Реалізація:

Після обміну повідомленнями "готовий" у файлах `client.js` та `server.js` комунікація продовжується, використовуючи отриманий ключ сеансу.

Ці кроки реалізації охоплюють основні аспекти базового процесу рукоштовування TLS/SSL, включаючи обмін ключами, шифрування та захищену комунікацію.

Цей проект дозволяє отримати базове розуміння процесу рукоштовування TLS/SSL. Відповідно до теоретичних принципів, клієнт та сервер успішно обмінюються ключами, встановлюють безпечне з'єднання та продовжують захищену комунікацію. Однак слід зауважити, що це лише спрощений приклад для освітніх цілей, і в реальних сценаріях слід використовувати більш продумані та безпечні практики, такі як використання сертифікатів від довірених центрів сертифікації та обробка помилок.