

Universitatea "Politehnica" din București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Analiza experimentală a protocolului TLS 1.3

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de

*Inginer în domeniul Electronică, Telecomunicații și Tehnologia
Informație*

programul de studii de licență *Rețele și software de telecomunicații*

Conducător științific

Conf.Dr.Ing. Octavian CATRINA

Absolvent

Hasan Bogdan

2021

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **HASAN D. Bogdan , 444D-RST**

1. Titlul temei: Analiză experimentală a protocolului TLS 1.3

2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):

Serviciile oferite în Internet asigură securitatea capăt-la-capăt a comunicațiilor folosind protocolul TLS (Transport Layer Security). Noua versiune, TLS 1.3 (2018), este rezultatul unui efort substanțial de reproiectare, pentru a elimina o serie de deficiențe perpetuate de la SSL 3.0 până la TLS 1.2. Obiectivul acestui proiect este să analizeze soluțiile criptografice folosite în TLS 1.3, evidențiind diferențele față de TLS 1.2. Proiectul va include următoarele activități:

- Analiza noilor soluții criptografice folosite în TLS 1.3 pentru stabilirea cheilor secrete de sesiune ("TLS Handshake") și pentru protejarea traficului de date ("TLS Record"). Evidențierea diferențelor dintre soluțiile folosite în TLS 1.3 și în versiunile precedente, precum și a avantajelor acestor noi soluții.
- Implementarea unei aplicații client-server cu comunicații protejate de TLS, folosind implementări existente ale protocolului în limbajele Java și/sau C. Scopul acestei aplicații este de a demonstra soluții de implementare a comunicațiilor protejate de TLS și de a permite un studiu experimental detaliat al funcționării protocolului, pentru diverse variante de configurare: TLS 1.3 sau TLS 1.2; variante ale protocolului de stabilire a cheilor de sesiune (autentificare prin semnătură și certificat pentru cheie publică, autentificare cu cheie secretă prestabilită), variante ale algoritmilor criptografici utilizați în stabilirea cheilor și transferul datelor.
- Testarea aplicației, analiza experimentală a funcționării protocolului TLS 1.3 și comparație cu TLS 1.2.

3. Discipline necesare pt. proiect:

Securitatea Serviciilor și Rețelelor, Arhitecturi și Protocoale de Comunicații, Rețele și Servicii

4. Data înregistrării temei: 2020-12-04 14:02:04

Conducător(i) lucrare,
Conf.Dr.Ing. Octavian CATRINA

Student,
HASAN D. Bogdan

Director departament,
Conf. dr. ing. Șerban OBREJA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **600aa73646**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Titlul complet al proiectului*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer/ Master* în domeniul *domeniul***, programul de studii *program**** este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, *data*

10.09.2021

Absolvent *Prenume NUME*



(semnătura în original)

Cuprins

| | |
|---|----|
| CAPITOLUL 1. INTRODUCERE..... | 13 |
| Capitolul 1.1 Motivația lucrării | 14 |
| CAPITOLUL 2. Sinteza arhitecturii și procedurile protocolului TLS | 15 |
| Capitolul 2.1 Arhitectura TLS..... | 15 |
| Capitolul 2.2 Protocolul TLS Handshake..... | 16 |
| Capitolul 2.3 Protocolul TLS Record..... | 18 |
| Capitolul 2.4 Protocolul TLS Alert | 18 |
| Capitolul 2.5 Protocolul Change Cipher Spec | 19 |
| Capitolul 2.6 Autentificarea | 19 |
| Capitolul 2.7 Criptarea | 21 |
| Capitolul 2.8 Criptarea cu cheie publică | 21 |
| Capitolul 2.9 – Criptarea simetrică sau criptarea cu cheie privată..... | 22 |
| Capitolul 2.10 – Algoritmi, Ciphers si Cipher suites..... | 22 |
| Capitolul 2.11 – Infrastructura cheii publice PKI..... | 23 |
| Capitolul 2.12 Protocolul TLS 1.3 in comparație cu TLS 1.2 | 23 |
| Capitolul 3 Folosirea certificatelor..... | 25 |
| Capitolul 3.1 Certificate si CA..... | 25 |
| Capitolul 3.2 Generarea de containere de chei in format JKS | 25 |
| Capitolul 3.3 Generarea de certificate folosind “OpenSSL” | 27 |
| CAPITOLUL 4. STUDIUL DE CAZ..... | 28 |
| Capitolul 4.1 Descrierea soluțiilor propuse | 28 |
| Capitolul 4.2 Aplicatia EchoServer | 30 |
| Capitolul 4.3 Aplicatia EchoClient | 32 |
| Capitolul 4.4 Aplicatia Server..... | 33 |
| Capitolul 4.5 Aplicatia EchoSSLClient | 35 |
| Capitolul 4.6 Aplicatiile ClassServer si ClassFileServer..... | 37 |
| Capitolul 4.6.1 Aplicatia ClassServer | 38 |
| Capitolul 4.6.2 Aplicatia ClassFileServer | 40 |
| Capitolul 4.7 Aplicatia SSLSocketAuth | 43 |
| Capitolul 5 Testare/Utilizare | 46 |
| Capitolul 5.1 Pornirea aplicatiilor in IDE-ul IntelliJ | 46 |
| Capitolul 5.2 Utilizarea Wireshark..... | 47 |
| Capitolul 5.3 Functionarea aplicatiilor EchoServer si EchoClient..... | 48 |
| Capitolul 5.4 Pornirea aplicatiilor EchoServer si EchoClient. | 49 |

| | |
|---|----|
| Capitolul 5.5 Functionarea aplicatiilor EchoSSLServer si EchoSSLClient | 50 |
| Capitolul 5.6 Utilizarea tool-ului Wireshark pentru a captura traficul dintre EchoClient si EchoServer | 50 |
| Capitolul 5.7 Demonstrarea functionarii aplicatiei Server. | 51 |
| Capitolul 5.8 Testarea aplicatiei EchoSSLClient..... | 56 |
| Capitolul 5.9 Testarea aplicatiilor ClassFileServer si SSLClientAuth | 57 |
| Concluzii și Direcții de Viitor | 63 |
| Bibliografie..... | 64 |
| Anexe..... | 66 |

Listă de figuri

| | |
|---|----|
| Figura 2. 1 Arhitectura protocolului TLS | 15 |
| Figura 2. 2 Localizarea protocolului TLS între spațiul utilizatorului și Kernel | 16 |
| Figura 2.3 Schimbul de mesaje între emițător și receptor | 17 |
| Tabel 1 Alarmer generate de protocolul TLS “alert” | 18 |
| Figura 2. 4 Schimbul de cifru în modelul client-server | 19 |
| Figura 2. 5 Procesul de autentificare al protocolului TLS | 20 |
| Figura 2.6 Procesul de criptare între două entități la nivel minimal..... | 21 |
| Tabel 2 Etapele Handshake-ului al TLS 1.3 | 24 |
| Figura 3. 1 Mesajul help al keytool..... | 25 |
| Figura 3. 2 Parola keystore | 26 |
| Figura 3. 3 Completarea chestionarului keystore-ului | 26 |
| Figura 3. 4 Confirmarea informatiilor introduse in keystore | 26 |
| Figura 3. 5 Generare certificat openssl | 27 |
| Figura 4. 1 Comunicația între client și server fără criptare | 29 |
| Figura 4. 2 Comunicația între client și server folosind algoritm de criptare..... | 29 |
| Figura 5. 1 Pornirea aplicatiei EchoServer..... | 46 |
| Figura 5. 2 Confirmarea pornirii aplicatiei EchoServer | 46 |
| Figura 5. 3 Confirmarea pornirii aplicatiei EchoClient | 47 |
| Figura 5. 4 Confirmarea conexiunii clientului | 47 |
| Figura 5. 5 Utilizarea Wireshark pe traficul local | 47 |
| Figura 5. 6 Aplicarea filtrului pentru port in Wireshark | 48 |
| Figura 5. 7 Trimiterea unui mesaj folosind EchoClient..... | 48 |
| Figura 5. 8 Afisarea mesajului trimis..... | 48 |
| Figura 5. 9 Folosirea cuvântului “exit” pentru terminarea aplicatiei EchoClient | 48 |
| Figura 5. 10 Pornirea aplicatiei EchoServer | 49 |
| Figura 5. 11 Confirmarea pornirii aplicatiei EchoServer | 49 |
| Figura 5. 12 Confirmarea pornirii aplicatiei EchoClient..... | 49 |
| Figura 5. 13 Confirmarea conexiunii clientului | 50 |
| Figura 5. 14 Trimiterea unui mesaj folosind EchoClient | 50 |
| Figura 5. 15 Afisarea mesajului trimis | 50 |
| Figura 5. 16 Captura trafic Wireshark dintre EchoServer si EchoClient | 51 |
| Figura 5. 17 Pornirea aplicatiei Server cu negocierea versiunii TLS 1.3 | 51 |
| Figura 5. 18 Pornirea aplicatiei EchoSSLClient fara logare cu negocierea versiunii TLS 1.3..... | 52 |
| Figura 5. 19 Captura de trafic asupra negocierii versiunii TLS 1.3 | 52 |
| Figura 5. 20 Pornirea aplicației Server cu negocierea versiunii TLS 1.2 | 53 |
| Figura 5. 21 Captură de trafic asupra negocierii versiunii TLS 1.2 | 53 |
| Figura 5. 22 Certificatul transmis folosind versiunea 1.2 a protocolului TLS | 54 |
| Figura 5. 23 Noul proces de "handshake" al versiunii 1.3 a protocolului TLS..... | 54 |
| Figura 5. 24 Folosirea OpenSSL pentru verificarea versiunii de TLS | 55 |
| Figura 5. 25 OpenSSL conectare cu versiunea 1.2 la un server 1.3 | 56 |
| Figura 5. 26 Eroare aplicatie “Server” cand versiunile negociate difera. | 56 |
| Figura 5. 27 Logarea mesajului de handshake de catre client – negocierea de cipher suites | 57 |
| Figura 5. 28 Logarea mesajului de handshake de catre client – negocierea versiunii protocolului. | 57 |
| Figura 5. 29 Logarea mesajului de handshake de catre client – prezentarea certificatului. | 57 |
| Figura 5. 30 Pornirea aplicatiei “ClassFileServer” cu telecomunicatie necriptata. | 57 |
| Figura 5. 31 Accesarea unui fisier de test prin HTTP | 58 |

| | |
|---|----|
| Figura 5. 32 Pornirea aplicatiei “ClassFileServer” cu comunicatie incryptata..... | 58 |
| Figura 5. 33 Serverul nu mai accepta “HTTP” ci doar “HTTPS” | 58 |
| Figura 5. 34 Comunicatie nesigura cand folosim certificate care nu sunt de incredere..... | 59 |
| Figura 5. 35 Certificat nevalid..... | 59 |
| Figura 5. 36 Afisarea fisierului in consola | 60 |
| Figura 5. 37 Crearea “keystore-ului” “clientAuth” | 60 |
| Figura 5. 38 Extragerea certificatului | 60 |
| Figura 5. 39 Modificarea proprietatilor de sistem | 61 |
| Figura 5. 40 Modificarea noului container de chei..... | 61 |
| Figura 5. 41 Adaugarea unui certificat intr-un “truststore” | 62 |

Listă de tabele

| | |
|---|----|
| Tabel 1 Alarme generate de protocolul TLS “alert” | 18 |
| Tabel 2 Etapele Handshake-ului al TLS 1.3 | 24 |

Lista acronimelor

| | |
|--|---|
| CA – Certificate Authority | MAC -Message authentication code |
| CMD – Command Line Interpreter | OCSP – Online Certificate Status Protocol |
| CRL – Certificate Revocation List | PKI – Public Key Infrastructure |
| FTPS – File Transfer Protocol Secure | SMTP – Simple Mail Transfer Protocol |
| HTTP - Hyper Text Transfer Protocol; | SSL – Secure Sockets Layer |
| HTTPS - Hyper Text Transfer Protocol Secure; | TLS - Transport Layer Security |
| IDE – Integrated Development Environment | HTML - Hypertext Markup Language |

CAPITOLUL 1. INTRODUCERE

Această lucrare are în vedere compararea versiunilor 1.3 și 1.2 ale protocolului TLS. Obiectivul acestui proiect este să analizeze soluțiile criptografice folosite în TLS 1.3, evidențiind diferențele față de TLS 1.2. Proiectul va include următoarele activități:

- Analiza noilor soluții criptografice folosite în TLS 1.3 pentru stabilirea cheilor secrete de sesiune ("TLS Handshake") și pentru protejarea traficului de date ("TLS Record"). Evidențierea diferențelor dintre soluțiile folosite în TLS 1.3 și în versiunile precedente, precum și a avantajelor acestor noi soluții;
- Implementarea unei aplicații client-server cu comunicații protejate de TLS, folosind implementări existente ale protocolului în limbajul Java. Scopul acestei aplicații este de a demonstra soluții de implementare a comunicațiilor protejate de TLS și de a permite un studiu experimental detaliat al funcționării protocolului, pentru diverse variante de configurare: TLS 1.3 sau TLS 1.2; variante ale protocolului de stabilire a cheilor de sesiune (autentificare prin semnătură și certificat pentru cheie publică, autentificare cu cheie secretă prestabilită), variante ale algoritmilor criptografici utilizați în stabilirea cheilor și transferul datelor;
- Testarea aplicației, analiza experimentală a funcționării protocolului TLS 1.3 și comparație cu TLS 1.2.

În realizarea acestui proiect vom utiliza o suită de protocole și sisteme după cum urmează:

- Client – o aplicație software care comunică cu un server;
- Server – o aplicație software care oferă un serviciu unui alt program pe calculator, numit client;
- TLS – Transport Layer Security este un protocol care asigură comunicații sigure pe Internet;
- SSL – Secure Sockets Layer;
- HTTP – Hyper Text Transfer Protocol;
- HTTPS - Hyper Text Transfer Protocol Secure;
- CA – Certificate Authority, o entitate care se ocupă cu validarea certificatelor;
- PKI – Public Key Infrastructure, este un set de politici;
- Java – Limbajul de programare utilizat în implementarea clientului și serverului. Vom arăta cum putem utiliza limbajul Java pentru a folosi comunicații sigure utilizând cele două versiuni ale protocolului TLS;
- Wireshark – un tool utilizat pentru analizarea protocoalelor. Noi îl vom folosi pentru efectuarea de capturi ale traficului asupra comunicației dintre client și server;
- keytool – un tool specific Java folosit pentru a manipula și genera chei, keystore-uri, truststore-uri și certificate;
- Keystore – este un container de chei și certificate;
- Truststore – este un container de certificate.

Lucrarea este structurată după cum urmează:

- Capitolul 1 Introducere, prezentarea motivației proiectului.
- Capitolul 2 Sinteza arhitecturii și procedurile protocolului TLS.
- Capitolul 3 Folosirea certificatelor
- Capitolul 4 Studiul de caz
- Capitolul 5 Testare/Utilizare

Capitolul 1.1 Motivația lucrării

Transmiterea datelor personale este aproape inevitabilă atunci când accesăm internetul. Fie că încercăm să facem o plată pe internet precum plata facturilor sau că folosim o aplicație de comunicații cu alte persoane vrem ca datele transmise să nu poată fi citite de alte persoane pentru a ne proteja intimitatea. Astfel, folosirea de certificate SSL/TLS a devenit un obicei deoarece adaugă un nivel de Securitate și anume criptarea și autentificarea.

Limbajul Java este un limbaj de programare foarte popular în comunicațiile de date și de dezvoltare de aplicații software și prezintă o implementare ușoară a protocolului TLS.

CAPITOLUL 2. Sinteza arhitecturii și procedurile protocolului TLS

Capitolul 2.1 Arhitectura TLS

TLS este un protocol ce operează direct deasupra TCP-ului. Acesta este motivul pentru care aplicațiile de nivel înalt rămân aproape neschimbate atunci când securizăm conexiunea. Un bun exemplu este HTTPS care este identic cu HTTP pe deasupra stratului TLS.

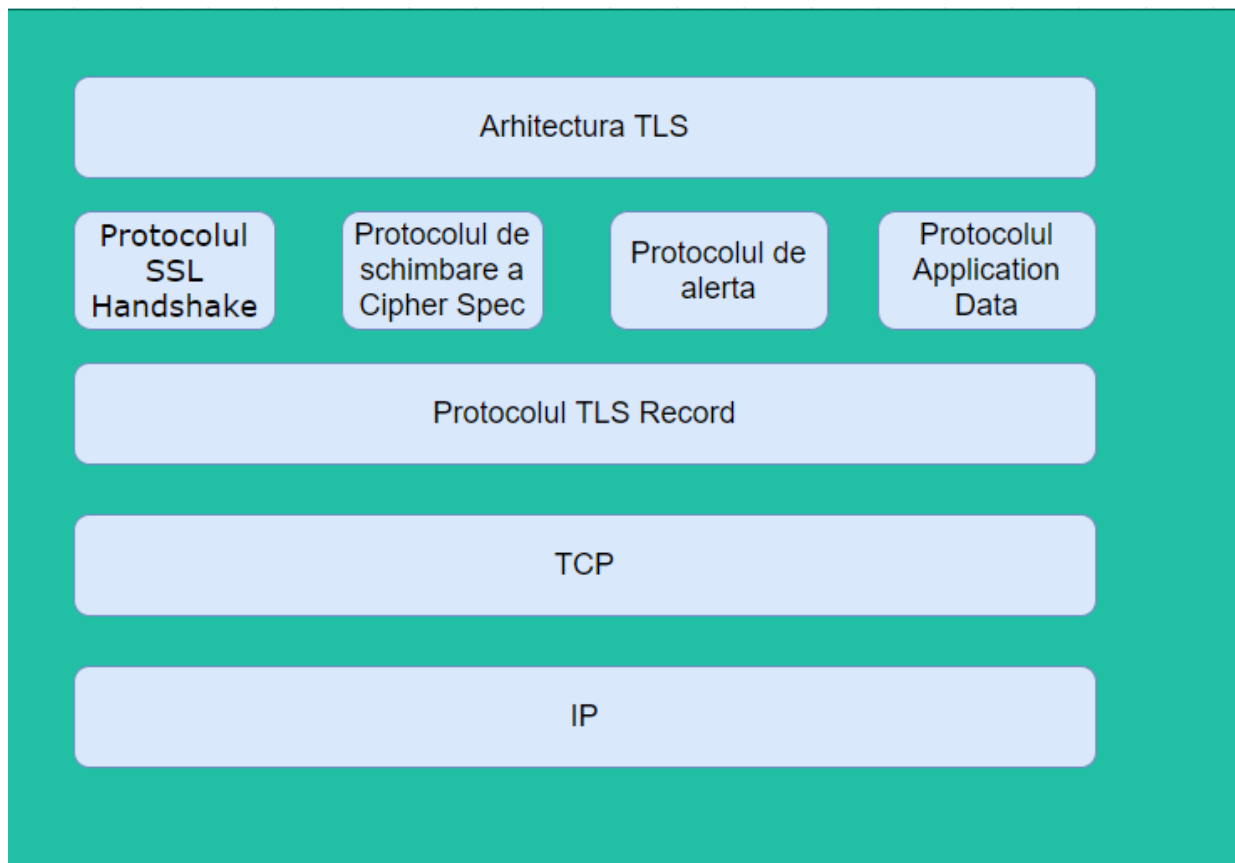


Figura 2.1 Arhitectura protocolului TLS

[1] Utilizări ale protocolului TLS:

- Canal securizat de la un capăt la altul (proces-proces).
- Oferă securitatea fluxului de date sau datagramelor, nu doar securitatea datagramelor oferită de straturile inferioare.
- Protocol unic de canal securizat standard pentru toți aplicații, în loc de securitate pentru fiecare aplicație (browser/server web, client/server de e-mail etc.).
- Trebuie configurat / activat pentru fiecare aplicație. Nu este complet transparent pentru aplicații și utilizatori.
- Nu protejează traficul de nivel inferior.
- Mai multă flexibilitate, dar și mai multe riscuri: aplicații care nu le protejează traficul. Ai nevoie de niște mijloace pentru a atenua aceste probleme la partea critică a clientului.

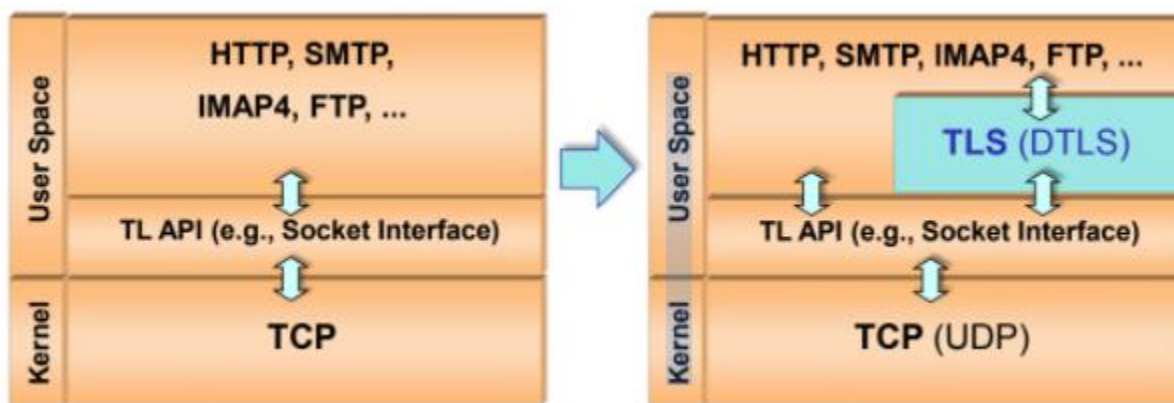


Figura 2. 2 Localizarea protocolului TLS între spațiul utilizatorului și Kernel

Preluat din cursul [“Security Protocols”, titular de curs Conf. Dr. Ing. Octavian Catrina)

TLS: Transport Layer Security.

- Rulează prin TCP.
- Comunicații securizate în flux de date.
- Cel mai utilizat protocol de canal securizat.
- Numit inițial SSL: Secure Socket Layer.

DTLS: Datagram TLS.

- TLS adaptat pentru a rula peste UDP.
- Comunicații de date sigure.

[1]

Cu o instalare corectă a protocolului putem să ne conectăm în siguranță la internet, atacatorii putând numai să vadă ip-urile și port-urile, cantitatea de informație care este transmisă, incipția și compresia folosite.

Acest protocol implică 2 entități, un server și un client. În acest caz, un server este o aplicație care așteaptă cereri de la o altă aplicație, numită client. Pe baza cererii clientului, serverul procesează cererea acestuia și trimite un răspuns. Un exemplu de server este site-ul web iar clientul este browser-ul care se conectează la site-ul web respectiv. Protocolul TLS se folosește de TCP pentru a oferi un serviciu de încredere. Acesta acționează ca un protocol cu dublu strat, protocolul TLS Record oferind serviciile de securitate de baza către alte protocoale de nivel înalt precum HTTP. Alte protocoale de nivel înalt sunt definite ca parte a SSL, și anume protocolul Alert, protocolul Handshake și protocolul Change CipherSpec.

Capitolul 2.2 Protocolul TLS Handshake

Protocolul Handshake este numele tehnic procesului dat care stabilește o conexiune HTTPS. Aproape toată munca este făcută în acest protocol. Principiul scop al acestui protocol este de a realiza procesele criptografice în vederea obținerii unei conexiuni sigure. Principalele procese întreprinse de acesta sunt verificarea autenticității certificatului prezentat, și verificarea asocierii cheii private cu certificatul.

Un amănunt important de menționat este acela că protocolul “Handshake” este o serie de pași în scopul obținerii următoarelor 3 obiective:

- Schimbul de capabilitati de criptare;
- Autentificarea certificatului TLS/SSL

- Schimbul sau generarea unei chei de sesiuni.

Înainte de începerea schimbului de date al aplicației prin TLS dintre client și server, tunelul criptat trebuie negociat: clientul și serverul trebuie să fie de acord cu versiunea protocolului TLS, să aleagă ciphersuite și să verifice certificatele, dacă este necesar. Din păcate, fiecare dintre acești pași necesită noi pachete dus-întors (Figura 2.3) între client și server, ceea ce adaugă latența de pornire la toate conexiunile TLS.

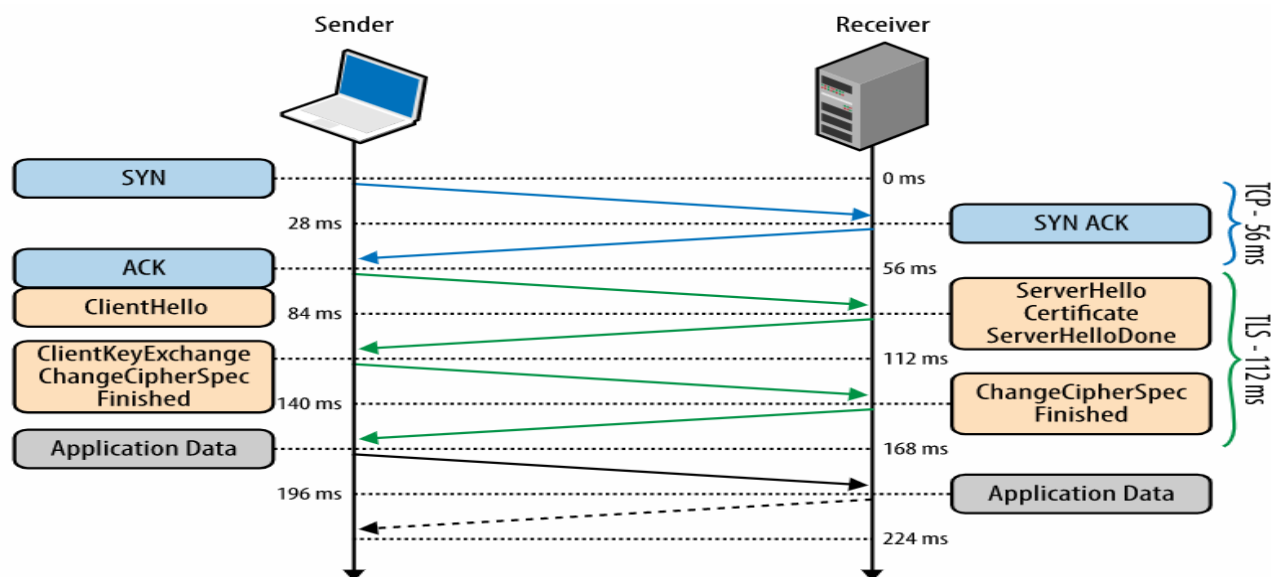


Figura 2.3 Schimbul de mesaje între emițător și receptor

[Preluat de pe pagina web: <https://hpbn.co/transport-layer-security-tls/>]

Voi observa și detalia fiecare mesaj transmis din figura de mai sus (figura 2.4) după cum urmează:

0 ms

TLS rulează printr-un transport de încredere (TCP), ceea ce înseamnă că trebuie să finalizăm mai întâi strângerea de mână TCP în trei direcții, care durează o singură călătorie dus-întors.

56 ms

Cu conexiunea TCP în loc, clientul trimite o serie de specificații în text simplu, cum ar fi versiunea protocolului TLS pe care îl rulează, lista ciphersuitelor acceptate și alte opțiuni TLS pe care ar putea dori să le folosească.

84 ms

Serverul alege versiunea protocolului TLS pentru o comunicare ulterioară, decide o serie de cifre din lista furnizată de client, atașează certificatul său și trimite răspunsul înapoi clientului. Opțional, serverul poate trimite, de asemenea, o cerere pentru certificatul clientului și parametrii pentru alte extensii TLS.

112 ms

Presupunând că ambele părți sunt capabile să negocieze o versiune și cifrare comune, iar clientul este mulțumit de certificatul furnizat de server, clientul inițiază fie RSA, fie schimbul de chei Diffie-Hellman, care este utilizat pentru a stabili cheia simetrică pentru sesiunea care urmează.

140 ms

Serverul procesează parametrii de schimb de chei trimiși de client, verifică integritatea mesajului prin verificarea MAC și returnează clientului un mesaj finalizat criptat.

168 ms

Clientul decriptează mesajul cu cheia simetrică negociată, verifică MAC-ul și dacă totul este bine, atunci tunelul este stabilit și datele aplicației pot fi acum trimise.

Capitolul 2.3 Protocolul TLS Record

Acest protocol este responsabil cu identificarea diferitelor tipuri de mesaje cât și cu verificarea integrității și asigurarea securității fiecărui mesaj.

Informația ajunge la protocolul “TLS Record”, care este împărțită în mai multe blocuri de biți, se adaugă un cod de autentificare a mesajului (MAC) și în final informația este criptată utilizând algoritmi negociati.

Capitolul 2.4 Protocolul TLS Alert

Protocolul de alertă este acolo pentru a permite transmiterea semnalelor între colegi. Aceste semnale sunt utilizate în principal pentru a informa peerul despre cauza unei defecțiuni a protocolului. Unele dintre aceste semnale sunt utilizate intern de protocol, iar protocolul aplicației nu trebuie să le facă față (de exemplu, GNUTLS_A_CLOSE_NOTIFY), iar altele se referă exclusiv la protocolul aplicației (de exemplu, GNUTLS_A_USER_CANCELLED). Un semnal de alertă include o indicație de nivel care poate fi fatală sau de avertizare (în TLS1.3 toate alertele sunt fatale). Alertele fatale încetează întotdeauna conexiunea curentă și împiedică re-negocierile viitoare folosind ID-ul sesiunii curente. Toate mesajele de alertă acceptate sunt rezumate în tabelul de mai jos.

Mesajele de alertă sunt protejate de protocolul de înregistrare, astfel informațiile incluse nu se scurg. Trebuie să aveți grijă deosebită ca informațiile de alertă să nu se scurgă către un posibil atacator, prin intermediul fișierelor jurnal publice etc.

Voi afișa mai jos câteva exemple de alarme precum urmează:

Tabel 1 Alarme generate de protocolul TLS “alert”

| Alarmă | ID | Descriere |
|-----------------------------|----|------------------------------------|
| GNUTLS_A_CLOSE_NOTIFY | 0 | Notificare de închidere |
| GNUTLS_A_UNEXPECTED_MESSAGE | 10 | Mesaj neașteptat |
| GNUTLS_A_BAD_RECORD_MAC | 20 | Înregistrare greșită a adresei MAC |
| GNUTLS_A_DECRYPTION_FAILED | 21 | Descriere eșuată |

Capitolul 2.5 Protocolul Change Cipher Spec

Mesajele ChangeCipherSpec sunt folosite în SSL pentru a indica faptul că comunicarea este transferată de la necriptată la criptată.

Acest mesaj informează că, următoarele date vor fi criptate cu secretul partajat sau, cu alte cuvinte, puteți spune că, acest mesaj este folosit pentru a spune celeilalte părți (Server și Client), că cheia secretă negociată și suita de cifrare vor fi utilizate pentru comunicarea curentă acum.

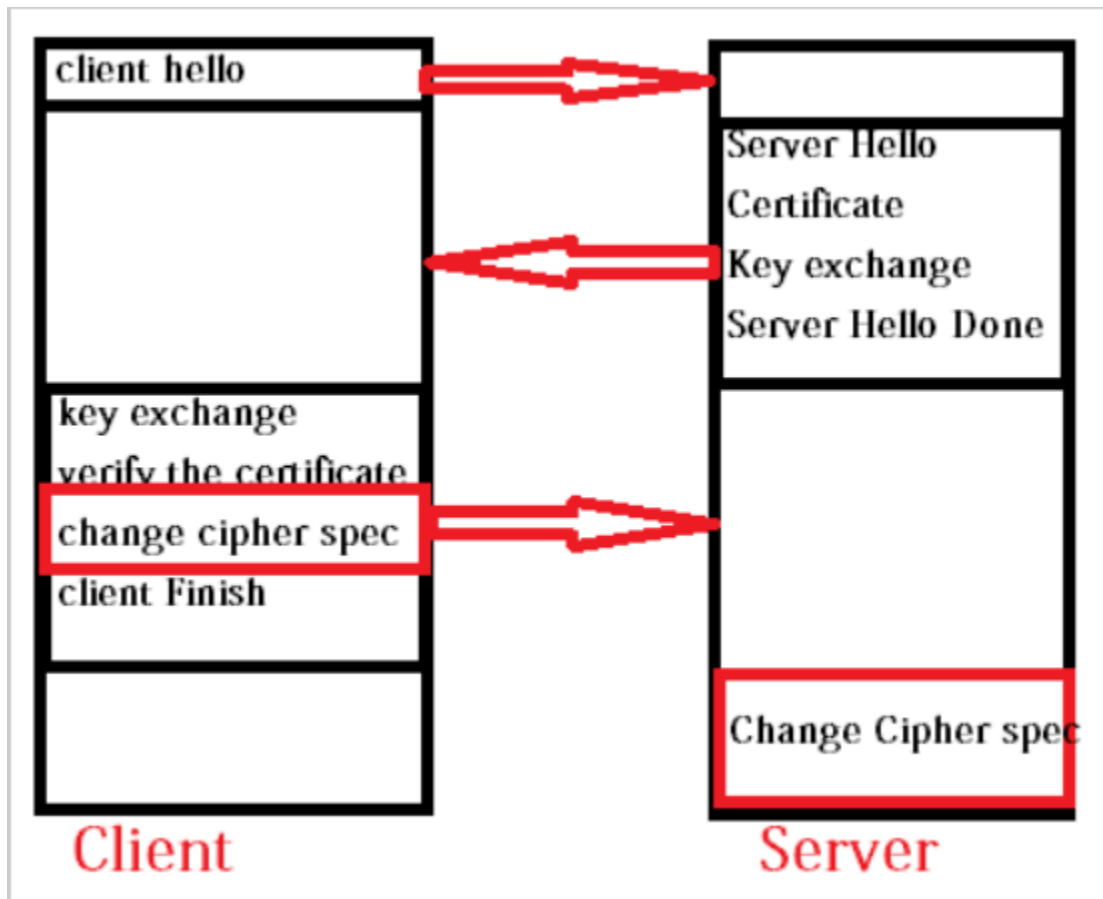


Figura 2. 4 Schimbul de cifru în modelul client-server

[Preluat de pe pagina web: <https://www.slashroot.in/changeipherspec-protocol-ssl>]

Acest mesaj este trimis atât de server, cât și de client, pentru a se anunța reciproc punerea în folosință ceea ce a fost negociat.

Este de precizat că acest mesaj are doar un singur octet.

Imediat după verificarea schimbului de chei și a certificatului, clientul trimite serverului acest mesaj specific specificației cifrării. Și la primirea mesajului de schimb de chei, serverul trimite, de asemenea, înapoi un mesaj specific specificației cifrului.

Capitolul 2.6 Autentificarea

Autentificarea este o formă simplă de verificare a identității. În general, comunicațiile pe internet sunt întreținute de un client, spre exemplu un browser web și serverul, site-ul web accesat

De notat este faptul că autentificarea clientului bazată pe certificate face parte din protocolul "TLS". În acest caz, client-ul semnează digital o bucată de date generată în mod aleatoriu și trimite această bucată de date și certificatul către rețea. La final, serverul confirmă validitatea certificatului și semnătura acestuia.

Următorii pași vor explica cum este autentificat un client de către server:

- Aplicația de client manageriază o bază de date care conține din chei private publicate în toate certificatele emise pentru acel client. La prima accesare a bazei de date va fi nevoie de introducerea parolei.
- După aceasta, clientul deschide baza de date și întoarce cheia privată asociată certificatului utilizatorului și se folosește această cheie pentru a semna date care sunt generate automat atât de la intrarea serverului cât și de la intrarea client-ului. Datele generate automat și certificatul utilizatorului sunt amândouă trimise către rețea de către client.
- Pentru a face autentificarea utilizatorului, serverul folosește atât datele semnate cât și certificatul.

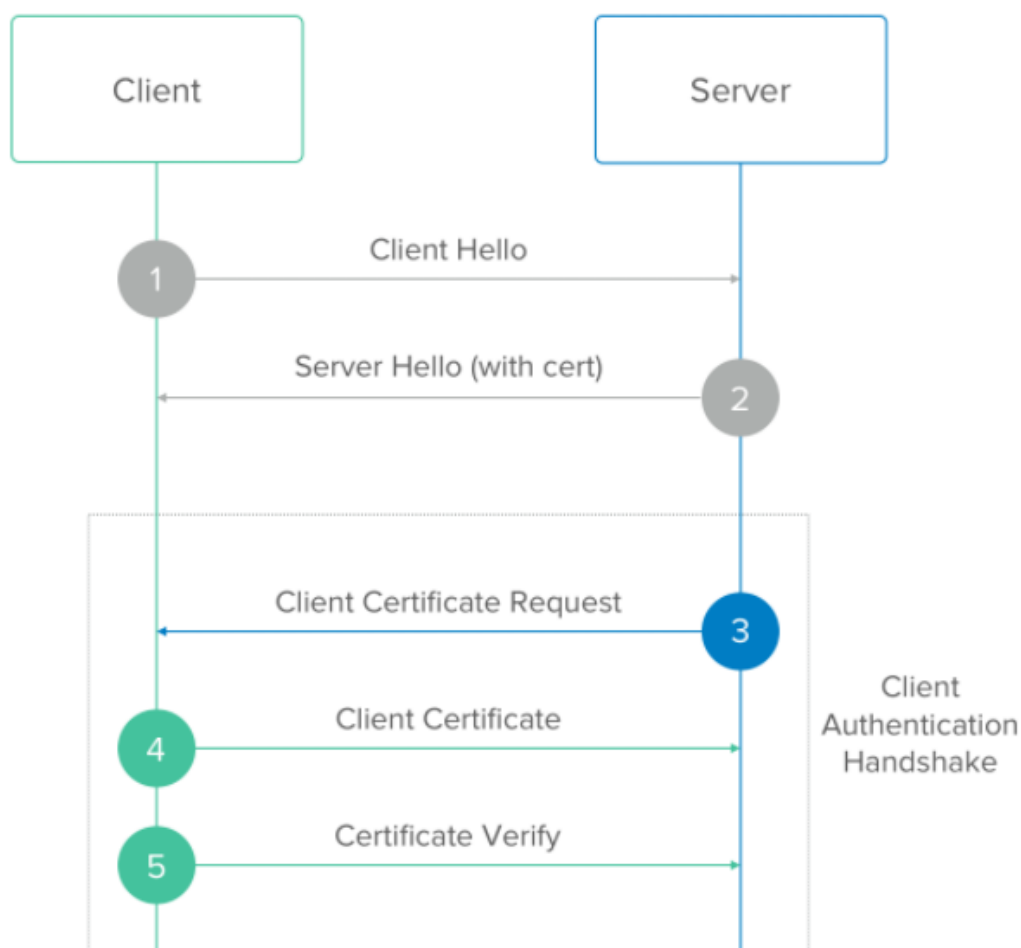


Figura 2. 5 Procesul de autentificare al protocolului TLS

[Preluat de pe pagina web: <https://developer.okta.com/blog/2015/12/02/tls-client-authentication-for-services>]

Capitolul 2.7 Criptarea

Criptarea este un proces care codifică un mesaj sau un fișier, astfel încât să poată fi citit doar de anumite persoane. Criptarea folosește un algoritm pentru a codifica sau cripta date și apoi folosește o cheie pentru ca partea destinatară să dezarhiveze sau să decripteze informațiile. Mesajul conținut într-un mesaj criptat este denumit text simplu. În forma sa criptată, ilizibilă, este denumit text cifrat.

Criptarea utilizează algoritmi pentru a amesteca informațiile. Acesta este apoi transmis părții care primește, care este capabilă să decodeze mesajul cu o cheie. Există mai multe tipuri de algoritmi, care implică toate modalități diferite de a codifica și apoi a decripta informațiile.

Cheile sunt de obicei generate cu generatoare de numere aleatorii sau algoritmi de computer care imită generatoare de numere aleatorii. O modalitate mai complexă prin care computerele pot crea chei este folosirea mișcării mouse-ului utilizatorului pentru a crea seminte unice. Sistemele moderne care păstrează secretul înainte implică generarea unei chei proaspete pentru fiecare sesiune, pentru a adăuga un alt nivel de securitate.

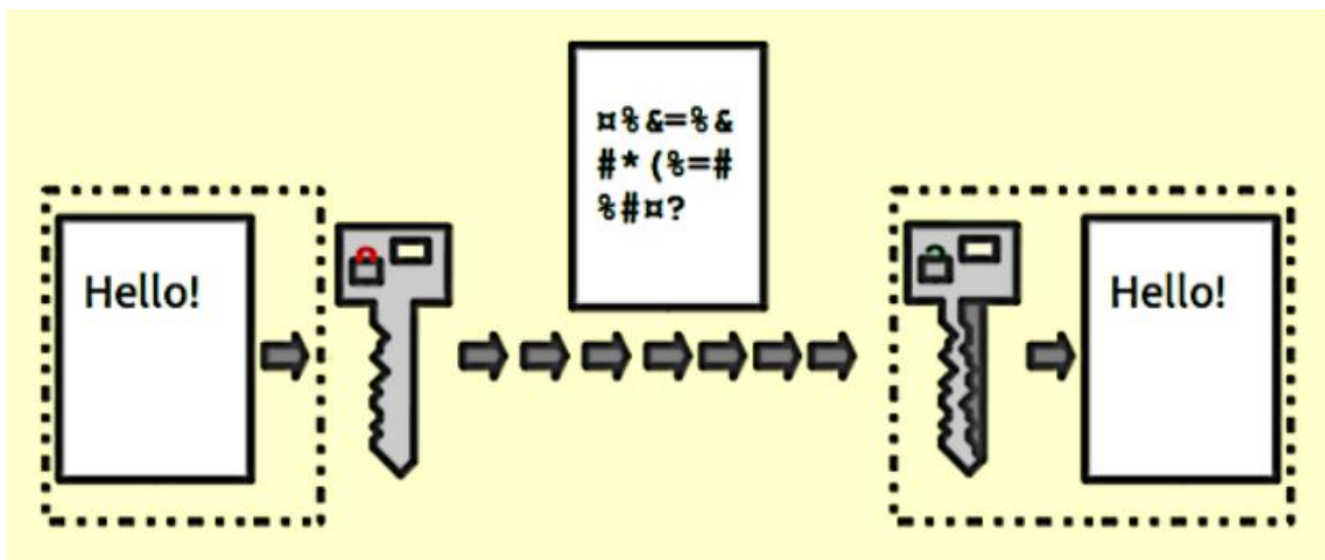


Figura 2.6 Procesul de criptare între două entități la nivel minimal

[Preluată de pe pagina web: <https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537>]

Capitolul 2.8 Criptarea cu cheie publică

Criptarea cu cheie publică mai este numită și criptare asimetrică. Aceasta folosește două chei diferite, una publică și una privată. Cheia publică este accesibilă oricui, iar cea privată rămâne la utilizator. Aceste două chei nu sunt identice, deoarece fiecare cheie este folosită în scop diferit. Cheia publică este folosită pentru a cripta mesaje, iar cheia privată este folosită pentru a le decripta.

La rândul ei, criptarea cu cheie publică folosește anumiți algoritmi de criptare precum “RSA & DSA” pentru crearea de chei publice și private.

De notat este faptul că din punct de vedere computațional, generarea de chei publice și private și criptarea cu aceste chei este ușor de făcut, însă este aproape imposibil obținerea cheii private din cheia publică.

În plus, criptarea cu cheie publică ajută în autentificarea și în transferul de chei. De exemplu, de fiecare dată când un utilizator vizitează un site web acesta va trimite un mesaj “clientHello” care conține o listă de suite cipher pe care le suportă apoi toate acestea sunt criptate cu cheia publică a serverului. Serverul se folosește de cheia sa privată pentru a decripta mesajul “clientHello” și răspunde cu un mesaj “serverHello” cu certificatul acestuia, un cipher ales și cheia. După ce mesajul “serverHello” este primit, clientul și serverul încep comunicația folosind cheia cu criptare simetrică schimbată.

Cheile de sesiuni se schimbă des, și de multe ori este posibil ca o cheie de sesiune diferită să fie folosită pentru fiecare mesaj.

Capitolul 2.9 – Criptarea simetrică sau criptarea cu cheie privată

Așa cum am menționat anterior, pentru a confirma autenticitatea serverului, procesul de autentificare care se face prin intermediul cheii publice, criptării asimetrice și prin semnăturile digitale. Însă, în momentul în care autentificarea serverului este îndeplinită, pentru restul sesiunii se folosește criptarea simetrică.

Altfel spus, criptarea asimetrică este folosită în momentul procesului de “handshake” ca o metodă de verificare, unde browser-ul și site-ul web negociază o comunicație criptată și fac schimb de sesiuni de chei. Sesiunile de chei folosesc criptarea simetrică pe tot parcursul întregului transfer de informații.

Criptarea simetrică este o metodă de criptare care se folosește de o cheie secretă pentru a cifra și descifra informația.

Cheia secretă este în general formată din numere și cuvinte din diferite alfabete.

Câteva exemple de algoritmi de criptare simetrică sunt: AES, DES, RC4, RC5 și RC6, dintre acestea cele mai populare fiind: AES-128, AES-192, AES-256.

Capitolul 2.10 – Algoritmi, Ciphers si Cipher suites

Un cifru este un tip de algoritm care prezintă secvența de pași care trebuie urmați pentru a îndeplini o funcție criptografică, cum ar fi criptarea sau decriptarea. Pentru criptarea SSL, acțiunile sunt de fapt efectuate de chei, dar cifrele oferă regulile criptosistemului și ordinea în care cheile îndeplinesc funcțiile criptografice necesare.

Când este stabilită o conexiune HTTPS, mai multe dintre aceste cifre funcționează în tandem. Aceasta este cunoscută sub numele de Cipher Suite. În esență, o colecție de diferite cifre care îndeplinesc diverse funcții criptografice, cum ar fi generarea cheilor și autentificarea, și oferă ordinea în care ar trebui să apară.

Când un browser web încearcă să se conecteze la serverul pe care este găzduit site-ul dvs., negociază ce suită de cifrare ar trebui utilizată pentru a stabili o conexiune sigură în timpul unui proces cunoscut sub numele de strângere de mână SSL.

Anatomia unei suite de cifrare depinde de protocoalele TLS activate atât pe client, cât și pe server. Scurt pentru Transport Layer Security, TLS este protocolul care stă la baza modului în care funcționează

certIFICATELE SSL. Cea mai recentă versiune a protocolului este 1.3, dar versiunea anterioară, 1.2, este încă utilizată pe scară largă. În timp ce TLS 1.2 este încă incredibil de sigur, 1.3 a adus unele îmbunătățiri și mai puțin expuse riscului anumitor vulnerabilități. O mare diferență este numărul de suite cifrate pe care le acceptă. TLS 1.2 are 37 de cifre, în timp ce 1.3 are doar cinci. În 1.2, o suită de cifrare conține patru cifre, în timp ce 1.3 are doar două. Cu 1.2, unele suite de cifrare sunt mai sigure decât altele.

Iată un exemplu de suită de cifrare acceptată de TLS 1.2:

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

Iată un exemplu de suită de cifrare acceptată de TLS 1.3:

TLS_AES_256_GCM_SHA384

Capitolul 2.11 – Infrastructura cheii publice PKI

“PKI” reprezintă un set de reguli, proceduri și politici în scopul de a crea, distribui, inmagazina, folosi și revoca certificate și de administrarea criptărilor cu cheie publică.

Un “PKI” conține următoarele componente:

- Politica certificatului: cererile de securitate folosite pentru ierarhizare și structurare ale mediului PKI. De asemenea, acestea se ocupă și administrarea cheilor, revocarea certificatelor și formatul acestora;
- Root Certificate Authority: este principala sursă de încredere și este responsabilă cu autentificarea identității mediului PKI;
- CA subordonat sau intermediar: acesta este certificat printr-un root CA pentru a fi folosit în conformitate cu politica certificatului;
- Baza de date a certificatului: o bază de date care stochează înregistrările unui certificat;
- Serviciile de revocare: servere care postează CRL-uri sau OCSP-uri;
- Certificatul digital: este un tip de identitate digitală care este încorporat într-un dispozitiv care oferă securitate și autentificare între servere și dispozitivele cu care comunică în timp ce permit accesul la resurse. Este în general eliberat de CA subordonat.

Capitolul 2.12 Protocolul TLS 1.3 în comparație cu TLS 1.2

Cea mai mare îmbunătățire a noii versiuni de protocol a fost la nivelul protocolului de handshake, reușind să înjumătățească pașii necesari autentificării.

Etaple detaliate ale handshake-ului în versiunea TLS 1.3 sunt expuse în Tabelul 2.1

Tabel 2 Etapele Handshake-ului al TLS 1.3

| Nr. etapa | Mesajul | Acțiunea întreprinsă |
|-----------|---|--|
| 1 | <ul style="list-style-type: none"> • Client Hello • Cipher Suite-urile stabilite • Ghicirea cheii • Stabilirea protocolului • Împărțirea cheilor | Aici se inițiază mesajul “clientHello”, diferența constând în faptul că aici clientul de asemenea trimite cipher suite-urile pe care le suporta în timp ce ghicește cheia folosită de protocol. La final, clientul trimite cheia să indiferent de protocolul stabilit. |
| 2 | <ul style="list-style-type: none"> • Server Hello • Stabilirea cheii folosite de protocol • Împărțirea cheilor • Serverul a terminat | Serverul raspunde cu protocolul ales, iar mesajul “serverHello” include cheia sa pe post de certificat și trimite mesajul “serverFinished” |
| 3 | <ul style="list-style-type: none"> • Verifica certificatul • Generează cheile • Clientul a terminat | In cele din urmă, clientul verifică certificatul serverului, generează cheia sa si trimite mesajul “Client Finished” si criptarea datelor poate fi pornită |

Din punct de vedere al securității, noua versiune a eliminat o suită de algoritmi predispuși vulnerabilitatilor și a adoptat noi algoritmi pentru care nu exista vulnerabilități existente.

Algoritmii eliminați în noua versiune:

- Schimbul de chei RSA
- RC4 Stream Cipher
- CBC (Block) Mode Ciphers
- Algoritmul MD5
- Functia Has SHA-1
- DES
- 3-DES

Capitolul 3 Folosirea certificatelor

Capitolul 3.1 Certificate si CA

Protocolul “TLS” are nevoie pentru a începe autentificare și procesul de “handshake” să prezinte partenerului la conexiune un certificat cu câteva informații despre cine este acesta și cheia secretă asociată acestui certificat. Informațiile pe care le conține un certificat sunt informații destinate identificării precum domeniul folosit și țara de unde a fost generat. Informațiile folosite în certificate sunt folosite pentru a fi verificate de un CA. Un CA este o entitate care verifică datele introduse în certificate și dacă acestea se potrivesc cu cele din certificat, atunci acesta îl semnează.

Este de remarcat că CA sunt entități de încredere deoarece oricine poate semna certificatul, însă nu toată lumea este de încredere. Când cineva semnează acel certificat acea semnătură rămâne.

Dacă considerăm că certificatul prezentat este de încredere atunci acesta se poate pune într-un “truststore”. Aceasta este o bază de date care se ocupă cu validarea internă a certificatelor. Dacă se găsește semnătura digitală a certificatului în această bază de date atunci el este de încredere.

Capitolul 3.2 Generarea de containere de chei in format JKS

Un container de chei ne permite să ne înmagazinăm certificatele și cheile. Formatul JKS este formatul pe care îl utilizează keytool și Java.

Pentru a folosi “keytool” se folosește un command-line.

Pentru a se afișa mesajul de “help” se poate utiliza:

keytool -?

```
C:\Users\BogdanHasan>keytool -?
Key and Certificate Management Tool

Commands:
-certreq          Generates a certificate request
-changealias      Changes an entry's alias
-delete           Deletes an entry
-exportcert       Exports certificate
-genkeypair       Generates a key pair
-genseckey        Generates a secret key
-gencert          Generates certificate from a certificate request
-importcert       Imports a certificate or a certificate chain
-importpass       Imports a password
-importkeystore   Imports one or all entries from another keystore
-keypasswd        Changes the key password of an entry
-list             Lists entries in a keystore
-printcert        Prints the content of a certificate
-printcertreq     Prints the content of a certificate request
-printcrl         Prints the content of a CRL file
-storepasswd      Changes the store password of a keystore

Use "keytool -?, -h, or --help" for this help message
Use "keytool -command_name --help" for usage of command_name.
Use the -conf <url> option to specify a pre-configured options file.

C:\Users\BogdanHasan>
```

Figura 3. 1 Mesajul help al keytool

Pentru generarea unui keystore rulăm următoarea comandă:

keytool -genkey -alias tls_analysis -keyalg RSA -keystore numele_keystore-ului.jks
care va face urmatoarele operații:

- -genkey – genereaza o pereche de chei;
- -alias – Atribuie un alias cheii;
- -keyalg – Algoritmul folosit in generarea cheii;
- -keystore – Numele keystore-ului dorit.

Dupa executarea comenzii ni se va cere să atribuim o parola pentru keystore.

```
C:\Bogdan\Facultate\Licenta\credentiale>keytool -genkey -alias tls_analysis -keyalg RSA -keystore "C:\Bogdan\Facultate\Licenta\credentiale\tls_analysis.jks"  
Enter keystore password:  
Re-enter new password:
```

Figura 3. 2 Parola keystore

În urma executării comenzii ni se vor cere în continuare datele de identificare.

```
What is your first and last name?  
[Unknown]: Bogdan Hasan  
What is the name of your organizational unit?  
[Unknown]: IT  
What is the name of your organization?  
[Unknown]: Facultatea ETTI  
What is the name of your City or Locality?  
[Unknown]: Bucuresti  
What is the name of your State or Province?  
[Unknown]: Romania  
What is the two-letter country code for this unit?  
[Unknown]: RO
```

Figura 3. 3 Completarea chestionarului keystore-ului

În final, ni se va cere să confirmăm datele introduse și o dată confirmate certificatul va fi generat

```
Is CN=Bogdan Hasan, OU=IT, O=Facultatea ETTI, L=Bucuresti, ST=Romania, C=RO correct?  
[no]: yes
```

Figura 3. 4 Confirmarea informațiilor introduse în keystore

Capitolul 3.3 Generarea de certificate folosind “OpenSSL”

O altă opțiune pentru stocarea, manipularea și crearea de chei criptografice este “OpenSSL”. Deși certificatele generate de acesta nu sunt direct compatibile cu limbajul Java, ele pot fi însă convertite în fișiere de tip “jks”.

Pentru a afișa mesajul de “help” se poate utiliza comanda:

```
openssl help
```

Pentru a obține un container de chei:

```
openssl req -newkey rsa:2048 -x509 -keyout openssl.pem -out cacert.pem -days 3650
```

```
C:\Bogdan\Facultate\Licenta\openssl>openssl req -newkey rsa:2048 -x509 -keyout openssl.pem -out cacert.pem -days 3650
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'openssl.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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) [XX]:RO
State or Province Name (full name) []:Romania
Locality Name (eg, city) [Default City]:Bucharest
Organization Name (eg, company) [Default Company Ltd]:IT
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:localhost
```

Figura 3. 5 Generare certificat openssl

Parametrii folositi:

- Req – generează o cerere de crearea a certificatului;
- -newkey:rsa:2048 – algoritmul pentru generarea cheii;
- -x509 – generează un certificat de tip “self signed”;
- -keyout - fișierul care să scrie cheia;
- -out – fișierul în care să scrie;
- -days – numărul de zile de valabilitate

CAPITOLUL 4. STUDIUL DE CAZ

PREZENTAREA GENERALA

Proiectul își propune să implementarea unor aplicații scrise în limbajul de programare Java, care să poată comunica în mod securizat între ele prin folosirea protocolului “TLS”. Astfel, voi evidenția configurarea protocolului în limbajul Java, alegerea folosirii diferitelor versiuni ale protocolului, diferitelor funcționalități care pot fi utilizate atunci când se dorește comunicarea pe internet și cum se poate analiza protocolul folosind capturi de trafic. De asemenea, voi arăta cum se lucrează cu certificate în interiorul unei aplicații.

Pentru a pune în evidența mai bine necesitatea securizării comunicațiilor pe internet și diferitele configurații și implementări ale protocolului la nivel de aplicație, voi crea trei perechi de aplicații client și server care deservească diferite cerințe.

Scrierea și testarea aplicațiilor s-a realizat folosind un “IDE”, IntelliJ, versiunea distribuției de Java folosite a fost 11, iar sistemul de operare a fost “Windows 10 64 bits”.

Capturile de trafic au fost făcute cu ajutorul aplicației “Wireshark”.

Capitolul 4.1 Descrierea soluțiilor propuse

Soluțiile propuse în acest proiect au în vedere crearea unor aplicații de test care să poată permite analiza atât comunicația criptată cât și pe cea necriptată. Astfel, voi începe prin a crea două aplicații client și server care să folosească o comunicație nesigură atunci când cele două comunică pe internet. În urma schimbului de mesaje, voi analiza traficul de date folosind utilitarul Wireshark simulând un caz real în care un atacator încearcă să fure date transmise pe internet. În continuare, pe baza celor două aplicații create anterior, am adăugat comunicații criptate prin protocolul TLS. Scopul programelor în acest caz este de a demonstra implementarea unei comunicații sigure în Java. Voi prezenta cele două versiuni de protocol studiate TLS 1.3 și TLS 1.2, atât prin demonstrarea în cod a alegerii versiunii, cât și prin traficul analizat cu Wireshark. În cele din urmă dorim să facem o comparație între cele două versiuni pentru a putea decide care este mai sigură și care sunt diferențele. Aplicațiile descrise anterior permit comunicarea între ele, prin schimbul unor mesaje text pe care utilizatorul le va introduce de la tastatură.

Un alt caz foarte des întâlnit al protocolului este cazul în care se folosește protocolul “HTTP” împreună cu protocolul “TLS”, devenind astfel “HTTPS”. În ultima parte a proiectului dorim să înțelegem diferența dintre cele două și cum se poate implementa un server în Java care înțelege cele două protocoale. De asemenea, în acest caz, voi prezenta cum putem administra certificatele în care avem încredere, în cazul în care dorim să filtrăm anumite conexiuni care nu sunt sigure.

Figura de mai jos are rolul de a prezenta cazul în care două aplicații comunica în mod nesecurizat, iar o a treia aplicație încearcă să intercepteze informațiile trimise.

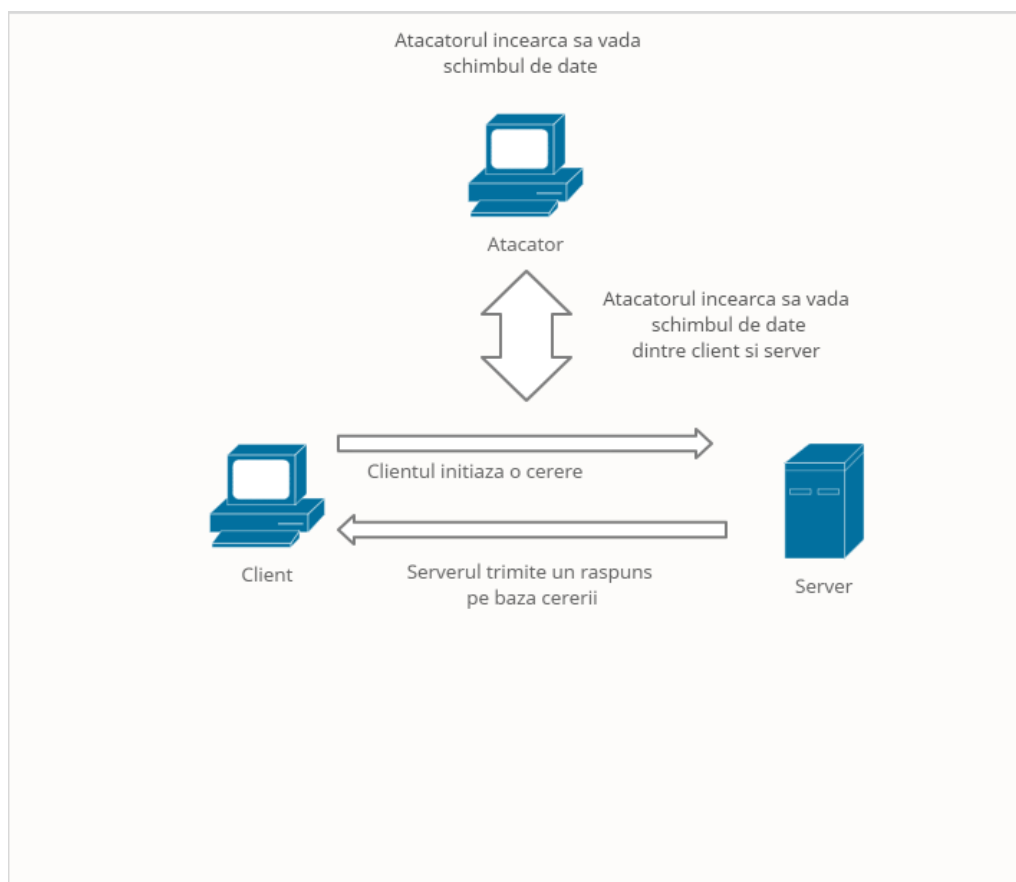


Figura 4. 1 Comunicația între client și server fără criptare

În cele din urmă, cele două aplicații vor putea comunica folosind criptarea datelor, iar atacatorul va avea acces numai la mesajele criptate.

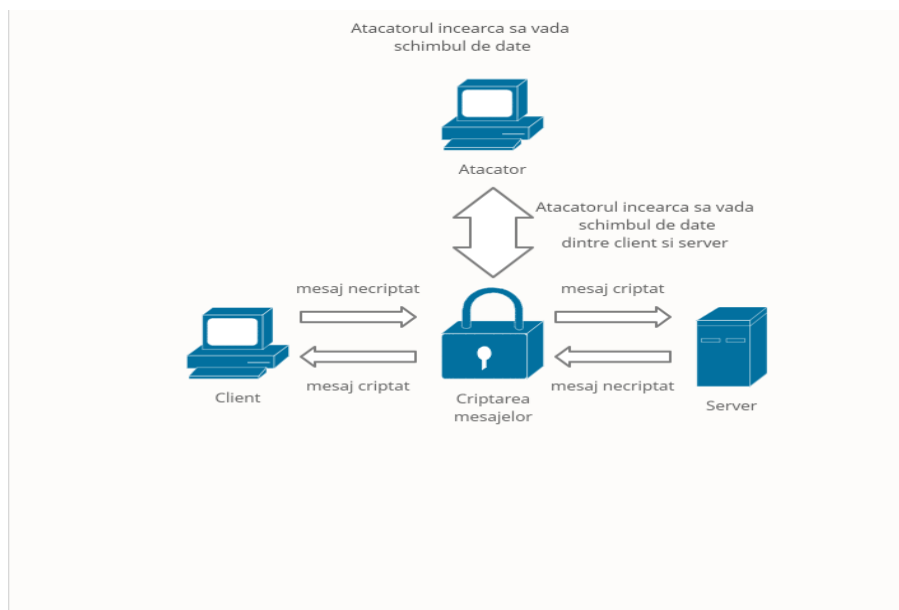


Figura 4. 2 Comunicația între client și server folosind algoritm de criptare

Prima pereche de aplicații client și server au scopul de a arată cum putem crea două aplicații care să schimbe mesaje între ele și scoaterea în evidență a pericolelor comunicațiilor necripte.

Aplicația EchoServer așteaptă o conexiune de la un client prin crearea unui obiect de tip “Socket” care să permită conexiuni pe o anumită adresă și pe un anumit port. După conectarea clientului acesta trebuie să poată să primească un mesaj de la client și să îl afișeze.

Aplicația EchoClient inițiază o cerere către o adresă și un port, folosind tot un obiect de tip “socket”. În urma conexiunii, aceasta citește un mesaj introdus de utilizator pe care să îl trimită mai departe către server.

Pentru a demonstra vulnerabilitățile unei astfel de comunicații, am analizat capturile de trafic folosind “Wireshark”.

Următoarea pereche de aplicații va demonstra implementarea unui mod securizat de a comunica cu serverul.

Funcționalitățile aplicațiilor Server și EchoServer, EchoSSLClient și Client sunt asemănătoare în sensul că amândouă primesc de la client un mesaj pe care serverul să îl primească, dar folosesc protocoale de comunicații în mod sigur și îi oferă posibilitatea utilizatorului de a face configurări în modul în care să pornească aplicațiile.

Aplicația Server încarcă certificatul folosit cu parola asociată acestuia în proprietățile de sistem ale limbajului pentru a le prezenta viitoarelor conexiuni. Aceasta așteaptă cererea de conexiuni pe o anumită adresă și la un anumit port, introdus de utilizator, folosindu-se de această dată “SSLSockets”, obiectele de tip “Sockets” care asigură criptarea conexiunii cu protocolului TLS. În continuare utilizatorul va putea alege cu ce variantă de protocol să pornească aplicația.

Aplicația EchoSSLClient inițiază o conexiune sigură către un server, adresa și portul la care se va iniția conexiunea vor fi introduse de utilizator. O altă configurare disponibilă utilizatorului este posibilitatea logării mesajului, și în cazul în care aceasta este activată va putea alege dintr-o serie de logări disponibile pe care la va alege utilizatorul.

Ca și în cazul precedent, aplicația trebuie să încarce în proprietățile de sistem certificatele folosite și să inițieze o cerere unui server folosind și de această dată “SSLSockets”.

Ca și în cazul precedent, vom analiza capturile de trafic pentru ambele variante de protocol și le vom compara cu capturile de ecran obținute în cazul aplicației cu comunicații nesigure.

Ultima pereche de aplicații, are ca scop demonstrarea utilizării protocolului “TLS” împreună cu protocolul “HTTP” folosind tehnologiile prezentate anterior. Conține trei aplicații, dintre care două sunt pentru pornirea serverului și o aplicație client care să se conecteze la server. Este nevoie de două aplicații de server deoarece o aplicație doar așteaptă conexiuni pe care le transmite mai departe celeilalte aplicații, astfel reușindu-se o comunicație multiplă însemnând că putem primi mai multe conexiuni simultan.

Serverul așteaptă o cerere de tip “GET” și acesta trebuie să întoarcă clientului biții fișierului care a venit în cerere.

Clientul inițiază cererea și apoi afișează conținutul fișierului.

Capitolul 4.2 Aplicația EchoServer

Principalul obiectiv al acestei aplicații este de a demonstra implementarea unui server simplu în Java care să poată primi și afișa în consolă mesajele trimise de un client cu care acesta este conectat. Astfel, vom putea vedea de ce această comunicație nu este sigură din punct de vedere al securității cibernetice capturând traficul de pe rețeaua locală. În urma schimbului de mesaje dintre client și server captura ne va arăta în text clar informațiile schimbate de aceștia.

Aplicația începe prin importarea bibliotecilor folosite:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
```

În continuare, voi defini clasa, setez variabila “port” care va fi folosită pentru a asculta conexiuni care, în cazul acestei aplicații, are valoarea de 9000.

```
public class EchoServer {
    private final static int port = 9000;
```

Am definit clasa main care începe printr-o confirmare și anume că aplicația a pornit și ascultă portul setat prin afișarea unui mesaj în consolă.

```
public static void main(String[] args) {
    System.out.println("Server-ul porneste...");
    System.out.println("Ascultam pe port-ul: " + port);
```

În interiorul acestei clase voi folosi o sintaxă de tip “try” și “catch”, deoarece unele din funcțiile apelate vor “arunca” excepții care trebuie “prinse”.

Urmează să creez obiectul de tip “ServerSocket” cu portul definit anterior și care ascultă după conexiuni. Această clasă, în cele din urmă, trebuie să întoarcă sesiunea clientului printr-un obiect de tip “Socket” și apoi se va afișa un mesaj în consolă cu confirmarea conexiunii.

```
try {
    ServerSocket serverSocket = new ServerSocket(port);
    Socket client = serverSocket.accept();

    System.out.println("Client conectat!");
```

Se va obține mesajul de la client de pe stream-ul de intrare și vom realiza un obiect care va scrie pe stream-ul de ieșire și care va afișa mesajul în consolă

```
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        PrintWriter output = new PrintWriter(client.getOutputStream(), true);
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println("De la Client: " + line);
            output.println(line);
        }
```

În cele din urmă, închidem sintaxa try catch, prinzând excepțiile posibile.

```
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Capitolul 4.3 Aplicatia EchoClient

Voi importa următoarele biblioteci folosite în proiect pentru :

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
```

Am definit două variabile de tip `int` și `String` care să conțină valoarea portului la care se conectează și adresa acestuia.

```
public class EchoClient {

    private final static int port = 9000;
    private final static String host = "localhost";
```

În funcția `main`, începem un bloc `try catch` în și creăm sesiunea către server.

```
public static void main(String[] args) {

    try {
        Socket socket = new Socket(host, port);
```

În continuare creăm obiectele care vor primi un mesaj de pe stream-ul de ieșire și care vor scrie pe stream-ul de intrare urmat de afisarea mesajului de confirmare al conexiunii.

```
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
BufferedReader buffer = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
System.out.println("Conectat la server cu succes!");
```

Pentru a citi mesaje din consolă, voi crea un obiect de tip `Scanner`:

```
Scanner scanner = new Scanner(System.in);
```

Cât timp sesiunea este pornită, aplicația vă aștepta un mesaj introdus din consolă și verifică dacă mesajul este "exit" caz în care aplicația se va termina.

```
while(true) {
    System.out.println("Spune ceva: ");
    String input = scanner.nextLine();
    if("exit".equalsIgnoreCase(input)) {
        break;
    }
}
```

Voi trimite mesajul către server și voi captura posibilele excepții apărute.


```

        out.println(input);
        String response = buffer.readLine();

    }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Capitolul 4.4 Aplicatia Server

Importăm bibliotecile folosite

```

import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.*;
import java.util.Scanner;

```

Aplicația trebuie să poată permite utilizatorului să configureze modul în care să pornească serverul în urma introducerii unor date de la tastatură. Pentru aceasta voi folosi un obiect de tip Scanner. Opțiunile de configurare pe care utilizatorul le poate alege sunt portul pe care să asculte și versiunea de protocol aleasă, pe care o vom stoca într-o variabilă de tip String. În consolă se vor loga diferite mesaje care să informeze utilizatorul în legătură cu starea programului.

Voi crea obiectul Scanner pentru a citi din consolă și se vor citi datele de la tastatură.

```

public class Server {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int port;
        String tls_version = "";

        System.out.print("Introdu port-ul pe care asculta server-ul: ");
        port = scanner.nextInt();
        scanner.nextLine();

        System.out.println("Server-ul asculta pe port-ul: " + port);

        System.out.print("Ce versiune de protocol folosim (TLSv1.2, TLSv1.3): ");
        tls_version = scanner.nextLine();
    }
}

```

Pentru a putea începe implementarea protocolului trebuie ca serverul și clientul să se poată identifica unul pe celălalt. Înainte de a începe procesul de "handshake", se va face întâi prezentarea certificatelor. Astfel, mă voi folosi de certificatele create anterior în capitolul 3 pentru a încărca în aplicație un certificat.

Încărcarea certificatelor se face utilizând proprietățile de sistem din Java, unde voi seta calea containerelor noastre de chei și certificate și parolele acestora.

Este important de menționat faptul că, în cazul aplicației de față, același container de chei joacă și rolul de container de certificate. Setarea unui container de certificate este utilă atunci când este dorită filtrarea conexiunilor și verificând dacă entitatea care a semnat certificatul respectiv face parte din entitățile pe care le considerăm de încredere în containerul de certificate.

```
System.setProperty("javax.net.ssl.keyStore",
"src/credentiale/tls_analysis.jks");
System.setProperty("javax.net.ssl.keyStorePassword", "tlsanalysis");
System.setProperty("javax.net.ssl.trustStore",
"src/credentiale/tls_analysis.jks");
System.setProperty("javax.net.ssl.trustStorePassword", "tlsanalysis");
```

Protocolul TLS este fix deasupra protocolului TCP, însemnând că, în cazul unei aplicații care dorește să comunice pe internet, criptarea cu TLS se va face la nivel de "Sockets". Astfel, Java ne pune la dispoziție o nouă bibliotecă "javax.net.ssl", unde a înlocuit vechile obiecte de tip "Sockets" cu obiecte de tip "SSLSockets" care vor face comunicatia sigură.

Obiectul "SSLServerSocket" este extins din "ServerSocket" și asigură o comunicație criptată. Instanțe ale acestei clase sunt de obicei create prin "SSLServerSocketFactory". Prin apelarea metodei SSLServerSocketFactory.getDefault() obținem un SSLServerSocketFactory cu configurațiile standard pe care îl putem folosi pentru a crea un SSLServerSocket.

De notat este faptul că apelul "SSLServerSocketFactory.getDefault().createServerSocket()" va întoarce un obiect de tip "ServerSocket" din acest motiv se face conversia în "SSLServerSocket" înainte de a se seta variabila "sslServerSocket").

De asemenea, apelul funcției "sslServerSocket.accept()" întoarce un obiect de tip "Socket" și în acest caz va trebui făcută conversia la "SSLSocket" înainte de a fi setată variabila "client".

```
try {
    System.out.println("Astepam conexiuni...");
    SSLServerSocket sslServerSocket = (SSLServerSocket)
SSLServerSocketFactory.getDefault().createServerSocket(port);
    SSLSocket client = (SSLSocket) sslServerSocket.accept();
```

După crearea obiectelor de tip "SSLSockets" și după ce am încărcat certificatele în sistem, protocolul este instalat și putem seta diferite configurări ale acestuia. Configurările prezentate în cadrul acestei aplicații constau în versiunea de protocol aleasă și în alegerea de suite de algoritmi pe care să se facă criptarea.

Pe baza informației de la tastatură citită anterior se va seta protocolul folosit prin apelul funcției "setEnabledProtocols", iar suita de algoritmi se va seta utilizând "setEnabledCipherSuites".

```
if(tls_version.equals("TLSv1.3")) {
    client.setEnabledProtocols(new String[] {"TLSv1.3"});
    client.setEnabledCipherSuites(new String[]
{"TLS_AES_128_GCM_SHA256"});
} else {
    client.setEnabledProtocols(new String[] {"TLSv1.2"});
}
```

În momentul în care un client s-a conectat la server, voi fi informat în consolă că există o conexiune prin printarea în consolă a adresei clientului. Adresa clientului se obține prin apelul funcției getInetAddress().

```
System.out.println("Connection established with: " + client.getInetAddress());
```

Asemănător aplicațiilor descrise anterior, creăm obiectele de scris și de citit pe stream-ul de intrare, respectiv de ieșire și afișăm mesajul trimis de client și în final prindem excepțiile.

```
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        PrintWriter output = new PrintWriter(client.getOutputStream(), true);
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println("De la client: " + line);
            output.println(line);
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Capitolul 4.5 Aplicația EchoSSLClient

Ca și în cazul aplicației “Server”, și aplicația “EchoSSLClient” ne oferă posibilitatea de a configura aplicația. În acest caz, informațiile pe care aplicația le cere sunt:

- Adresa serverului – un String;
- Portul serverului – un int;
- Opțiunea de logare a mesajelor ce au loc la nivelul protocolului – variabila de tip Boolean, în cazul în care este “true” aplicația va loga în consolă mesajele transmise.
- În cazul în care opțiunea de logare, este “true”, se va cere specificarea tipului de logare folosit.

Opțiunile de logare a mesajelor sunt după cum urmează:

- all – în cazul în care vreau ca aplicația să logheze toate evenimentele care au loc în timpul rulării aplicației.
- ssl – în cazul în care vreau ca aplicația să logheze numai evenimentele ce țin de protocolul SSL.

În cazul alegerii opțiunii “ssl” mai sunt valabile câteva opțiuni de logare:

- record: Urmărește activitățile fiecărei înregistrări;
- handshake: Afișează fiecare mesaj din procesul de handshake;
- keygen: Afișează informațiile privind generarea cheilor;
- session: Afișează activitatea sesiunii;
- defaultctx : Afișează initializarea standard a protocolului SSL;
- sslctx: Afișează activitatea SSLContext;
- sessioncache: Afișează activitatea sesiunii de cache;
- keymanager: Afișează activitățile ce țin de keymanager;
- trustmanager: Afișează activitățile trustmanager-ului.

Import bibliotecile folosite:

```
import javax.net.ssl.SSLSocket;
```

```
import javax.net.ssl.SSLSocketFactory;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Scanner;
```

Voi defini clasa si functia main si setam certificatele folosite.

```
public class EchoSSLClient {

    public static void main(String[] args) {

        System.setProperty("javax.net.ssl.keyStore",
"src/credentiale/tls_analysis.jks");
        System.setProperty("javax.net.ssl.keyStorePassword", "tlsanalysis");
        System.setProperty("javax.net.ssl.trustStore",
"src/credentiale/tls_analysis.jks");
        System.setProperty("javax.net.ssl.trustStorePassword",
"tlsanalysis");
```

Voi defini variabilele pentru port si adresa si inca trei variabile care vor define optiunile de mesaj de logare.

```
String hostname;
int port;
boolean enable_log = false;
String log_type = "";
String log_options = "";
String log_ssl_parameters = "";
```

Voi crea obiectul cu care vor putea fi citi mesajele introduse din consolă și verific dacă mesajul introdus este “true”, caz în care aplicația va loga mesajele în consolă și va cere mai multe opțiuni de logare.

```
        if(log_type.equals("ssl")) {
            System.out.println("Optiuni pentru logare de tip ssl: ");
            System.out.print("record, handshake, keygen, session, defaultctx,
sslctx, sessioncache, trustmanager");
            System.out.print("Alege optiunea: ");
            log_ssl_parameters = scanner.nextLine();
            System.setProperty("javax.net.debug", log_type + ":" + log_options
+ ":" + log_ssl_parameters);
        } else {
            System.setProperty("javax.net.debug", log_type + ":" + log_options);
        }
    }
```

Mesajul de logare a fost setat prin folosirea proprietatilor de sistem funcției System.setProperty() atribuind o valoare câmpului “javax.net.debug”.

Se afișează un mesaj de confirmare a adresei și portului și se crează obiectul care va genera sesiunea prin apelul celor două funcții SSLSocketFactory.getDefault() și createSocket(). Obiectul obținut se va converti într-un obiect de tip “SSLSockets” care asigură comunicații criptate.

```

        try {
            SSLSocket client = (SSLSocket)
SSLSocketFactory.getDefault().createSocket(hostname, port);
            System.out.println("Conectare cu succes!");
            Ca si in cazul aplicatiei EchoClient citeste mesajul din consola si il trimite
mai departe catre server si apoi inchidem clauza catch.

            try {
                SSLSocket client = (SSLSocket)
SSLSocketFactory.getDefault().createSocket(hostname, port);
                System.out.println("Conectare cu succes!");

                PrintWriter out = new PrintWriter(client.getOutputStream(), true);
                BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                while (true) {
                    System.out.println("Spune ceva prin SSL: ");
                    String input = scanner.nextLine();
                    if("exit".equalsIgnoreCase(input)) {
                        break;
                    }
                    out.println(input);
                    String response = bufferedReader.readLine();
                    System.out.println("Server: " + response);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

Capitolul 4.6 Aplicatiile ClassServer si ClassFileServer

Aplicatia constă în crearea unui server care distribuie o locație pe internet prin apelul unei cereri de tip “GET”. Astfel, serverul ar trebui să întoarcă biții fișierului cerut, pe standardul de ieșire.

Astfel, serverul trebuie să suporte următoarele funcționalități:

- Acceptarea de conexiuni multiple către server;
- Oferirea posibilității de a alege între o comunicație sigură folosind “HTTPS” sau o comunicație nesigură folosind “HTTP”;
- Punerea la dispoziție a unor fișiere dintr-un director care să poată fi accesate folosind HTTP sau HTTPS
- Parsarea cererii clientului pentru a obține numele fișierului care se dorește a fi accesat
- Scrierea pe “stream-ul” de ieșire biții fișierului pentru a distribui fișierul cu clientul.

Această aplicație este compusă din alte două clase ce reprezintă două servere, “ClassServer” care acționează ca un “Thread” și “ClassFileServer” care extinde această clasă pentru a putea suporta mai multe conexiuni simultan.

Capitolul 4.6.1 Aplicatia ClassServer

Serverul, trebuie să poată oferi accesul la fișier utilizatorului, în timp ce sesiunea este încă activă, astfel că va trebui să implementăm interfața “Runnable”.

Voi importa librăriile și interfața “Runnable” și voi defini clasa că fiind abstractă.

```
public abstract class ClassServer implements Runnable {  
  
    private ServerSocket server = null;  
  
    protected ClassServer(ServerSocket ss) {  
        server = ss;  
        newListener();  
    }  
}
```

Funcția ClassServer(ServerSocket) este un constructor care atribuie valoarea parametrului “ServerSocket” definit anterior.

În acest moment, voi defini funcția getBytes(String) care va întoarce biții fișierului

```
public abstract byte[] getBytes(String path) throws IOException,  
FileNotFoundException;
```

Aceasta poate arunca 2 eceptii, FileNotFoundException când fișierul nu este accesibil și IOException daca s-a întâmplat ceva cu conexiunea.

Deoarece am implementat “Runnable” trebuie să implementăm o funcție “run()” care nu primește niciun parametru. Această funcție permite îndeplinirea unei sarcini cât timp sesiunea este încă activă. În această funcție voi implementa tot ce tine de comunicația dintre client și server.

Acceptăm conexiunea cu clientul sau “arunc” o excepție în cazul în care conectarea nu a fost făcută cu succes.

```
public void run() {  
    Socket socket;  
  
    // Accepta o conexiune  
    try {  
        socket = server.accept();  
    } catch (IOException e) {  
        System.out.println("Class Server a picat...");  
        e.printStackTrace();  
        return;  
    }  
}
```

Apelăm funcția NewListener(), care creează un nou “Thread”. Însă nu o vom implementa acum și ne creăm obiectele care vor face scrierea și citirea.

```
newListener();  
  
    try {  
        OutputStream rawOut = socket.getOutputStream();  
        PrintWriter out = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(rawOut)));
```

În continuare voi citi mesajul de la client și pe baza acestui mesaj apelez funcția `getPath` care va întoarce numele fișierului cerut de către client și voi extrage biții fișierului apelând funcția `getBytes()` cu numele fișierului.

```
try {
    // Obține calea către fisier din header
    BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    String path = getPath(in);

    // Obține biții
    byte[] bytetimes = getBytes(path);
```

Se vor trimite biții fișierului drept răspuns, respectând standardul protocolului “HTTP”

```
try {
    out.print("HTTP/1.0 200 OK\r\n");
    out.print("Content-Length: " + bytetimes.length + "\r\n");
    out.print("Content-Type: text/html\r\n\r\n");
    out.flush();
    rawOut.write(bytetimes);
    rawOut.flush();
} catch (IOException ie) {
    ie.printStackTrace();
    return;
}
} catch (Exception e) {
    e.printStackTrace();
    // Scriem mesajul de eroare
    out.println("HTTP/1.0 400 " + e.getMessage() + "\r\n");
    out.println("Content-Type: text/html\r\n\r\n");
    out.flush();
}
} catch (IOException ex) {
    System.out.println("Error writing response: " + ex.getMessage());
    ex.printStackTrace();
} finally {
    try {
        socket.close();
    } catch (IOException e) {
    }
}
}
```

Voi defini funcția care creează Thread-ul

```
private void newListener() { (new Thread(this)).start(); }
```

Voi defini funcția `getPath` care parsează mesajul de la client și întoarce numele fișierului

```
private static String getPath(BufferedReader in) throws IOException {
    String line = in.readLine();
    String path = "";
```

```

        if(line.startsWith("GET /")) {
            line = line.substring(5, line.length()-1).trim();
            int index = line.indexOf(' ');
            if(index != -1) {
                path = line.substring(0, index);
            }
        }

        // Parcurgem restul header-ului
        do {
            line = in.readLine();
        } while ((line.length() != 0) && (line.charAt(0) != '\r') && (line.charAt(0)
!= '\n'));

        if (path.length() != 0) {
            return path;
        } else {
            throw new IOException("Header Malformat");
        }
    }
}

```

Capitolul 4.6.2 Aplicatia ClassFileServer

Deoarece aplicația aceasta se folosește structura clasei “ClassServer” va trebui să o extind pe aceasta din urmă în aplicația curentă.

Începem prin importarea librărilor, extinderea clasei și crearea constructorului. Voi defini și portul pe care va porni aplicația implicit, și o cale sub forma unui text, ce reprezintă calea directorului pus la dispoziția clientului.

```

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.*;
import java.security.*;
import java.util.Scanner;
import javax.net.*;
import javax.net.ssl.*;

public class ClassFileServer extends ClassServer {

    private String docroot;
    private static int DefaultServerPort = 2001;

    /** Construiește un obiect de tip ClassFileServer
     * Parametrii: calea către fisier
     */

    public ClassFileServer(ServerSocket ss, String docroot) {
        super(ss);
        this.docroot = docroot;
    }
}

```


Voi defini funcția definită și în aplicația anterioară care va întoarce biții fișierului cerut de client care are ca parametru calea către fișier.

```
public byte[] getBytes(String path) throws IOException {
    System.out.println("reading: " + path);
    File f = new File(docroot + File.separator + path);
    int length = (int)(f.length());
    if (length == 0) {
        throw new IOException("Dimensiunea fisierului este 0: " + path);
    } else {
        FileInputStream fin = new FileInputStream(f);
        DataInputStream in = new DataInputStream(fin);

        byte[] bytetimes = new byte[length];
        in.readFully(bytetimes);
        return bytetimes;
    }
}
```

Voi continua cu funcția main(), prin a declara variabilele folosite. Aplicația va citi din consolă informațiile de utilizare introduse de client pe care le va atribui variabilelor definite anterior, pe baza cărora se vor face ulterior diferite configurări.

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int port;
    String docroot = "src/test_file/";
    String protocol;
    boolean clientauth;

    System.out.print("Introdu portul pe care sa asculte server-ul: ");
    port = in.nextInt();
    in.nextLine();

    String line;
    System.out.print("Alege protocolul folosit (TLS pentru comunicatie
securizata, enter pentru comunicatie in plain text): ");
    line = in.nextLine();
    if(line.equals("TLS")) {
        protocol = "TLS";
        System.out.print("Server-ul foloseste ClientAuthentication (true pentru
da, false pentru nu): ");
        line = in.nextLine();

    } else {
        protocol = "PlainSocket";
    }
}
```

Se va afișa un mesaj de confirmare, urmat de verificarea alegerii opțiunii de către client de a filtra conexiunile pe baza certificatului prezentat.

```

        if(line.equals("true")) {
            clientauth = true;
        } else {
            clientauth = false;
        }
    }

```

Continui cu implementarea serverului prin crearea obiectelor folosite pentru a comunica.

```

        try {
            ServerSocketFactory ssf =
ClassFileServer.getServerSocketFactory(protocol);
            ServerSocket ss = ssf.createServerSocket(port);

```

Dacă opțiunea de autentificare a clientului a fost aleasă, se va cere autentificarea prin apelul funcției `setNeedClientAuth` cu parametrul `true`.

```

        if(clientauth == true) {
            ((SSLServerSocket)ss).setNeedClientAuth(true);
        }

```

Voi realiza un obiect de tip `ClassFileServer` cu conexiunea creată anterior și cu calea către directorul pe care să îl distribuie și prindem excepțiile în cazul în care acestea apar.

```

        new ClassFileServer(ss, docroot);
    } catch (IOException e) {
        System.out.println("Eroare pornind server-ul: " + e.getMessage());
        e.printStackTrace();
    }
}

```

În cele ce urmează voi define funcția `getServerSocketFactory()` care primește ca parametru un text, reprezentând protocolul ales.

Aplicația oferă două opțiuni pentru alegerea protocolului, `TLS` sau comunicația necriptată. În cazul folosirii comunicației criptate vom folosi acest constructor.

```

private static ServerSocketFactory getServerSocketFactory(String protocol) {
    if(protocol.equals("TLS")) {
        SSLServerSocketFactory ssf = null;

```

Pentru cazul în care voi cere autentificarea clientului, voi avea nevoie să ne creăm contextul sesiunii

```

        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase = "password".toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

```

Voi încărca în containerul de chei create certificatul pe care îl va folosi serverul și apoi voi returna conexiunea cu configurațiile setate.

```

        ks.load(new
FileInputStream("src/credentiale/OC/keys/server_ks.pkcs12"), passphrase);
        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), null, null);

        ssf = ctx.getServerSocketFactory();

        return ssf;
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    return ServerSocketFactory.getDefault();
}
return null;
}
}

```

Capitolul 4.7 Aplicatia SSLSocketAuth

Voi librăriile și voi defini variabilele folosite.

```

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
import java.security.KeyStore;
import java.util.Scanner;

public class SSLSocketClientAuth {

    public static void main(String[] args) {

        String host = "";
        int port = -1;
        String path = "";
        boolean enable_log = false;
        String log_type = "";
        String log_options = "";
        String log_ssl_parameters = "";
    }
}

```

Aceste variabile vor desemna alegerile utilizatorului introduse din consolă. Ele reprezintă adresa și portul serverului, numele fișierului care se dorește a fi accesat și opțiunile de logare în cazul în care s-a dorit ca logarea mesajelor să fie activă.

```

Scanner scanner = new Scanner(System.in);

System.out.print("Introdu hostname-ul la care sa ne conectam: ");
host = scanner.nextLine();

System.out.print("Introdu port-ul pe care asculta server-ul: ");
port = scanner.nextInt();
scanner.nextLine();

```

```

System.out.print("Introdu numele fisierului pe care vrei sa-l accesezi: ");
path = scanner.nextLine();

String line;
System.out.print("Logam mesajul (true pentru da, false pentru nu): ");
line = scanner.nextLine();

```

Dacă logarea se dorește a fi activată, aplicația va cere informații suplimentare asupra modului cum se va face logarea.

Opțiunile de logare a mesajelor sunt asemănătoare celor prezentate în cadrul aplicației EchoSSLClient.

```

if(line.equals("true")) {
    enable_log = true;
    System.out.print("Ce log sa folosim (all sau ssl): ");
    log_type = scanner.nextLine();
    System.out.print("Ce optiune sa alegem pentru log(data, verbose): ");
    log_options = scanner.nextLine();
    if(log_type.equals("ssl")) {
        System.out.println("Optiuni pentru logare de tip ssl: ");
        System.out.print("record, handshake, keygen, session, defaultctx,
sslctx, sessioncache, keymanager, trustmanager");
        System.out.print("Alege optiunea: ");
        log_ssl_parameters = scanner.nextLine();
        System.setProperty("javax.net.debug", log_type + ":" + log_options
+ ":" + log_ssl_parameters);
    } else {
        System.setProperty("javax.net.debug", log_type + ":" + log_options);
    }
}

```

Asemănător aplicației anterioare, voi crea obiectul care se va ocupa de administrarea cheilor și certificatelor și "prindem" excepțiile în cazul în care au apărut.

```

try {
    SSLSocketFactory factory = null;
    try {
        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase = "openssl22".toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

        ks.load(new FileInputStream("src/credentiale/clientAuth.jks"),
passphrase);

        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), null, null);

        factory = ctx.getSocketFactory();
    }
}

```

```

    } catch (Exception e) {
        throw new IOException(e.getMessage());
    }

```

Se va realiza conexiunea cu serverul și voi porni procesul de “handshake”.

```

SSLSocket socket = (SSLSocket) factory.createSocket(host, port);

socket.startHandshake();

```

Voi crea obiectul care va face scrierea și va trimite un mesaj de tip “HTTP GET”:

```

PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())));
out.println("GET /" + path + " HTTP/1.0");
out.println();
out.flush();

```

Se vor afișa erorile posibile:

```

if(out.checkError()) {
    System.out.println("SSLSocketClient: java.io.PrintWriter error");
}

```

În cele din urmă, voi realiza obiectul care va scrie în consolă, închidem conexiunile și prindem excepțiile.

```

BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

String inputLine;
while((inputLine = in.readLine()) != null) {
    System.out.println(inputLine);
}

in.close();
out.close();
socket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Capitolul 5 Testare/Utilizare

Capitolul 5.1 Pornirea aplicatiilor in IDE-ul IntelliJ

Pentru a exemplifica pornirea aplicațiilor, voi porni aplicațiile EchoServer și EchoClient.

Pentru pornirea aplicației utilizând IDE-ul IntelliJ deschidem fila cu codul sursă al aplicației și apăsăm pe săgeată verde din dreptul numelui clasei și apoi apăsăm “Run EchoServer.main()”. Procesul pornirii aplicației EchoServer este descris în figura de mai jos. Pornirea celorlalte aplicații se face în mod similar.

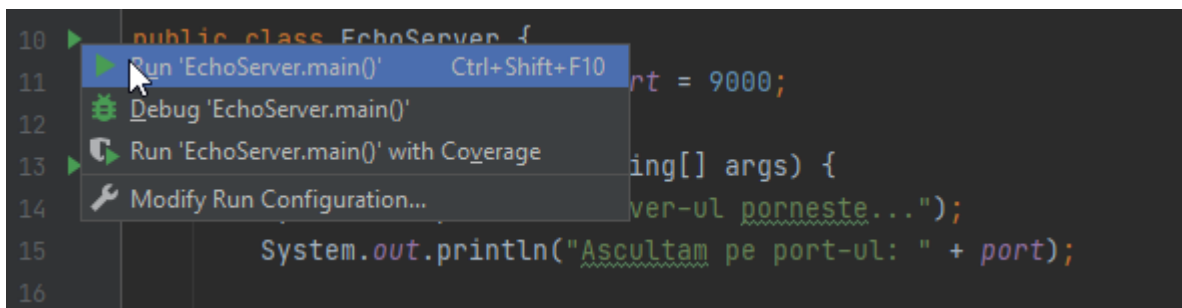


Figura 5. 1 Pornirea aplicatiei EchoServer

După pornirea aplicației în consolă ar trebui să fie vizibile mesajele care confirmă faptul că serverul a pornit și că așteaptă conexiuni.

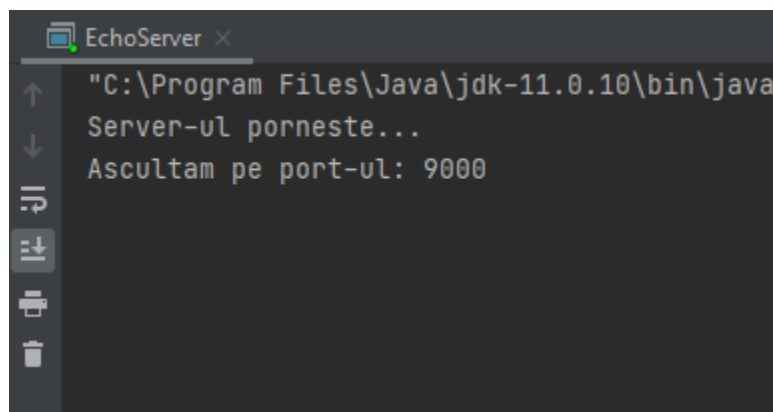


Figura 5. 2 Confirmarea pornirii aplicatiei EchoServer

În mod similar, pentru pornirea aplicației EchoClient, voi deschide fila ce va conține codul sursă al aplicației și rulăm aplicația. Dacă totul a funcționat corect, lângă consola EchoServer ar trebui să se deschidă o consolă nouă. Afișează mesajul cu confirmarea conexiunii la server și așteaptă introducerea unui mesaj pe care să-l trimită acestuia.

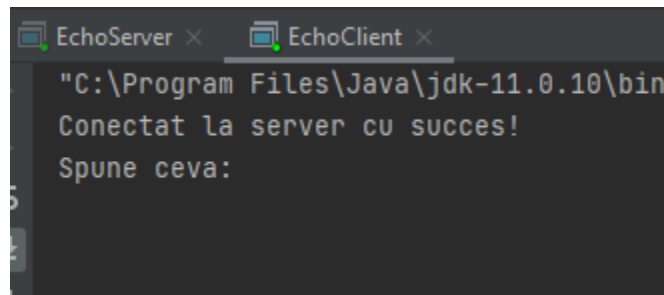


Figura 5.3 Confirmarea pornirii aplicației EchoClient

În consola aplicației EchoServer primim mesajul de confirmare a conexiunii dintre client și server.

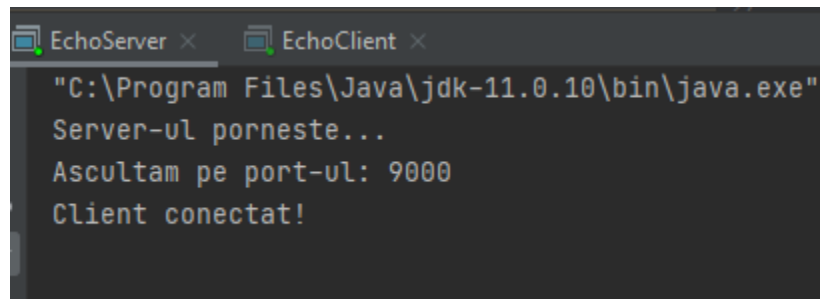


Figura 5.4 Confirmarea conexiunii clientului

Capitolul 5.2 Utilizarea Wireshark

Se deschide aplicația Wireshark. În cazul experimentelor descrise în acest proiect analiza traficului s-a făcut numai pe traficul local.

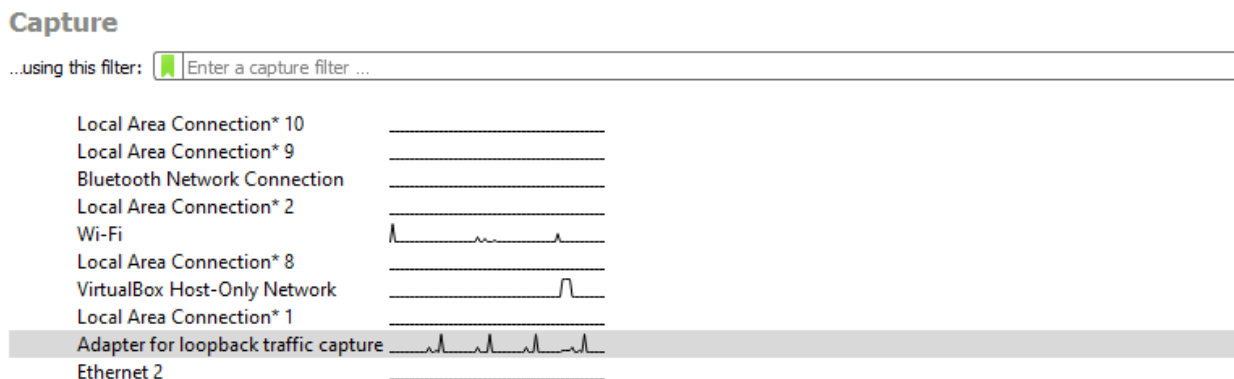


Figura 5.5 Utilizarea Wireshark pe traficul local

În cazul în care dorim să aplicăm un filtru analizei traficului putem folosi bara de filtre, “display filter”

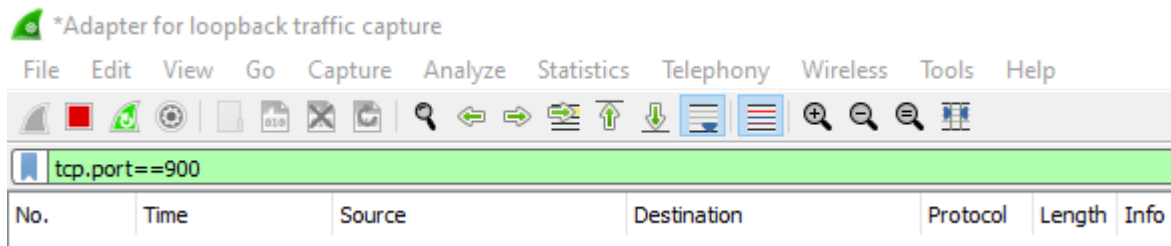


Figura 5. 6 Aplicarea filtrului pentru port in Wireshark

Capitolul 5.3 Functionarea aplicatiilor EchoServer si EchoClient

În momentul introducerii unui mesaj în consola aplicației EchoClient, voi primi acest mesaj în consola aplicației EchoServer.

```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Conectat la server cu succes!
Spune ceva:
Hello world!
Spune ceva:
|
```

Figura 5. 7 Trimiterea unui mesaj folosind EchoClient

```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Server-ul porneste...
Ascultam pe port-ul: 9000
Client conectat!
De la Client: Hello world!
```

Figura 5. 8 Afisarea mesajului trimis

```
Conectat la server cu succes!
Spune ceva:
Hello!
Spune ceva:
exit

Process finished with exit code 0
|
```

Figura 5. 9 Folosirea cuvântului "exit" pentru terminarea aplicației EchoClient

Capitolul 5.4 Pornirea aplicatiilor EchoServer si EchoClient.

Pentru început, trebuie pornită aplicația EchoServer deoarece aceasta este aplicația care așteaptă conexiuni.

Pentru pornirea aplicației utilizând IDE-ul IntelliJ deschidem fila cu codul sursă al aplicației și apăsăm pe săgeata verde din dreptul numelui clasei și apoi apăsăm “Run EchoServer.main()”. Procesul pornirii aplicației EchoServer este descris în figura de mai jos. Celelalte aplicații pornesc în mod similar.

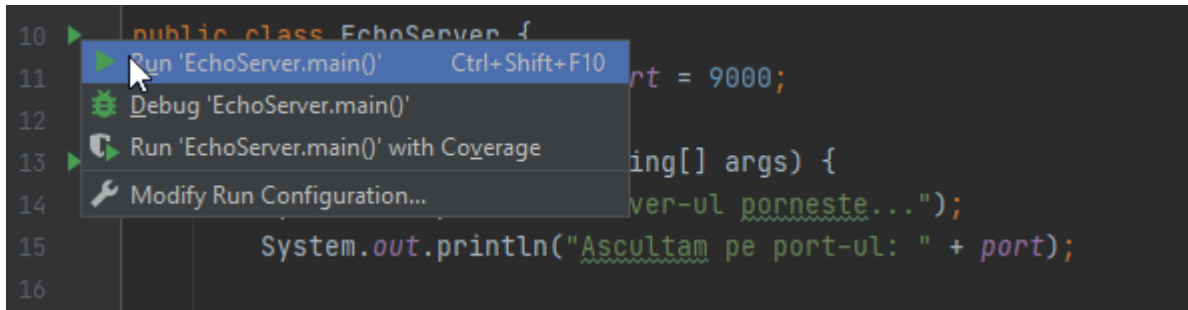


Figura 5. 10 Pornirea aplicatiei EchoServer

În urma pornirii aplicației, în consolă ar trebui să se afișeze mesaje de confirmare cu pornirea serverului și portul pe care acesta așteaptă conexiuni.

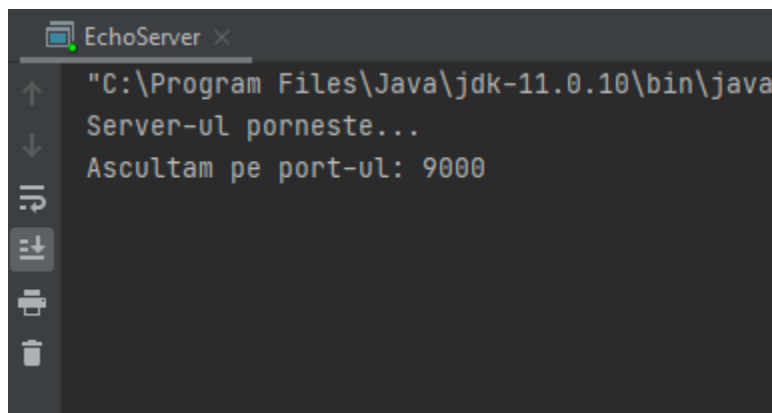


Figura 5. 11 Confirmarea pornirii aplicatiei EchoServer

În mod similar, pentru pornirea aplicației EchoClient, deschidem fila cu codul sursa al aplicației și rulăm aplicația. Dacă totul a funcționat corect, lângă consola EchoServer ar trebui să se deschidă o consolă nouă. Se afișează mesajul cu confirmarea conexiunii la server și așteaptă introducerea unui mesaj pe care să-l trimită acestuia.

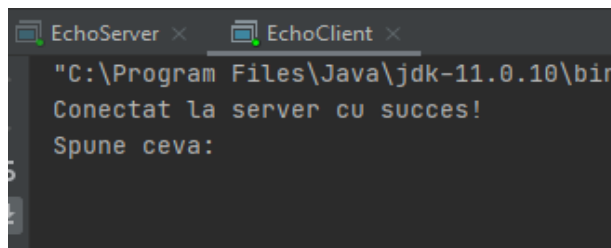
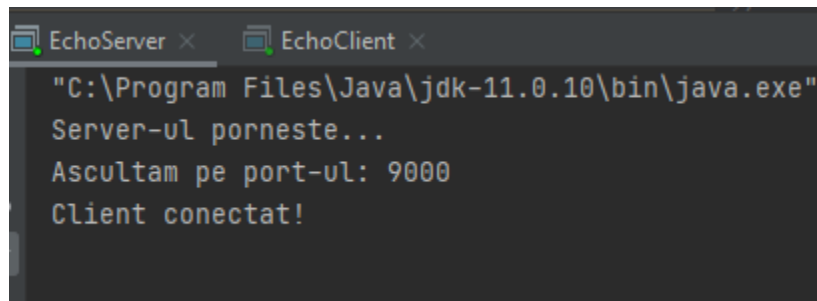


Figura 5. 12 Confirmarea pornirii aplicatiei EchoClient

În consola aplicației EchoServer primim mesajul de confirmare a conexiunii dintre client și server.

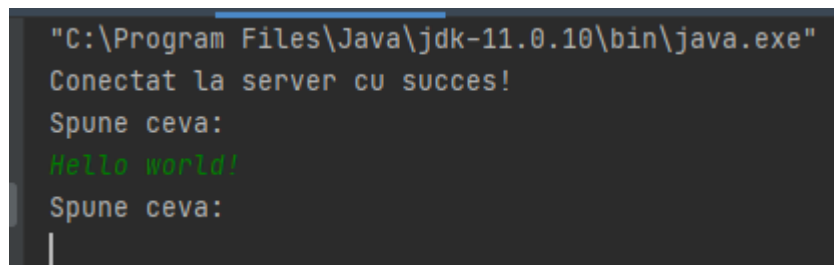


```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Server-ul porneste...
Ascultam pe port-ul: 9000
Client conectat!
```

Figura 5. 13 Confirmarea conexiunii clientului

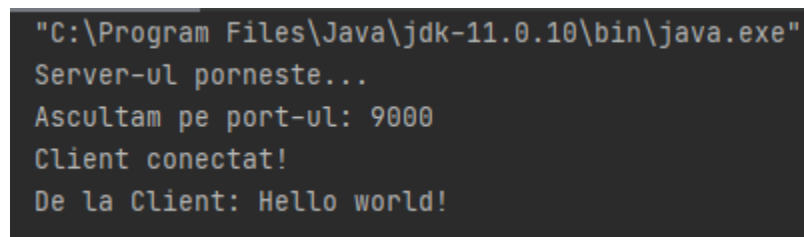
Capitolul 5.5 Functionarea aplicatiilor EchoSSLServer si EchoSSLClient

În momentul introducerii unui mesaj în consola aplicatiei EchoClient, vom primi acest mesaj in consola aplicatiei EchoServer.



```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Conectat la server cu succes!
Spune ceva:
Hello world!
Spune ceva:
|
```

Figura 5. 14 Trimiterea unui mesaj folosind EchoClient



```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Server-ul porneste...
Ascultam pe port-ul: 9000
Client conectat!
De la Client: Hello world!
```

Figura 5. 15 Afisarea mesajului trimis

Capitolul 5.6 Utilizarea tool-ului Wireshark pentru a captura traficul dintre EchoClient si EchoServer

În continuare, voi trimite din nou un mesaj de pe aplicația EchoClient. În Wireshark se poate observa acum captura traficului dintre client și server. Se constată că mesajul trimis poate fi citit și înțeles din Wireshark.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|------------|-----------|-------------|----------|--------|---|
| 370 | 274.202202 | 127.0.0.1 | 127.0.0.1 | S101 | 98 | 61673 → 9000 [PSH, ACK] Seq=1 Ack=1 Win=10233 Len=14 |
| 371 | 274.202243 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 61673 [ACK] Seq=1 Ack=15 Win=10233 Len=0 |
| 372 | 274.202503 | 127.0.0.1 | 127.0.0.1 | S101 | 98 | 9000 → 61673 [PSH, ACK] Seq=1 Ack=15 Win=10233 Len=14 |
| 373 | 274.202516 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 61673 → 9000 [ACK] Seq=15 Ack=15 Win=10233 Len=0 |

> Frame 372: 98 bytes on wire (784 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{...}_Loopback, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 9000, Dst Port: 61673, Seq: 1, Ack: 15, Len: 14

> Data (14 bytes)

```

0000  02 00 00 00 45 00 00 36 ce 42 40 00 80 06 00 00  ....E...B@....
0010  7f 00 00 01 7f 00 00 01 23 28 f0 e9 92 c6 95 26  ....#(.....&
0020  52 17 35 54 50 18 27 f9 67 5f 00 00 48 65 6c 6c  R-STP...g...Hell
0030  6f 20 57 6f 72 6c 64 21 0d 0a                   o World! ..

```

Figura 5. 16 Captura trafic Wireshark dintre EchoServer si EchoClient

Se poate observa din Wireshark mesajul transmis : “Hello World”. Aceasta este o gravă problemă de securitate. Practic, aceasta dovedește faptul că un atacator care nu comunică în mod direct cu aplicațiile EchoClient și EchoServer, ci doar fiind conectat la aceeași rețea. Asemănător, un atacator ar putea vedea datele bancare sau alte informații sensibile atunci când se accesează diferite site-uri web, precum magazinele online.

Capitolul 5.7 Demonstrarea functionarii aplicatiei Server.

În cazul aplicației Server singurele configurări pe care le putem face sunt la nivelul versiunii de protocol folosite și a portului folosit. În acest caz, pentru aplicația Server putem testa doar corectitudinea negocierii protocolului. În acest sens, se pornește aplicația Server, se alege portul pe care asculta serverul și alegem varianta de protocol pe care dorim să o testăm.

În acest prim test, am setat că portul serverului să fie 9000 și varianta protocolului să fie “TLS1.3”.

```

Introdu port-ul pe care asculta server-ul: 9000
Server-ul asculta pe port-ul: 9000
Ce versiune de protocol folosim (TLSv1.2, TLSv1.3): TLSv1.3

```

Figura 5. 17 Pornirea aplicatiei Server cu negocierea versiunii TLS 1.3

După pornirea aplicației Server, putem porni aplicația EchoSSLClient pentru a ne conecta la server. În cazul proiectului de fata, adresa la care ne vom conecta va fi întotdeauna “localhost” și portul este cel introdus mai devreme în aplicația Server.

```

Introdu host-ul server-ului: localhost
Introdu port-ul pe care asculta server-ul: 9000
Logam mesajul (true pentru da, false pentru nu): false
Conectare catre: localhost pe port-ul: 9000
Conectare cu succes!
Spune ceva prin SSL:
Hello World!
Server: Hello World!
Spune ceva prin SSL:

```

Figura 5. 18 Pornirea aplicatiei EchoSSLClient fara logare cu negocierea versiunii TLS 1.3

Pentru a putea testa corectitudinea negocierii versiunii de protocol, putem folosi captura de trafic Wireshark.

Se deschide aplicația Wireshark cu filtrul pe “Adapter for loopback traffic capture”. De pe aplicația EchoSSLClient se transmite un mesaj către server și se verifică captura efectuată de Wireshark care ne vă confirma varianta de protocol folosită. În urma negocierii versiunii TLS 1.3 și a transmiterii mesajului “Hello World!” s-a obținut următoarea captura de trafic:

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|-----------|-------------|----------|--------|---|
| 1 | 0.000000 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 451 | Client Hello |
| 2 | 0.000042 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 54643 [ACK] Seq=1 Ack=368 Win=10233 Len=0 |
| 3 | 0.031120 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 211 | Server Hello |
| 4 | 0.031141 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=368 Ack=128 Win=10233 Len=0 |
| 5 | 0.038581 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 90 | Change Cipher Spec |
| 6 | 0.038601 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=368 Ack=134 Win=10233 Len=0 |
| 7 | 0.041910 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 90 | Change Cipher Spec |
| 8 | 0.041936 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 54643 [ACK] Seq=134 Ack=374 Win=10233 Len=0 |
| 9 | 0.045995 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 154 | Application Data |
| 10 | 0.046019 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=374 Ack=204 Win=10232 Len=0 |
| 11 | 0.047620 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 1036 | Application Data |
| 12 | 0.047641 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=374 Ack=1156 Win=10229 Len=0 |
| 13 | 0.080424 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 386 | Application Data |
| 14 | 0.080447 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=374 Ack=1458 Win=10227 Len=0 |
| 15 | 0.081528 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 158 | Application Data |
| 16 | 0.081540 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=374 Ack=1532 Win=10227 Len=0 |
| 17 | 0.085411 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 158 | Application Data |
| 18 | 0.085433 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 54643 [ACK] Seq=1532 Ack=448 Win=10233 Len=0 |
| 19 | 0.086362 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 135 | Application Data |
| 20 | 0.086378 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 54643 [ACK] Seq=1532 Ack=499 Win=10233 Len=0 |
| 21 | 0.087474 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 172 | Application Data |
| 22 | 0.087490 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=499 Ack=1620 Win=10227 Len=0 |
| 23 | 0.089670 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 135 | Application Data |
| 24 | 0.089688 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 54643 → 9000 [ACK] Seq=499 Ack=1671 Win=10227 Len=0 |

Figura 5. 19 Captura de trafic asupra negocierii versiunii TLS 1.3

În continuare, vom rula același test, dar impunând aplicației “Server” să folosească versiunea anterioară a protocolului. Astfel, se pornește aplicația “Server” că și până acum, specificând varianta aleasa că fiind “TLSv1.2”.

```

Introdu port-ul pe care asculta server-ul: 9000
Server-ul asculta pe port-ul: 9000
Ce versiune de protocol folosim (TLSv1.2, TLSv1.3): TLSv1.2
Astepam conexiuni...
Connection established with: /127.0.0.1
From client: Hello World!
|

```

Figura 5. 20 Pornirea aplicației Server cu negocierea versiunii TLS 1.2

Rulăm aplicația EchoSSLClient și trimitem și de aceasta dată un mesaj către server pe care îl vom intercepta cu Wireshark. De această dată, protocolul identificat de Wireshark a fost “TLS 1.2”.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|-----------|-------------|----------|--------|---|
| 1 | 0.000000 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 451 | Client Hello |
| 2 | 0.000024 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 59427 [ACK] Seq=1 Ack=368 Win=10233 Len=0 |
| 3 | 0.023005 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 174 | Server Hello |
| 4 | 0.023046 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=368 Ack=91 Win=10233 Len=0 |
| 5 | 0.023191 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 1000 | Certificate |
| 6 | 0.023200 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=368 Ack=1007 Win=10229 Len=0 |
| 7 | 0.040616 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 389 | Server Key Exchange |
| 8 | 0.040637 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=368 Ack=1312 Win=10228 Len=0 |
| 9 | 0.040853 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 93 | Server Hello Done |
| 10 | 0.040862 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=368 Ack=1321 Win=10228 Len=0 |
| 11 | 0.047635 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 126 | Client Key Exchange |
| 12 | 0.047666 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 59427 [ACK] Seq=1321 Ack=410 Win=10233 Len=0 |
| 13 | 0.054612 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 90 | Change Cipher Spec |
| 14 | 0.054630 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 59427 [ACK] Seq=1321 Ack=416 Win=10233 Len=0 |
| 15 | 0.060676 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 129 | Encrypted Handshake Message |
| 16 | 0.060700 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 59427 [ACK] Seq=1321 Ack=461 Win=10233 Len=0 |
| 17 | 0.067897 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 90 | Change Cipher Spec |
| 18 | 0.067917 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=461 Ack=1327 Win=10228 Len=0 |
| 19 | 0.068083 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 129 | Encrypted Handshake Message |
| 20 | 0.068091 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=461 Ack=1372 Win=10228 Len=0 |
| 21 | 0.069238 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 127 | Application Data |
| 22 | 0.069253 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 9000 → 59427 [ACK] Seq=1372 Ack=504 Win=10233 Len=0 |
| 23 | 0.069823 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 127 | Application Data |
| 24 | 0.069836 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 59427 → 9000 [ACK] Seq=504 Ack=1415 Win=10228 Len=0 |

Figura 5. 21 Captură de trafic asupra negocierii versiunii TLS 1.2

În urma rulării celor două teste și a capturilor de trafic, putem face o analiză a celor două protocoale.

Versiunea nouă de protocol a făcut îmbunătățiri majore în procesul de "handshake", pe lângă faptul că a redus numărul de cicluri necesare autentificării și astfel obține o performanță mai bună, acum schimbul de algoritmi folosiți se va face înaintea prezentării certificatelor.

Deși TLS 1.2 criptează mesajele transmise, facandu-le indescifrabile, atacatorii tot pot vedea certificatele prezentate după cum se observă în figurile de mai jos.

Astfel, începând criptarea înaintea transmiterii certificatelor, TLS 1.3 oferă o mai bună securizare, deoarece acestea nu mai pot fi citite.

| | | | | | |
|----|----------|-----------|-----------|---------|--|
| 9 | 5.121616 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 451 Client Hello |
| 10 | 5.121670 | 127.0.0.1 | 127.0.0.1 | TCP | 84 9000 → 57547 [ACK] Seq=1 Ack=368 Win=10233 Len=0 |
| 11 | 5.148277 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 174 Server Hello |
| 12 | 5.148299 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57547 → 9000 [ACK] Seq=368 Ack=91 Win=10233 Len=0 |
| 13 | 5.148462 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 1000 Certificate |
| 14 | 5.148472 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57547 → 9000 [ACK] Seq=368 Ack=1007 Win=10229 Len=0 |
| 15 | 5.166360 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 389 Server Key Exchange |
| 16 | 5.166382 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57547 → 9000 [ACK] Seq=368 Ack=1312 Win=10228 Len=0 |
| 17 | 5.166541 | 127.0.0.1 | 127.0.0.1 | TLSv1.2 | 93 Server Hello Done |
| 18 | 5.166554 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57547 → 9000 [ACK] Seq=368 Ack=1321 Win=10228 Len=0 |

Transport Layer Security

▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 911

▼ Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 907
Certificates Length: 904

▼ Certificates (904 bytes)
Certificate Length: 901

▼ Certificate: 3082038130820269a00302010202040ba7e5ba300d06092a.. (id-at-commonName=Bogdan Hasan,id-at-organizationalUnitName=IT,id-at-organizationName=Facultatea ETTI,id-at-localityName=Bucuresti,id-at-stateOrProvinceName=Romania,id-at-countryName=RO)

▼ signedCertificate
version: v3 (2)
serialNumber: 195552698
signature (sha256WithRSAEncryption)
Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)

▼ issuer: rdnSequence (0)
> rdnSequence: 6 items (id-at-commonName=Bogdan Hasan,id-at-organizationalUnitName=IT,id-at-organizationName=Facultatea ETTI,id-at-localityName=Bucuresti,id-at-stateOrProvinceName=Romania,id-at-countryName=RO)
> validity
> subject: rdnSequence (0)
> subjectPublicKeyInfo
> extensions: 1 item

▼ algorithmIdentifier (sha256WithRSAEncryption)
Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
padding: 0
encrypted: d80063f9371bdf1e0b635ba5fa0e0e18d5ae7d9fb96d8cc5..

Figura 5. 22 Certificatul transmis folosind versiunea 1.2 a protocolului TLS

| | | | | | |
|----|----------|-----------|-----------|---------|---|
| 4 | 4.397249 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 451 Client Hello |
| 5 | 4.397276 | 127.0.0.1 | 127.0.0.1 | TCP | 84 9000 → 65267 [ACK] Seq=1 Ack=368 Win=2619648 Len=0 |
| 6 | 4.431377 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 211 Server Hello |
| 7 | 4.431401 | 127.0.0.1 | 127.0.0.1 | TCP | 84 65267 → 9000 [ACK] Seq=368 Ack=128 Win=2619648 Len=0 |
| 8 | 4.439003 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 90 Change Cipher Spec |
| 9 | 4.439041 | 127.0.0.1 | 127.0.0.1 | TCP | 84 65267 → 9000 [ACK] Seq=368 Ack=134 Win=2619648 Len=0 |
| 10 | 4.443422 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 90 Change Cipher Spec |

Figura 5. 23 Noul proces de "handshake" al versiunii 1.3 a protocolului TLS

O altă metodă de a verifica corectitudinea implementării protocolului este folosirea utilitarului “OpenSSL”.

Spre deosebire de Wireshark, “OpenSSL” nu ne va spune în mod clar varianta de protocol folosită, dar putem face teste specifice asupra unei anumite versiuni a protocolului și de a analiza rezultate obținute.

Pentru următorul test, se pornește aplicația Server. Se setează un port pe care să asculte, iar versiunea negociată se alege “TLSv1.3”. Pentru a putea testa cu OpenSSL versiunea de TLS, putem să-I specificăm cu ce versiune a protocolului să încerce conexiunea. Astfel, vom încerca mai întâi să ne conectăm la server cu OpenSSL folosind mai întâi versiunea 1.3 și apoi 1.2.

Astfel, după ce am pornit server-ul, se deschide un command prompt și se introduce următoarea comandă:

```
"openssl s_client -connect localhost:9000 -tls1_3".
```

Această comandă vă încerca să se conecteze la aplicația “Server” folosind ultima versiune a protocolului. Dacă comanda a fost efectuată cu success, ni se va afișa certificatul.

```

C:\Users\BogdanHasan>openssl s_client -connect localhost:9000 -tls1_3
CONNECTED(00000004)
Can't use SSL_get_servername
depth=0 C = RO, ST = Romania, L = Bucuresti, O = Facultatea ETTI, OU = IT, CN = Bogdan Hasan
verify error:num=18:self signed certificate
verify return:1
depth=0 C = RO, ST = Romania, L = Bucuresti, O = Facultatea ETTI, OU = IT, CN = Bogdan Hasan
verify return:1
---
Certificate chain
 0 s:C = RO, ST = Romania, L = Bucuresti, O = Facultatea ETTI, OU = IT, CN = Bogdan Hasan
  i:C = RO, ST = Romania, L = Bucuresti, O = Facultatea ETTI, OU = IT, CN = Bogdan Hasan
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDgTCCAmmgAwIBAgIEC6fluJANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQGEwJS
TzEQMA4GA1UECBMHUM9tYW5pYTESMBAGA1UEBxMJQnVjdXJlc3RpMRgwFgYDVQQK
Ew9GYWN1bHRhdGVhIEVUVEkxCzAJBgNVBAsTAK1UMRUwEwYDVQQDEwxCb2dkYW4g
SGFzYW4wHhcNMjEwODIwMTcwMTIwHcNMjEwMTc4MTcwMTIwWjBxMQswCQYDVQQG
EwJSTzEQMA4GA1UECBMHUM9tYW5pYTESMBAGA1UEBxMJQnVjdXJlc3RpMRgwFgYD
VQQKEw9GYWN1bHRhdGVhIEVUVEkxCzAJBgNVBAsTAK1UMRUwEwYDVQQDEwxCb2dk
YW4gSGFzYW4wggEiMA0GCSCqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaWf7lvKem
l/FtI2qQhQJhzKEUWUcMSnoH4NpBt0kxYbxNjZf7tCgJXXy/jH/gh1fnMLareAI
MuD9KFfuHUKwab93a5te0FadfgaakztEYzcNcaX4L66QbRwJFWK48wRv3TnPCfSE
KggAfwS0i+JBZDeomHcnjQdxWBHVvncByBYdNFMDjwH70/PcSDT/5M9DNpTzNch1
j8Fxr0zhZBklsdjoPikNkgSwH7yK1wvLfzPJJ0w+DeMxBbxd+rsDeB4Tzs8pdpbn
I20xn31JUiHnvOg4W40EeLj7aUfWGi90wV+iMDvskE2+LqX0SWR2LXNFDDeNVFv
RB3h70yEsZy/AgMBAAGjITAFMB0GA1UdDgQWBBRx3zitaQPOBvmSXb/KCWwug8h/
zTANBgkqhkiG9w0BAQsFAAOCAQEAA2ABj+Tcb3x4LY1ul+g40GNWufz+5bYzFVviZ
++fAG0JmNXfLUKNGu7XaDy80CSVBksSRDFUr88vhD7Xf9KYXgerriuteWmM7IuhK
4Y99lomU6LLNGucCdNULGgjc8r0vATO6X1BXqEVImZTMZCngPfNzJrWRXtxgv2W
Iqniv8TW4uXhvnY/PxoJ9JP1RfontxQemivMder3oBX3R4QxpjWixS1ZH2EX6FD
KZ90ZdIBR7h89qc+EECKa105eFog1xXt282asHHIAm6kQogI6z76QLUOMN1zsKh8
05gf7b1wsgHu4N8siq3+5kGworKcnb1DwvrdMPOBbbia3GhJeQ==
-----END CERTIFICATE-----

```

Figura 5. 24 Folosirea OpenSSL pentru verificarea versiunii de TLS

Este important să vedem ce rezultate obținem și când încercăm să ne conectăm la server forțând versiunea anterioară când serverul impune versiunea cea mai recentă. Pentru aceasta, se pornește iar aplicația Server cu aceleași setări ca și pentru testul anterior. De pe OpenSSL se va încerca conectarea cu versiunea “1.2” rulând comanda:

```
"openssl s_client -connect localhost:9000 -tls1_2".
```

De această dată, conexiunea a eșuat ceea ce ne confirmă că aplicația negociază corect versiunile folosite.


```

C:\Users\BogdanHasan>openssl s_client -connect localhost:9000 -tls1_2
CONNECTED(00000004)
34359836736:error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version:ssl/record/rec_layer_s3.c:1543:SSL
alert number 70
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 202 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1.2
    Cipher   : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1631150868
    Timeout  : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
---

```

Figura 5. 25 OpenSSL conectare cu versiunea 1.2 la un server 1.3

Eroarea capturată de aplicația “Server” ne confirmă faptul că cele două aplicații nu s-au putut conecta din cauza variantei de TLS .

```

Introdu port-ul pe care asculta server-ul: 9000
Server-ul asculta pe port-ul: 9000
Ce versiune de protocol folosim (TLSv1.2, TLSv1.3): TLSv1.3
Așteptam conexiuni...
Connection established with: /0:0:0:0:0:0:1
javax.net.ssl.SSLHandshakeException: Create breakpoint : Client requested protocol TLSv1.2 is not enabled or supported in server context

```

Figura 5. 26 Eroare aplicație “Server” când versiunile negociate diferă.

Capitolul 5.8 Testarea aplicației EchoSSLClient

Pentru testarea acestei aplicații se poate rula cu diferite opțiuni de logare și apoi analizarea acestora.

Un exemplu de test pentru aplicația EchoSSLClient, este conectarea acesteia la aplicația “Server” și alegerea unui mod de logare a mesajelor. Exemplul următor are în vedere logarea de mesaje de tip “ssl:verbose:handshake”. Rezultatul este un mesaj destul de lung cu tot procesul de handshake. Câteva dintre informațiile logate de client vor fi expuse în figurile de mai jos.


```

Introdu host-ul server-ului: localhost
Introdu port-ul pe care asculta server-ul: 9000
Logan mesajul (true pentru da, false pentru nu): true
Ce log sa folosim (all sau ssl): all
Ce optiune sa alegem pentru log(data, verbose): verbose
Optiuni pentru logare de tip ssl:
record, handshake, keygen, session, defaultctx, sslctx, sessioncache, keymanager, trustmanagerAlege optiunea: handshake
Conectare catre: localhost pe port-ul: 9000
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:22.472 EEST|SSLCipher.java:438|jdk.tls.keyLimits: entry = AES/GCM/NoPadding KeyUpdate 2^37. AES/GCM/NO_PADDING:KEYUPDATE = 137438953472
Conectare cu succes!
Spune ceva prin SSL:
Hello World!
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.625 EEST|HandshakeContext.java:296|Ignore unsupported cipher suite: TLS_AES_128_GCM_SHA256 for TLS12
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.625 EEST|HandshakeContext.java:296|Ignore unsupported cipher suite: TLS_AES_256_GCM_SHA384 for TLS12
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.628 EEST|HandshakeContext.java:296|Ignore unsupported cipher suite: TLS_AES_128_GCM_SHA256 for TLS11
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.628 EEST|HandshakeContext.java:296|Ignore unsupported cipher suite: TLS_AES_256_GCM_SHA384 for TLS11
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.628 EEST|HandshakeContext.java:296|Ignore unsupported cipher suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 for TLS11

```

Figura 5. 27 Logarea mesajului de handshake de catre client – negocierea de cipher suites

```

javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.699 EEST|SSLExtensions.java:192|Consumed extension: supported_versions
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|ServerHello.java:968|Negotiated protocol version: TLSv1.3
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|SSLExtensions.java:163|Ignore unsupported extension: server_name
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|SSLExtensions.java:163|Ignore unsupported extension: max_fragment_length
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|SSLExtensions.java:163|Ignore unsupported extension: status_request
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|SSLExtensions.java:163|Ignore unsupported extension: ec_point_formats
javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.700 EEST|SSLExtensions.java:163|Ignore unsupported extension: application_layer_protocol_negotiation

```

Figura 5. 28 Logarea mesajului de handshake de catre client – negocierea versiunii protocolului.

```

javax.net.ssl|DEBUG|01|main|2021-09-09 04:40:34.725 EEST|CertificateMessage.java:1171|Consuming server Certificate handshake message (
"Certificate": {
  "certificate_request_context": "",
  "certificate_list": [
    {
      "certificate" : {
        "version"      : "v3",
        "serial number" : "0B A7 E5 8A",
        "signature algorithm": "SHA256withRSA",
        "issuer"        : "CN=Bogdan Hasan, OU=IT, O=Facultatea ETTI, L=Bucuresti, ST=Romania, C=RO",
        "not before"    : "2021-08-20 20:01:20.000 EEST",
        "not after"     : "2021-11-18 19:01:20.000 EET",
        "subject"       : "CN=Bogdan Hasan, OU=IT, O=Facultatea ETTI, L=Bucuresti, ST=Romania, C=RO",
        "subject public key" : "RSA",
        "extensions"    : [
          {
            ObjectId: 2.5.29.14 Criticality=false
            SubjectKeyIdentifier [
              KeyIdentifier [
                0000: 71 DF 38 AD 69 03 CE 06  F9 92 5D BF CA 09 6C 2E  q.8.i.....)....
                0010: 83 CB 7F CD                                ....
              ]
            ]
          }
        ]
      }
    ]
  }
}
"extensions": {
  <no extension>
}

```

Figura 5. 29 Logarea mesajului de handshake de catre client – prezentarea certificatului.

Capitolul 5.9 Testarea aplicatiilor ClassFileServer si SSLClientAuth

Pentru a testa cele două aplicații, le vom porni și vom introduce diferite intrări de test

Exemplu de intrare: `http://localhost:9000/test_file.txt`

```

Introdu portul pe care sa asculte server-ul: 9000
Alege protocolul folosit (TLS petru comunicatie securizata, enter pentru comunicatie in plain text):
Server-ul porneste...
Asculta pe port-ul: 9000

```

Figura 5. 30 Pornirea aplicatiei “ClassFileServer” cu telecomunicatie necriptata.

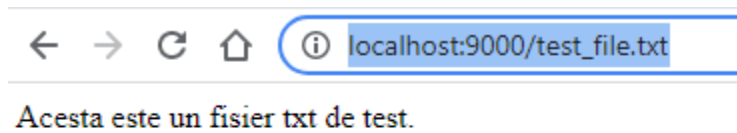


Figura 5. 31 Accesarea unui fisier de test prin HTTP

De această dată, serverul a pornit cu protocolul “TLS” setat.

```
Introdu portul pe care sa asculte server-ul: 9000
Alege protocolul folosit (TLS petru comunicatie securizata, enter pentru comunicatie in plain text): TLS
Server-ul foloseste ClientAuthentication (true pentru da, false pentru nu): false
Server-ul porneste...
Asculta pe port-ul: 9000
```

Figura 5. 32 Pornirea aplicatiei “ClassFileServer” cu comunicatie incryptata

Aceasta înseamnă că acum serverul nostru nu ar mai trebui să folosească protocolul “HTTP” ci să permită comunicația sigură.

Acesta este mesajul browser-urului când accesăm: http://localhost:9000/test_file.txt



This page isn't working

localhost sent an invalid response.

ERR_INVALID_HTTP_RESPONSE

Reload

Figura 5. 33 Serverul nu mai accepta “HTTP” ci doar “HTTPS”

Serverul nostru nu ar mai trebui să poată permite conexiuni folosind “HTTP”, dar ar trebui să funcționeze accesând: https://localhost:9000/test_file.txt

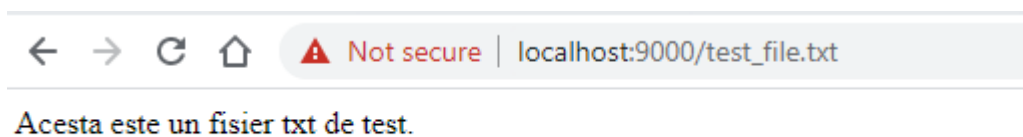


Figura 5. 34 Comunicatie nesigura cand folosim certificate care nu sunt de incredere

Chiar dacă am trecut pe o comunicație criptată, încă primim mesajul de avertizare “Not secure” din partea browser-ului. Acest lucru se poate întâmpla când certificatul nu mai este valid, sau este “self signed” ca și în cazul certificatelor folosite în realizarea acestui proiect, sau acesta este semnat dar nu este considerat de încredere, nefăcând parte dintr-un container de chei pe care serverul să îl considere de încredere.

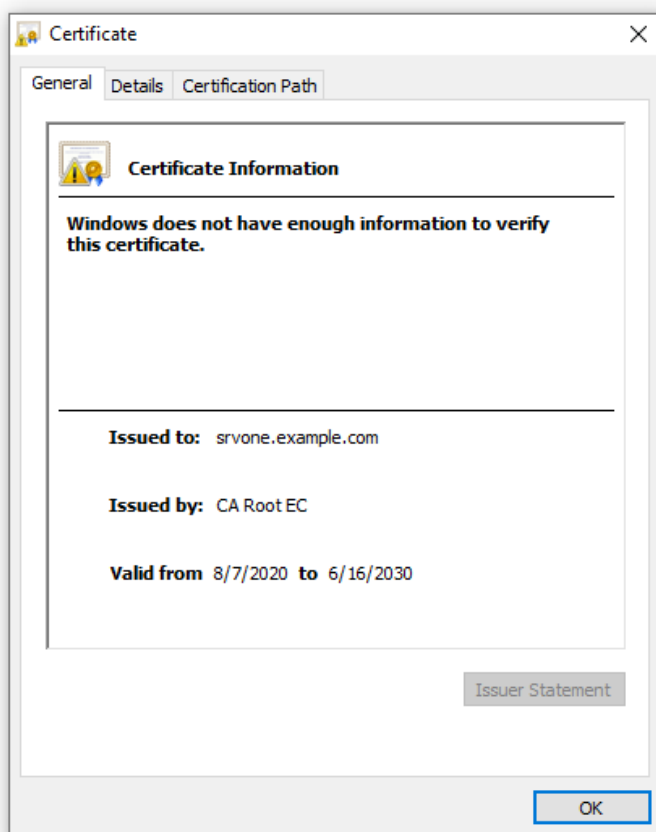


Figura 5. 35 Certificat nevalid

Conexiunea sigură prin browser funcționează. Testăm și “SSLSocketClientAuth”, care ar trebui să se conecteze la server din Java, să citească numele fișierului din consola și apoi să primească răspunsul cu conținutul fișierului în consola de log.

```
Introdu hostname-ul la care sa ne conectam: localhost
Introdu port-ul pe care asculta server-ul: 9000
Introdu numele fisierului pe care vrei sa-l accesezi: test_file.txt
Logam mesajul (true pentru da, false pentru nu): false
HTTP/1.0 200 OK
Content-Length: 34
Content-Type: text/html

Acesta este un fisier txt de test.

Process finished with exit code 0
|
```

Figura 5. 36 Afisarea fisierului in consola

Opțiunea de autentificarea clientului ar trebui să respingă cererile venite din partea clienților care prezintă un certificat care nu este de încredere.

Astfel, am create un nou “keystore”

```
C:\Bogdan\Facultate\Licenta\openssl>keytool -genkey -alias clientAuth -keyalg RSA -keystore clientAuth.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: localhost.com
What is the name of your organizational unit?
[Unknown]: Research
What is the name of your organization?
[Unknown]: ETTI
What is the name of your City or Locality?
[Unknown]: Bucharest
What is the name of your State or Province?
[Unknown]: Romania
What is the two-letter country code for this unit?
[Unknown]: RO
Is CN=localhost.com, OU=Research, O=ETTI, L=Bucharest, ST=Romania, C=RO correct?
[no]: yes
```

Figura 5. 37 Crearea “keystore-ului” “clientAuth”

Extragem certificatul din “keystore-ul” creat

```
C:\Bogdan\Facultate\Licenta\openssl>keytool -export -alias clientAuth -keystore clientAuth.jks -rfc -file clientAuth.cert
Enter keystore password:
Certificate stored in file <clientAuth.cert>
```

Figura 5. 38 Extragerea certificatului

Modificam proprietatile de sistem din Java utilizand:

```
try {
    SSLSocketFactory factory = null;
    try {
        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase = "tlsanalysis".toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

        ks.load(new FileInputStream( name: "src/credentiale/tls_analysis.jks"), passphrase);
```

Figura 5. 39 Modificarea proprietatilor de sistem

Modificam cu noul container de chei

```
try {
    SSLSocketFactory factory = null;
    try {
        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase = "openssl22".toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

        ks.load(new FileInputStream( name: "src/credentiale/clientAuth.jks"), passphrase);

        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), tm: null, random: null);
```

Figura 5. 40 Modificarea noului container de chei

Testăm conexiunea client-server folosind opțiunea de “clientAuthentication” setat.

Când încercam să rulăm aplicația “ClassFileServer” cu opțiunea de “clientAuthentication” setat, primim o eroare “Empty Certificate Chain” deoarece fie nu a fost semnat de un CA, fie nu este valid. Pentru a testa totuși aplicația și cu această opțiune setată, trebuie să extragem certificatul din containerul de chei și apoi să încărcăm acest certificate într-un “trustore”.

```

C:\Windows\system32>keytool -import -alias clientAuth -cacerts -file "C:\Bogdan\Facultate\Licenta\src\credentiale\clientAuth.cert"
Enter keystore password:
Owner: CN=localhost.com, OU=Research, O=ETTI, L=Bucharest, ST=Romania, C=RO
Issuer: CN=localhost.com, OU=Research, O=ETTI, L=Bucharest, ST=Romania, C=RO
Serial number: 26fab0d6
Valid from: Thu Sep 09 22:08:11 EEST 2021 until: Wed Dec 08 21:08:11 EET 2021
Certificate fingerprints:
    SHA1: 8E:E3:B6:65:88:B8:E0:DA:B1:A5:77:06:23:76:E3:EC:14:15:88:5D
    SHA256: 9E:23:B1:17:61:41:74:B3:2A:1D:91:18:41:E9:FB:57:7F:9A:68:91:A1:EC:02:7A:2D:FD:25:F3:92:DE:DD:DC
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B8 7A 50 D8 A8 FE 82 11  10 59 5D 3F 57 7C 35 B2  ..zP.....Y]?W.5.
0010: 2D 90 82 98                ....
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore

C:\Windows\system32>

```

Figura 5. 41 Adaugarea unui certificat intr-un "truststore"

Concluzii și Direcții de Viitor

Proiectul și-a propus implementarea și analiza protocolului TLS. În vederea acestui scop, au fost implementate trei aplicații perechi de client și server care să comunice între ele, fiecare având scopuri diferite. Astfel, proiectul a început prin prezentarea unei situații în care două aplicații comunică în mod nesecurizat. În acest caz am demonstrat de ce este de evitat această situație prin analiza capturilor de trafic și citind în text clar informațiile transmise capturate de Wireshark.

În continuare am dorit prezentarea modului în care implementarea protocolului se poate face în Java. Am implementat două versiuni ale protocolului pentru a putea analiza diferențele între acestea. Și aici am folosit utilitarul Wireshark pentru a putea vedea procesele din timpul criptării.

În cele din urmă am demonstrat implementarea protocoalelor HTTP și HTTPS, punând în evidență un nou mod de a lucra cu containerele de chei. Această aplicație oferă și posibilitatea filtrării conexiunilor. Putem pe baza certificatului prezentat să respingem sau să acceptăm diferite conexiuni.

Principalele funcționalități de care dispun aplicațiile create sunt:

- Folosirea de comunicații necriptate:
- Alegerea versiunii de protocol TLS folosite
- Opțiunea de logare a mesajelor
- Încărcarea de fișiere și accesarea acestora pe internet
- Posibilitatea filtrării conexiunii pe baza certificatelor

Posibilele îmbunătățiri ale aplicațiilor sunt:

- Folosirea unor adrese pe care și alți utilizatori din afara rețelei noastre locale să o poată accesa.
- Crearea de certificate semnate de un CA
- Implementarea de aplicații care să suporte schimbul de mesaje între mai multe conexiuni, aplicațiile EchoClient și Server prezentate în proiect, suportă schimbul de mesaje doar într-o singură sesiune.
- Pentru ultimele aplicații de test în care am folosit HTTPS, ar putea fi îmbunătățite prin afișarea unor paginii HTML.

Bibliografie

- [1]. Curs "Security Protocols", ETTI, titular de curs Conf. Dr. Ing. Octavian Catrina, accesat la data de 4.07.2021
- [2]. Cursuri Programare Orientată pe Obiecte, ETTI, seria F, titular curs Ş.I.dr.ing Radu Hobincu, accesat la data: 10.07.2021
- [3]. Curs Arhitecturi şi protocoale de Comunicaţii, ETTI, seria D, titular curs Conf. Dr. Ing. Octavian Catrina, accesat la data de: 13.07.2021
- [4]. "Bulletproof SSL and TLS" realizată de către Ivan Ristic, editura Feisty Duck
- [5]. "THE SSL/TLS ULTIMATE GUIDE E-BOOK" https://aboutssl.org/download-ssl-guide-ebook/?utm_source=AboutSSL&utm_medium=PromoBar&utm_campaign=eBookDownload
- [6]. java server: <https://whatis.techtarget.com/definition/server>, accesat la data: 14.07.2021
- [7]. socket: <https://medium.com/swlh/understanding-socket-connections-in-computer-networking-bac304812b5c>, accesat la data: 16.07.2021
- [8]. ServerSocket: <https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html>, accesat la data: 19.07.2021
- [9]. keytool: <https://docs.oracle.com/en/java/javase/13/docs/specs/man/keytool.html>, accesat la data: 21.07.2021
- [10]. SSLServerSocket: <https://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLServerSocket.html>, accesat la data: 22.07.2021
- [11]. SSLServerSocketFactory: <https://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLServerSocketFactory.html>, accesat la data de: 22.07.2021
- [12]. openssl: <https://www.openssl.org/docs/man1.0.2/man1/openssl-req.html>, accesat la data: 23.07.2021
- [13]. TlsAlert: <https://sites.google.com/site/tlssloverview/ssl-tls-protocol-layers/handshake-layer/alert-protocol>, accesat la data: 24.07.2021

Anexe

EchoServer.java package

```
com.tls.analysis.echo.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class EchoServer {
    private final static int port = 9000;

    public static void main(String[] args) {
        System.out.println("Server-ul porneste...");
        System.out.println("Ascultam pe port-ul: " +
port);

        try {
            ServerSocket serverSocket = new
ServerSocket(port);
            Socket client = serverSocket.accept();

            System.out.println("Client conectat!");

            BufferedReader bufferedReader = new
BufferedReader(new
InputStreamReader(client.getInputStream()));
            PrintWriter output = new
PrintWriter(client.getOutputStream(), true);
            String line;
            while ((line = bufferedReader.readLine()) !=
null) {
                System.out.println("De la Client: " + line);
                output.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

EchoClient.java

```
package com.tls.analysis.echo.client;

import javax.net.SocketFactory;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class EchoClient {

    private final static int port = 9000;
    private final static String host = "localhost";

    public static void main(String[] args) {

        try {
            Socket socket = new Socket(host, port);
            PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
            BufferedReader buffer = new
BufferedReader(new
InputStreamReader(socket.getInputStream()));
            System.out.println("Conectat la server cu
succes!");
            Scanner scanner = new Scanner(System.in);
            while(true) {
                System.out.println("Spune ceva: ");
                String input = scanner.nextLine();
                if("exit".equalsIgnoreCase(input)) {
                    break;
                }
                out.println(input);
                String response = buffer.readLine();

            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Server.java

```

package com.tls.analysis.echoTLS.EchoSSLServer;

import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.*;
import java.util.Scanner;

public class Server {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int port;
        String tls_version = "";

        System.out.print("Introdu port-ul pe care
asculta server-ul: ");
        port = scanner.nextInt();
        scanner.nextLine();

        System.out.println("Server-ul asculta pe port-
ul: " + port);

        System.out.print("Ce versiune de protocol
folosim (TLSv1.2, TLSv1.3): ");
        tls_version = scanner.nextLine();

        System.setProperty("javax.net.ssl.keyStore",
"src/credentiale/tls_analysis.jks");

        System.setProperty("javax.net.ssl.keyStorePasswor
d", "tlsanalysis");
        System.setProperty("javax.net.ssl.trustStore",
"src/credentiale/tls_analysis.jks");

        System.setProperty("javax.net.ssl.trustStorePasswo
rd", "tlsanalysis");

        try {
            System.out.println("Astepam conexiuni...");
            SSLServerSocket sslServerSocket =
(SSLServerSocket)

```

```

SSLServerSocketFactory.getDefault().createServerS
ocket(port);
            SSLSocket client = (SSLSocket)
sslServerSocket.accept();
            if(tls_version.equals("TLSv1.3")) {
                client.setEnabledProtocols(new String[]
{"TLSv1.3"});
                client.setEnabledCipherSuites(new String[]
{"TLS_AES_128_GCM_SHA256"});
            } else {
                client.setEnabledProtocols(new
String[]{"TLSv1.2"});
            }
            System.out.println("Connection established
with: " + client.getInetAddress());

            BufferedReader bufferedReader = new
BufferedReader(new
InputStreamReader(client.getInputStream()));
            PrintWriter output = new
PrintWriter(client.getOutputStream(), true);
            String line;
            while ((line = bufferedReader.readLine()) !=
null) {
                System.out.println("De la client: " + line);
                output.println(line);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

EchoSSLClient.java

```

package com.tls.analysis.echoTLS.EchoSSLClient;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLServerSocketFactory;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Scanner;

public class EchoSSLClient {

```

```

public static void main(String[] args) {

    System.setProperty("javax.net.ssl.keyStore",
"src/credentiale/tls_analysis.jks");

    System.setProperty("javax.net.ssl.keyStorePasswor
d", "tlsanalysis");
    System.setProperty("javax.net.ssl.trustStore",
"src/credentiale/tls_analysis.jks");

    System.setProperty("javax.net.ssl.trustStorePasswo
rd", "tlsanalysis");

    String hostname;
    int port;
    boolean enable_log = false;
    String log_type = "";
    String log_options = "";
    String log_ssl_parameters = "";

    Scanner scanner = new Scanner(System.in);
    System.out.print("Introdu host-ul server-ului:
");
    hostname = scanner.nextLine();

    System.out.print("Introdu port-ul pe care
asculta server-ul: ");
    port = scanner.nextInt();
    scanner.nextLine();

    String line;
    System.out.print("Logam mesajul (true pentru
da, false pentru nu: ");
    line = scanner.nextLine();

    if(line.equals("true")) {
        System.out.print("Ce log sa folosim (all sau
ssl): ");
        log_type = scanner.nextLine();
        System.out.print("Ce optiune sa alegem
pentru log(data, verbose): ");
        log_options = scanner.nextLine();
        if(log_type.equals("ssl")) {
            System.out.println("Optiuni pentru logare
de tip ssl: ");
            System.out.print("record,      handshake,
keygen, session, defaultctx, sslctx, sessioncache,
keymanager, trustmanager");

```

```

        System.out.print("Alege optiunea: ");
        log_ssl_parameters = scanner.nextLine();
        System.setProperty("javax.net.debug",
log_type + " " + log_options + " " +
log_ssl_parameters);

        } else {
            System.setProperty("javax.net.debug",
log_type + " " + log_options);
        }

    }

    System.out.println("Conectare  catre: " +
hostname + " pe port-ul: " + port);

    try {
        SSLSocket client = (SSLSocket)
SSLSocketFactory.getDefault().createSocket(hostna
me, port);
        System.out.println("Conectare cu succes!");

        PrintWriter out = new
PrintWriter(client.getOutputStream(), true);
        BufferedReader bufferedReader = new
BufferedReader(new
InputStreamReader(client.getInputStream()));
        while (true) {
            System.out.println("Spune ceva prin SSL:
");
            String input = scanner.nextLine();
            if("exit".equalsIgnoreCase(input)) {
                break;
            }
            out.println(input);
            String response =
bufferedReader.readLine();
            System.out.println("Server: " + response);
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

ClassServer.java

```
package
com.tls.analysis.client.authentication.servers;

import java.io.*;
import java.net.*;
import javax.net.*;

public abstract class ClassServer implements
Runnable {

    private ServerSocket server = null;

    /**
     * Construiește un obiect de tip ClassServer pe
     baza unui server socket
     * Obține bitii unui fisier prin metoda "getBytes"
     */

    protected ClassServer(ServerSocket ss) {
        server = ss;
        newListener();
    }

    /** Intoarce un array de biti care contine bitii
    fisierului cerut
     * Poate arunca 2 exceptii:
     * FileNotFoundException - daca nu a putut fi
    accesat fisierul cerut
     * IOException - daca sunt erori in citirea clasei
     */

    public abstract byte[] getBytes(String path)
    throws IOException, FileNotFoundException;

    /** Thread-ul care accepta conexiuni catre server,
    parseaza header-ul pentru a obtine numele fisierului
     * si trimite inapoi bitii fisierului (sau o eroare daca
    fisierul nu a putut fi gasit sau raspunsul a fost
    malformat
     */

    public void run() {
        Socket socket;
```

```
// Accepta o conexiune
try {
    socket = server.accept();
} catch (IOException e) {
    System.out.println("Class Server a picat...");
    e.printStackTrace();
    return;
}

// Creem un nou thread pentru a accepta
conexiunea
newListener();

try {
    OutputStream rawOut =
socket.getOutputStream();
    PrintWriter out = new PrintWriter(new
BufferedWriter(new
OutputStreamWriter(rawOut)));

    try {
        // Obține calea către fisier din header
        BufferedReader in = new
BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String path = getPath(in);

        // Obține bitii
        byte[] bytewords = getBytes(path);

        // Trimitem bitii ca si raspuns - presupunem
HTTP/1.0 sau o varianta mai recenta

        try {
            out.print("HTTP/1.0 200 OK\r\n");
            out.print("Content-Length: " +
bytewords.length + "\r\n");
            out.print("Content-Type:
text/html\r\n\r\n");
            out.flush();
            rawOut.write(bytewords);
            rawOut.flush();
        } catch (IOException ie) {
            ie.printStackTrace();
            return;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

        // Scriem mesajul de eroare
        out.println("HTTP/1.0 400 " +
e.getMessage() + "\r\n");
        out.println("Content-Type:
text/html\r\n\r\n");
        out.flush();
    }
} catch (IOException ex) {
    System.out.println("Error writing response: " +
ex.getMessage());
    ex.printStackTrace();
} finally {
    try {
        socket.close();
    } catch (IOException e) {

    }
}

// Creeam un nou thread pe care sa ascultam
private void newListener() { (new
Thread(this)).start(); }

// Intoarce calea catre fisierul obtinut din
parsarea header-ului

private static String getPath(BufferedReader in)
throws IOException {
    String line = in.readLine();
    String path = "";

    if(line.startsWith("GET /")) {
        line = line.substring(5, line.length()-1).trim();
        int index = line.indexOf(' ');
        if(index != -1) {
            path = line.substring(0, index);
        }
    }

    // Parcurgem restul header-ului
    do {
        line = in.readLine();
    } while ((line.length() != 0) && (line.charAt(0) !=
'\r') && (line.charAt(0) != '\n'));

    if (path.length() != 0) {
        return path;

```

```

    } else {
        throw new IOException("Header
Malformat");
    }
}
}

```

ClassFileServer.java

```

package
com.tls.analysis.client.authentication.servers;

```

```

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.*;
import java.security.*;
import java.util.Scanner;
import javax.net.*;
import javax.net.ssl.*;

```

```

public class ClassFileServer extends ClassServer {

```

```

    private String docroot;
    private static int DefaultServerPort = 2001;

```

```

    /** Construieste un obiect de tip ClassFileServer
    * Parametrii: calea catre fisier
    */

```

```

    public ClassFileServer(ServerSocket ss, String
docroot) {
        super(ss);
        this.docroot = docroot;
    }

```

```

    /** Intoarce un array de biti ce contine bitii
    fisierului cerut
    * Arunca FileNotFoundException daca fisierul nu
    a putut fi incarcat.
    */

```

```

    public byte[] getBytes(String path) throws
IOException {
        System.out.println("reading: " + path);
        File f = new File(docroot + File.separator +
path);
        int length = (int)(f.length());
        if (length == 0) {
            throw new IOException("Dimensiunea
fisierului este 0: " + path);
        } else {
            FileInputStream fin = new FileInputStream(f);
            DataInputStream in = new
DataInputStream(fin);

            byte[] bytcodes = new byte[length];
            in.readFully(bytcodes);
            return bytcodes;
        }
    }

/** Metoda Main care creeaza server-ul.
 *
 */

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int port;
    String docroot = "src/test_file/";
    String protocol;
    boolean clientauth;

    System.out.print("Introdu portul pe care sa
asculte server-ul: ");
    port = in.nextInt();
    in.nextLine();

    String line;
    System.out.print("Alege protocolul folosit (TLS
petru comunicatie securizata, enter pentru
comunicatie in plain text): ");
    line = in.nextLine();
    if (line.equals("TLS")) {
        protocol = "TLS";
        System.out.print("Server-ul foloseste
ClientAuthentication (true pentru da, false pentru
nu): ");
        line = in.nextLine();

```

```

    } else {
        protocol = "PlainSocket";
    }

    System.out.println("Server-ul porneste...");
    System.out.println("Asculta pe port-ul: " +
port);

    if (line.equals("true")) {
        clientauth = true;
    } else {
        clientauth = false;
    }

    try {
        ServerSocketFactory ssf =
ClassFileServer.getServerSocketFactory(protocol);
        ServerSocket ss =
ssf.createServerSocket(port);

        if (clientauth == true) {

            ((SSLServerSocket)ss).setNeedClientAuth(true);
        }
        new ClassFileServer(ss, docroot);
    } catch (IOException e) {
        System.out.println("Eroare pornind server-
ul: " + e.getMessage());
        e.printStackTrace();
    }
}

private static ServerSocketFactory
getServerSocketFactory(String protocol) {
    if (protocol.equals("TLS")) {
        SSLServerSocketFactory ssf = null;
        try {
            // Setam key manager-ul ca sa faca
autentificarea server-ului.
            SSLContext ctx;
            KeyManagerFactory kmf;
            KeyStore ks;
            char[] passphrase =
"password".toCharArray();

            ctx = SSLContext.getInstance("TLS");

```

```

        kmf
KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

        ks.load(new
FileInputStream("src/credentiale/OC/keys/server_k
s.pkcs12"), passphrase);
        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), null, null);

        ssf = ctx.getServerSocketFactory();

        return ssf;
    } catch (Exception e) {
        e.printStackTrace();
    }
    } else {
        return ServerSocketFactory.getDefault();
    }
    return null;
}
}

```

SSLClientAuth.java

```

package
com.tls.analysis.client.authentication.client;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
import java.security.KeyStore;
import java.util.Scanner;

public class SSLSocketClientAuth {

    public static void main(String[] args) {

        String host = "";
        int port = -1;
        String path = "";
        boolean enable_log = false;
        String log_type = "";
        String log_options = "";
        String log_ssl_parameters = "";

```

```

        Scanner scanner = new Scanner(System.in);

        System.out.print("Introdu hostname-ul la care
sa ne conectam: ");
        host = scanner.nextLine();

        System.out.print("Introdu port-ul pe care
asculta server-ul: ");
        port = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Introdu numele fisierului pe
care vrei sa-l accesezi: ");
        path = scanner.nextLine();

        String line;
        System.out.print("Logam mesajul (true pentru
da, false pentru nu): ");
        line = scanner.nextLine();

        if(line.equals("true")) {
            enable_log = true;
            System.out.print("Ce log sa folosim (all sau
ssl): ");
            log_type = scanner.nextLine();
            System.out.print("Ce optiune sa alegem
pentru log(data, verbose): ");
            log_options = scanner.nextLine();
            if(log_type.equals("ssl")) {
                System.out.println("Optiuni pentru logare
de tip ssl: ");
                System.out.print("record,      handshake,
keygen, session, defaultctx, sslctx, sessioncache,
keymanager, trustmanager");
                System.out.print("Alege optiunea: ");
                log_ssl_parameters = scanner.nextLine();
                System.setProperty("javax.net.debug",
log_type + ":@" + log_options + ":@" +
log_ssl_parameters);

            } else {
                System.setProperty("javax.net.debug",
log_type + ":@" + log_options);
            }
        }
    }
}

```



```

try {
    SSLSocketFactory factory = null;
    try {
        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        char[] passphrase =
"openssl22".toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf =
KeyManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");

        ks.load(new
FileInputStream("src/credential/clientAuth.jks"),
passphrase);

        kmf.init(ks, passphrase);
        ctx.init(kmf.getKeyManagers(), null, null);

        factory = ctx.getSocketFactory();
    } catch (Exception e) {
        throw new IOException(e.getMessage());
    }
    SSLSocket socket = (SSLSocket)
factory.createSocket(host, port);

    socket.startHandshake();

```

```

PrintWriter out = new PrintWriter(new
BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())));
out.println("GET /" + path + " HTTP/1.0");
out.println();
out.flush();

if(out.checkError()) {
    System.out.println("SSLSocketClient:
java.io.PrintWriter error");
}

BufferedReader in = new
BufferedReader(new
InputStreamReader(socket.getInputStream()));

String inputLine;
while((inputLine = in.readLine()) != null) {
    System.out.println(inputLine);
}

in.close();
out.close();
socket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}

```

