

Universitatea "Politehnica" din București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Analiză 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 Motivatia lucrarii	14
CAPITOLUL 2. Sinteza arhitecturii si procedurile protocolului TLS.....	15
Capitolul 2.1 Arhitectura TLS.....	15
Capitolul 2.2 Protocolul Handshake	16
Capitolul 2.3 Protocolul TLS Record	16
Capitolul 2.4 Protocolul TLS Alert.....	16
Capitolul 2.5 Protocolul Change Cipher Spec	16
Capitolul 2.6 Autentificarea.....	16
Capitolul 2.7 Incriptia	17
Capitolul 2.8 Incriptia Public-key.....	17
Capitolul 2.9 – Incriptia simetrica sau Bulk	17
Capitolul 2.10 – Algoritmi, Ciphers si Cipher suites.....	18
Capitolul 2.11 – Infrastructura cheii publice PKI	18
Capitolul 2.12 Protocolul TLS 1.3 in comparatie cu TLS 1.2	19
Capitolul 3 Folosirea certificatelor	20
Capitolul 3.1 Certificate si CA	20
Capitolul 3.2 Generarea de containere de chei in format JKS.....	20
Capitolul 3.3 Generarea de certificate folosind “OpenSSL”	22
CAPITOLUL 4. STUDIUL DE CAZ.....	24
Capitolul 4.1 Descrierea solutiilor propuse	24
Capitolul 4.2 Aplicatia EchoServer.....	25
Capitolul 4.3 Aplicatia EchoClient.....	26
Capitolul 4.4 Aplicatia Server	27
Capitolul 4.5 Aplicatia EchoSSLClient	29
Capitolul 4.6 Aplicatiile ClassServer si ClassFileServer	31
Capitolul 4.6.1 Aplicatia ClassServer	32
Capitolul 4.6.2 Aplicatia ClassFileServer	34
Capitolul 4.7 Aplicatia SSLSocketAuth	37
Capitolul 5 Testare/Utilizare.....	41
Capitolul 5.1 Pornirea aplicatiilor in IDE-ul IntelliJ	41
Capitolul 5.2 Utilizarea Wireshark.....	42

Capitolul 5.3 Functionarea aplicatiilor EchoServer si EchoClient	43
Capitolul 5.4 Pornirea aplicatiilor EchoServer si EchoClient.	44
Capitolul 5.5 Functionarea aplicatiilor EchoSSLServer si EchoSSLClient	45
Capitolul 5.6 Utilizarea tool-ului Wireshark pentru a captura traficul dintre EchoClient si EchoServer.....	46
Capitolul 5.7 Demonstrarea functionarii aplicatiei Server.	46
Capitolul 5.8 Testarea aplicatiei EchoSSLClient.....	50
Capitolul 5.9 Testarea aplicatiilor ClassFileServer si SSLClientAuth	51
Concluzii și Direcții de Viitor	57
Bibliografie.....	58
Anexe	59

Listă de figuri

Figura 2. 1 Arhitectura protocolului TLS.....	15
Tabel 1 Etapele Handshake-ului al TLS 1.3	19
Figura 3. 1 Mesajul help al keytool.....	21
Figura 3. 2 Parola keystore	21
Figura 3. 3 Completarea chestionarului keystore-ului	22
Figura 3. 4 Confirmarea informatiilor introduse in keystore.....	22
Figura 3. 5 Generare certificat openssl.....	23
Figura 5. 1 Pornirea aplicatiei EchoServer	41
Figura 5. 2 Confirmarea pornirii aplicatiei EchoServer.....	41
Figura 5. 3 Confirmarea pornirii aplicatiei EchoClient.....	42
Figura 5. 4 Confirmarea conexiunii clientului.....	42
Figura 5. 5 Utilizarea Wireshark pe traficul local.....	42
Figura 5. 6 Aplicarea filtrului pentru port in Wireshark	43
Figura 5. 7 Trimiterea unui mesaj folosind EchoClient.....	43
Figura 5. 8 Afisarea mesajului trimis	43
Figura 5. 9 Folosirea cuvântului “exit” pentru terminarea aplicatiei EchoClient	43
Figura 5. 10 Pornirea aplicatiei EchoServer	44
Figura 5. 11 Confirmarea pornirii aplicatiei EchoServer.....	44
Figura 5. 12 Confirmarea pornirii aplicatiei EchoClient.....	45
Figura 5. 13 Confirmarea conexiunii clientului.....	45
Figura 5. 14 Trimiterea unui mesaj folosind EchoClient.....	45
Figura 5. 15 Afisarea mesajului trimis	45
Figura 5. 16 Captura trafic Wireshark dintre EchoServer si EchoClient	46
Figura 5. 17 Pornirea aplicatiei Server cu negocierea versiunii TLS 1.3	46
Figura 5. 18 Pornirea aplicatiei EchoSSLClient fara logare cu negocierea versiunii TLS 1.3.....	47
Figura 5. 19 Captura de trafic asupra negocierii versiunii TLS 1.3.....	47
Figura 5. 20 Pornirea aplicației Server cu negocierea versiunii TLS 1.2	48
Figura 5. 21 Captură de trafic asupra negocierii versiunii TLS 1.2.....	48
Figura 5. 22 Folosirea OpenSSL pentru verificarea versiunii de TLS.....	49
Figura 5. 23 OpenSSL conectare cu versiunea 1.2 la un server 1.3	50
Figura 5. 24 Eroare aplicatie “Server” cand versiunile negociate difera.....	50
Figura 5. 25 Logarea mesajului de handshake de catre client – negocierea de cipher suites	51
Figura 5. 26 Logarea mesajului de handshake de catre client – negocierea versiunii protocolului.....	51
Figura 5. 27 Logarea mesajului de handshake de catre client – prezentarea certificatului.....	51
Figura 5. 28 Pornirea aplicatiei “ClassFileServer” cu telecomunicatie necriptata.	51
Figura 5. 29 Accesarea unui fisier de test prin HTTP	52
Figura 5. 30 Pornirea aplicatiei “ClassFileServer” cu comunicatie incriptata.....	52
Figura 5. 31 Serverul nu mai accepta “HTTP” ci doar “HTTPS”	52
Figura 5. 32 Comunicatie nesigura cand folosim certificate care nu sunt de incredere	53
Figura 5. 33 Certificat nevalid.....	53
Figura 5. 34 Afisarea fisierului in consola.....	54
Figura 5. 35 Crearea “keystore-ului” “clientAuth”	54
Figura 5. 36 Extragerea certificatului	54

Figura 5. 37 Modificarea proprietatilor de sistem.....	55
Figura 5. 38 Modificarea noului container de chei.....	55
Figura 5. 39 Adaugarea unui certificat intr-un "truststore"	56

Listă de tabele

Tabel 1 Etapele Handshake-ului al TLS 1.3	19
---	----

Lista acronimelor

CA – Certificate Authority

CMD – Command Line Interpreter

CRL – Certificate Revocation List

FTPS – File Transfer Protocol Secure

HTTP - Hyper Text Transfer Protocol;

HTTPS - Hyper Text Transfer Protocol Secure;

IDE – Integrated Development Environment

MAC -Message authentication code

OCSP – Online Certificate Status Protocol

PKI – Public Key Infrastructure

SMTP – Simple Mail Transfer Protocol

SSL – Secure Sockets Layer

TLS - Transport Layer Security

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 protocoale 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 plăți 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 servește rolurile securității și anume incryptarea 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.

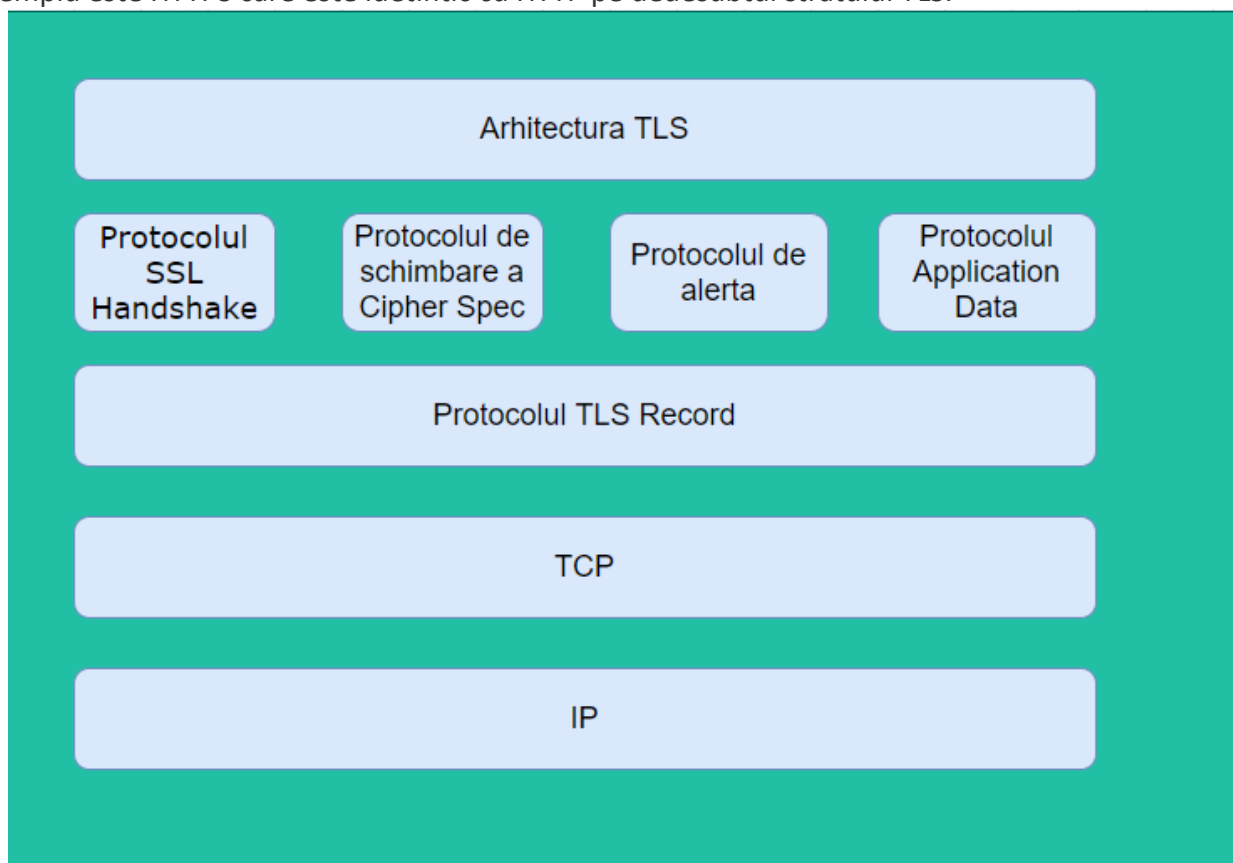


Figura 2. 1 Arhitectura protocolului TLS

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, criptarea 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 bază 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 Handshake

Protocolul Handshake este numele tehnic procesului dat care stabilește o conexiune HTTPS. Aproape toată munca este făcută în acest protocol. Principalul 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 incipție;
- Autentificarea certificatului TLS/SSL
- Schimbul sau generarea unei chei de sesiuni.

Capitolul 2.3 Protocolul TLS Record

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

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

Capitolul 2.4 Protocolul TLS Alert

Acest protocol este folosit pentru a semnaliza diferitele schimbări de status ce au loc. O schimbare de status poate conține erori, mesaje invalide sau care nu au putut fi deciptate sau ne anunță că sesiunea s-a încheiat.

În general, acesta se ocupă de semnalarea următoarelor trei tipuri de alerte.

- Avertizare
- Critice
- Fatale

Capitolul 2.5 Protocolul Change Cipher Spec

Este protocolul care informează participanții implicați în procesul de sezurizare a conexiunii că algoritmul folosit pentru incipție fost schimbat. Acest protocol folosește diferite metode criptografice în schimbul de date dintre client și server.

Capitolul 2.6 Autentificarea

Autentificarea este o formă simplă de verificarea a identitatii. 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 bazata pe certificate face parte din protocolul “TLS”. În acest caz, client-ul semnează digital o bucată de date generata 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:

1. Aplicația de client manageriază o bază de date care constă din chei private publicate în toate certificatele emise pentru acel client. La prima accesare a bazei de date va fi nevoie de introducerea parolei.
2. 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.
3. Pentru a face autentificarea utilizatorului, serverul folosește atât datele semnate cât și certificatul.

Capitolul 2.7 Incripția

Incripția este procedeul prin care informația este transformată într-un cod secret pentru a ascunde informația mesajului folosind diferiți algoritmi matematici.

Capitolul 2.8 Incripția Public-key

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 mesajele iar cheia privată este folosită pentru a le decripta.

La rândul ei, incripția 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 Bulk

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 alfabeturi.

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 și Cipher suites

“Cipher suits” reprezintă o suită de algoritmi folosiți pentru a securiza comunicațiile pe internet.

Numele fiecărui “cipher suite” este reprezentativ, deoarece este format din algoritmii folosiți. Un “cipher suite” obișnuit arată în felul următor:

ECDHE-ECDSA-AES128-GCM-SHA256

Și este o combinație de algoritmi și protocoale.

- ECDHE – algoritmul de schimbare chei;
- ECDSA – algoritm de autentificare;
- AES128 – algoritm tip de criptare simetrică;
- SHA256 – algoritm MAC.

MAC este unul din algoritmii folosiți pentru verificarea și autentificarea mesajului primit.

Capitolul 2.11 – Infrastructura cheii publice PKI

“PKI” reprezintă un set de reguli, proceduri și politici în scopul de a crea, distribui, înmagazina, 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 in 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.

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

Tabel 1 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 răspunde 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	În cele din urmă, clientul verifică certificatul serverului, generează cheia sa și trimite mesajul "Client Finished" și criptarea datelor poate fi pornită

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

Algoritmii eliminați în noua versiune:

- Schimbul de chei RSA
- RC4 Stream Cipher
- CBC (Block) Mode Ciphers
- Algoritmul MD5
- Funcția Hash 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, acesta semnează că 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 în 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 următoarele operații:

- -genkey – generează o pereche de chei;
- -alias – Atribuie un alias cheii;
- -keyalg – Algoritmul folosit în 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\L
icenta\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 – genereaza o cerere de crearea a certificatului;
- -newkey:rsa:2048 – algoritmul pentru generarea cheii;
- -x509 – generează un certificat de tip “self signed”;
- -keyout - fisierul care sa scrie cheia;
- -out – fisierul 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, putem evidenția configurarea protocolului în limbajul Java, alegerea folosirii diferitelor versiuni ale protocolului, diferitelor funcționalități pe care le putem utiliza atunci când comunicăm pe internet și cum putem analiza protocolul folosind capturi de trafic. De asemenea, voi arăta cum putem lucra cu certificate în interiorul unei aplicații.

Pentru a pune în evidență mai bine necesitatea securizării comunicațiilor pe internet și diferențele configurației și implementării 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

Prima pereche de aplicații client și server au scopul de a arăta 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 configurații 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 Aplicatia EchoServer

Începem proiectul prin a ne importa bibliotecile și librăriile folosite în realizarea aplicației.

```
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, definim clasa, setăm variabila "port" care va fi folosită pentru a asculta conexiuni, în cazul acestei aplicații aceasta valoare va fi 9000.

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

Definim clasa main care începe prin a ne confirma ca aplicatia a pornit si ca asculta portul setat prin afisarea unui mesaj in consola.

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

În interiorul acestei clase trebuie să folosim o sintaxă de tip "try" și "catch", deoarece unele din funcțiile apelate vorunca excepții care trebuie prinse.

Urmează să creăm obiectul de tip "ServerSocket" cu portul definit anterior și care ascultă după conexiuni. Aceasta 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!");
```

Obținem mesajul de la client de pe stream-ul de intrare și ne creăm 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 urma, închidem sintaxa try catch, prinzând excepțiile posibile.

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

Capitolul 4.3 Aplicatia EchoClient

Ne importam librariile folosite:

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

Definim 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 mesajele din consolă creăm 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 vă termina.

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

Trimitem mesajul către server și prindem posibilele excepții apărute.

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

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

Capitolul 4.4 Aplicatia Server

Importăm librăriile

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

Definim clasa și funcția main. Creăm un obiect de tip Scanner care să citească mesajul inserat de utilizator în consolă și creăm două variabile care să primească numere întregi, reprezentând portul pe care vă asculta, respectiv un text care să stabilească versiunea protocolului "TLS" folosit.

```
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 ca protocolul să funcționeze trebuie setate proprietățile sistemului pentru a alege ce keystore și truststore vor fi folosiți. Este de menționat faptul că în cazul aplicației, fișierul “tls_analysis.jks” joacă rolul atât de keystore cât și de truststore.

```

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");

```

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 ce au fost create obiectele de tip “SSLServerSocket” și “SSLSocket” putem impune ce variantă de protocol să folosim. Astfel, aplicația verifică varianta de protocol aleasă, și setează această valoare în funcție de informația introdusă prin apelarea funcției “client.setEnabledProtocols” care primește ca argument un String array.

O funcție asemănătoare funcției “setEnabledProtocols”, este “setEnabledCipherSuites” care și aceasta primește ca argument un “String array”.

```

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 consolă se va afișa adresa clientului conectat la server prin funcția `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 ni 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”, ni se va cere să specificăm ce tip logare se va folosi.

Opțiunile de logare a mesajelor sunt după cum urmează:

- all – în cazul în care dorim ca aplicația să logheze toate evenimentele care au loc în timpul rulării aplicației.
- ssl – în cazul în care dorim 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.

Importăm librăriile 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;
```

Definim 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");
    }
}
```

Definim 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 = "";
```

Cream obiectul cu care vom citi mesajele introduse din consolă și verificăm 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, 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);
}

}
```

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

Importăm librăriile și implementăm interfața “Runnable” și definim 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.

Definim funcția getBytes(String) care va întoarce biții fișierului

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

Aceasta poate arunca 2 excepții, FileNotFoundException când fișierul nu este accesibil și IOException dacă 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 sesiunea este încă activă. În această funcție vom implementa tot ce ține de comunicația dintre client și server.

Acceptăm conexiunea cu clientul sau aruncăm 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 citim mesajul de la client și pe baza acestui mesaj apelăm funcția `getPath` care va întoarce numele fișierului cerut de către client și extragem 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[] bytecodes = getBytes(path);
```

Trimitem biții fișierului drept răspuns, respectând standardul protocolului "HTTP"

```
try {
    out.print("HTTP/1.0 200 OK\r\n");
    out.print("Content-Length: " + bytecodes.length + "\r\n");
    out.print("Content-Type: text/html\r\n\r\n");
    out.flush();
    rawOut.write(bytecodes);
    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) {

    }
}
}
```

Definim funcția care creează Thread-ul

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

Definim 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 aplicatia aceasta se foloseste structura clasei “ClassServer” va trebui sa o extindem pe aceasta din urma in aplicatia curenta.

Incepem prin importarea librariilor, extinderea clasei si crearea constructorului. Definim si portul pe care va porni aplicatia implicit, si o cale sub forma unui text, ce reprezinta calea directorului pus la dispozitia 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 fișier
 */

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

```

Definim 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 fișierului este 0: " + path);
    } else {
        FileInputStream fin = new FileInputStream(f);
        DataInputStream in = new DataInputStream(fin);

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

```

Continuăm cu funcția main(), prin a ne declara variabilele folosite. Aplicația va citi din consolă informațiile de utilizare introduse de client pe care le va atribui variabililor 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;
}

```

Continuăm 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, vom cere autentificarea prin apelul funcției `setNeedClientAuth` cu parametrul `"true"`.

```

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

```

Ne creăm 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();
}
}

```

Urmează să definim 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 vom cere autentificarea clientului, vom 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");

```

Incarcam in containerul de chei create certificatul pe care il va folosi serverul si apoi vom returna conexiunea cu configuratiile 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

Importăm librăriile și definim 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 ne creăm 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());
    }
}

```

Creăm conexiunea cu serverul și pornim procesul de “handshake”

```

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

socket.startHandshake();

```

Creăm obiectul care va face scrierea și 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();

```

Afișăm posibilele erori apărute

```

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

```

În cele din urmă create 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.

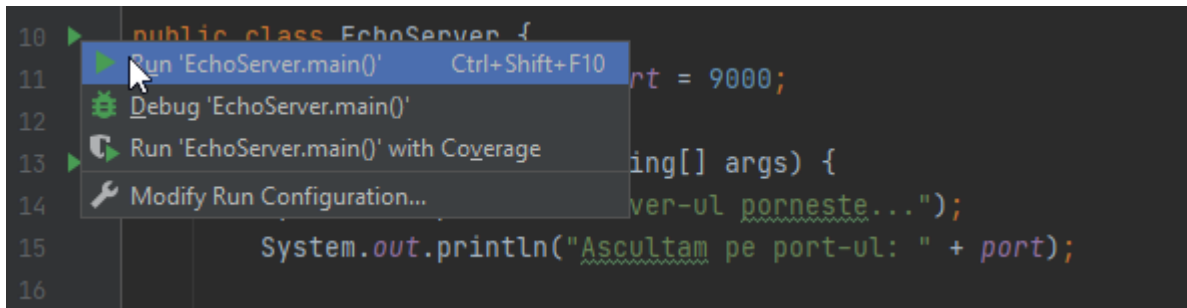


Figura 5. 1 Pornirea aplicatiei EchoServer

După pornirea aplicației în consolă ar trebui să vedem mesaje care ne confirmă că serverul a pornit și că așteaptă conexiuni.

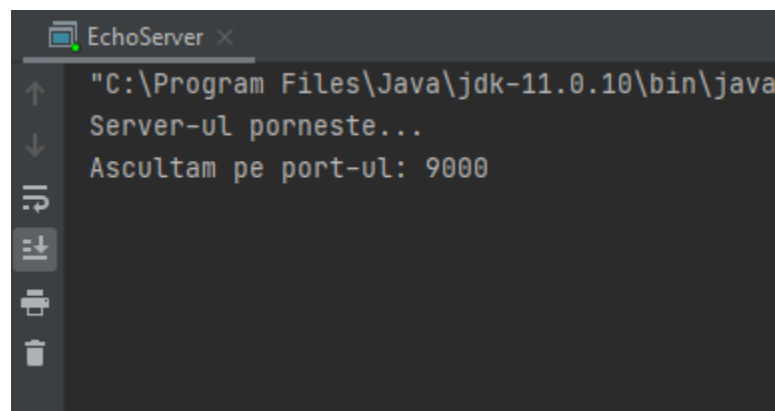


Figura 5. 2 Confirmarea pornirii aplicatiei EchoServer

În mod similar, pentru pornirea aplicației EchoClient, deschidem fila cu 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.

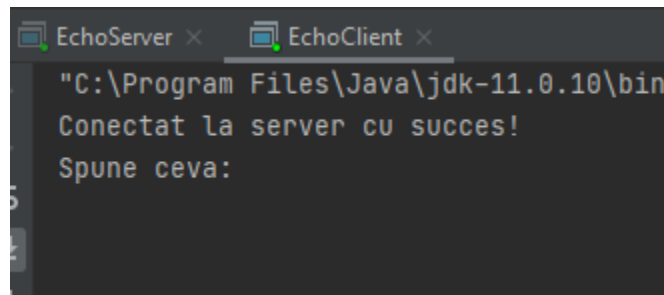


Figura 5. 3 Confirmarea pornirii aplicației EchoClient

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

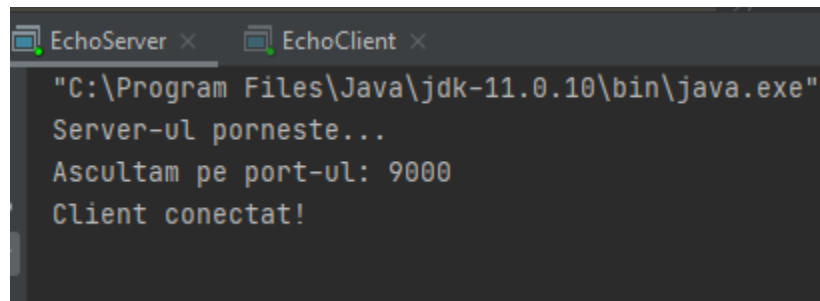


Figura 5. 4 Confirmarea conexiunii clientului

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.

Capture

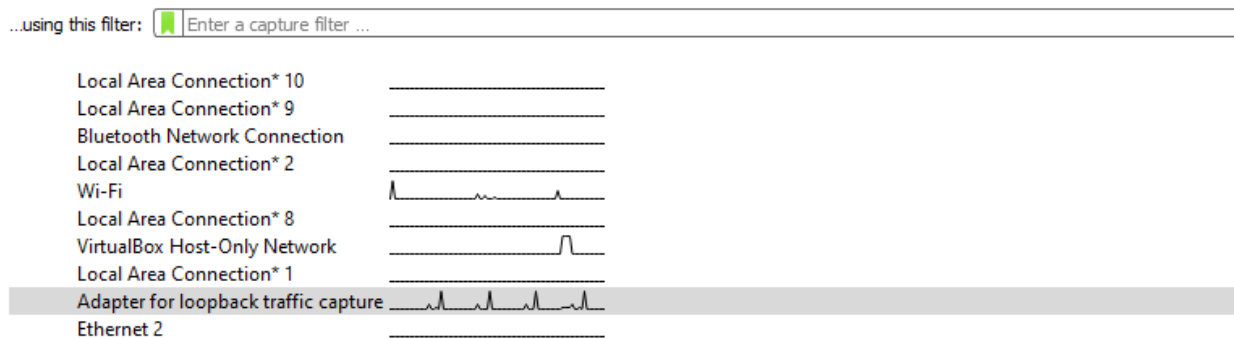


Figura 5. 5 Utilizarea Wireshark pe traficul local

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

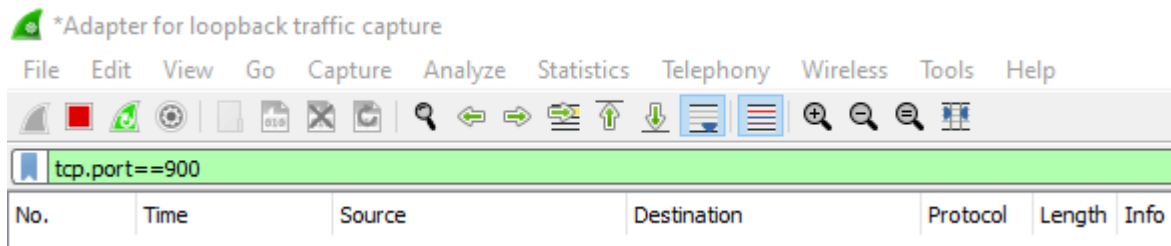


Figura 5. 6 Aplicarea filtrului pentru port in Wireshark

Capitolul 5.3 Functionarea aplicatiilor EchoServer si EchoClient

In momentul introducerii unui mesaj in 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. 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 cuvintului "exit" pentru terminarea aplicatiei 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.

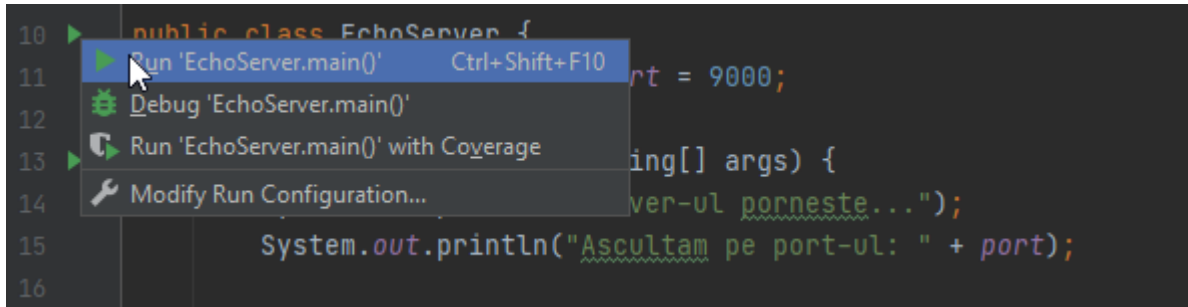


Figura 5. 10 Pornirea aplicatiei EchoServer

Dupa pornirea aplicatiei in consola ar trebui sa vedem mesajele care ne confirma ca serverul a pornit si ca asteapta conexiuni.

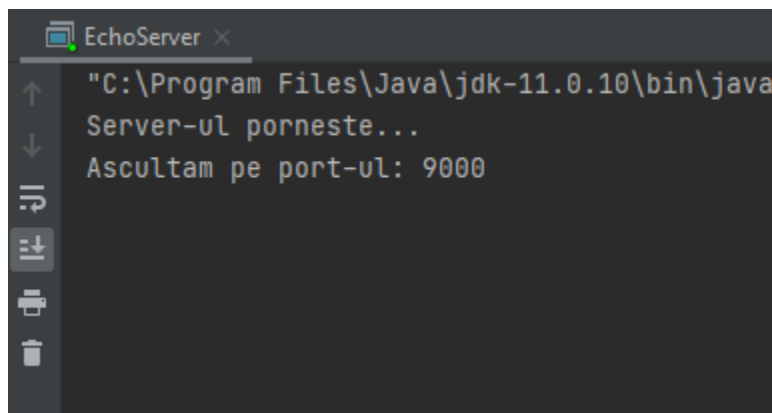
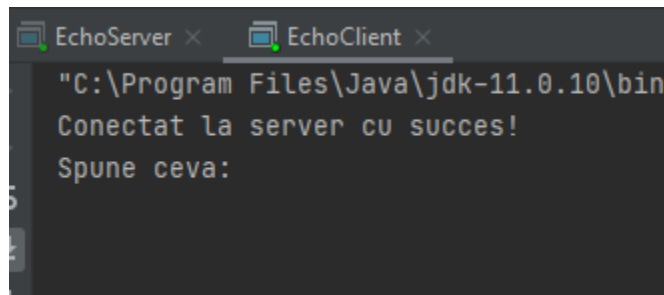


Figura 5. 11 Confirmarea pornirii aplicatiei EchoServer

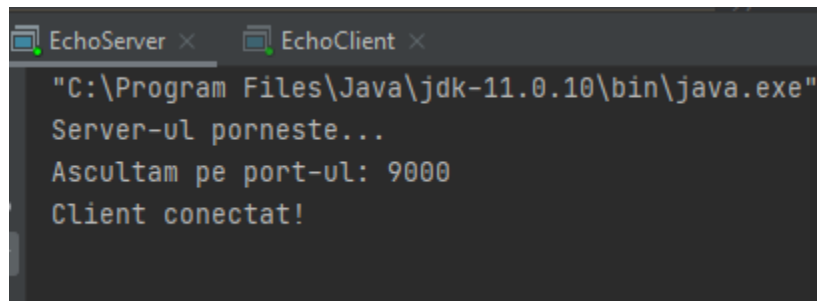
În mod similar, pentru pornirea aplicației EchoClient, deschidem fila cu codul sursa al aplicației și rulăm aplicația. Dacă totul a funcționat corect, lângă consola EchoServer ar trebui să se deschidă o consolă nouă. Se afișează mesajul cu confirmarea conexiunii la server și așteaptă introducerea unui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



```
"C:\Program Files\Java\jdk-11.0.10\bin
Conectat la server cu succes!
Spune ceva:
```

Figura 5. 12 Confirmarea pornirii aplicatiei EchoClient

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

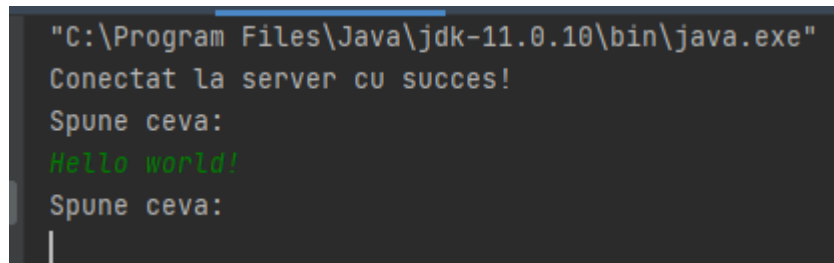


```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe"
Server-ul porneste...
Ascultam pe port-ul: 9000
Client conectat!
```

Figura 5. 13 Confirmarea conexiunii clientului

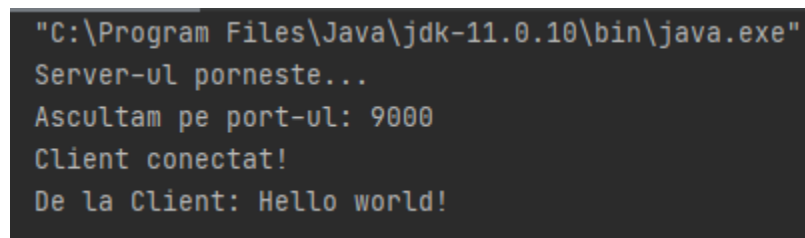
Capitolul 5.5 Functionarea aplicatiilor EchoSSLServer si EchoSSLClient

In momentul introducerii unui mesaj in 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, vom trimite din nou un mesaj de pe aplicația EchoClient. În Wireshark putem observa acum captura traficului dintre client și server. Observăm că mesajul trimis poate fi citit și înțeles din Wireshark confirmându-ne astfel de ce este de preferat evitarea utilizării unei comunicații nesecurizate.

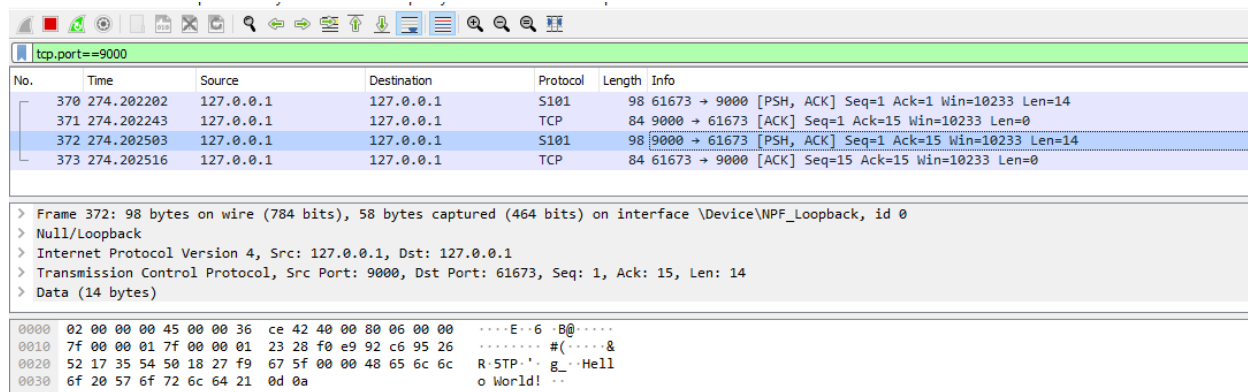


Figura 5. 16 Captura trafic Wireshark dintre EchoServer si EchoClient

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”.

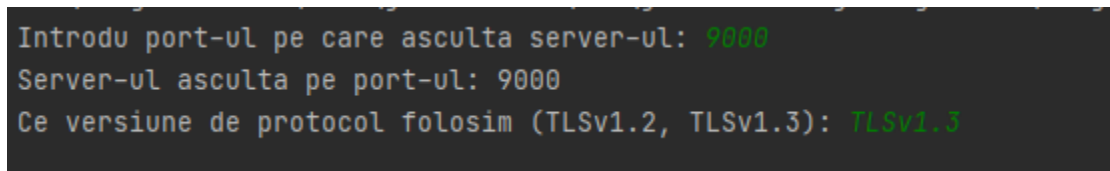


Figura 5. 17 Pornirea aplicatiei Server cu negocierea versiunii TLS 1.3

După pornirea aplicației Server, putem porni aplicația EchoSSLClient pentru a ne conecta la server. În cazul proiectului de față, 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 va 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

O alta 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-l 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 succes, ni se ca 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
SGFzYW4wHhcNMjEwMTcwMTIwHcNMjEwMTcwMTIwWjBxMQswCQYDVQQGEwJS
TzEQMA4GA1UECBMHUm9tYW5pYTESMBAGA1UEBxMJQnVjdXJlc3RpMRgwFgYD
VQQKEw9GYWN1bHRhdGVhIEVUVEkxCzAJBgNVBAsTAK1UMRUwEwYDVQQDEwxCb2dk
YW4gSGFzYW4wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaWf7lvKem
l/FtI2qQhQJhZKEUWUcMSnoH4NpBt0kxYbxNjZf7tCgJXXy/jH/gh1fnMLareAI
Mud9KffuHUKwab93a5te0FadfgaakztEYzcNcaX4L66QbRwJFWK48wRv3TnPCfSE
KggAfwS0i+JBZDeomHcNjQdxWBHVvncByBYdNFMDjwH70/PcSDT/5M9DNpTzNch1
j8Fxr0zhZBklsdjoPikNKgSwH7yK1wvLfzPJJ0w+DeMxBbxd+rsDeB4Tzs8pdpbN
I20xn31JUiHnvOg4W40EeLj7aUfWGi90wV+iMDvskE2+LqX0SWR2LXNFDDeNVFv
RB3h70yEsZy/AgMBAAGjITAFMB0GA1UdDgQWBBR3zitaQPOBvmSXb/KCwWug8h/
zTANBgkqhkiG9w0BAQsFAAOCQAQEA2ABj+Tcb3x4LY1ul+g40GNWufZ+5bYzFVviZ
++fAG0JmNXfLUKNGu7XaDy80CSVBksSRDFUr88vhD7Xf9KYXgerriuteWmM7IuhK
4Y99lomU6LLNGucCdNULGgjc8r0vATO6X1BXqEVIImZTMZCngPfNzJrWRXtxgv2W
Iqniv8TW4uXhvnY/PxoJ9JPlRfontxQemivMder3oBX3R4QxpjWixS1ZH2EX6FD
KZ90ZdIBR7h89qc+EECKa105eFog1xXt282asHHIAm6kQogI6z76QLUOMN1zsKh8
05gf7b1wsgHu4N8siq3+5kGworKcnb1DwvrdMPOBbbia3GhJeQ==
-----END CERTIFICATE-----

```

Figura 5. 22 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 că 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. 23 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. 24 Eroare aplicație “Server” cand versiunile negociate difera.

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. 25 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. 26 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" : "08 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 0F 38 AD 69 03 CE 06 F9 92 5D BF CA 09 6C 2E q.8.i.....)....
                0010: 83 C8 7F CD ....
              ]
            ]
          }
        ]
      }
    ]
  }
}
"extensions": {
  <no extension>
}

```

Figura 5. 27 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 pentru comunicatie securizata, enter pentru comunicatie in plain text):
Server-ul porneste...
Asculta pe port-ul: 9000

```

Figura 5. 28 Pornirea aplicatiei "ClassFileServer" cu telecomunicatie necriptata.

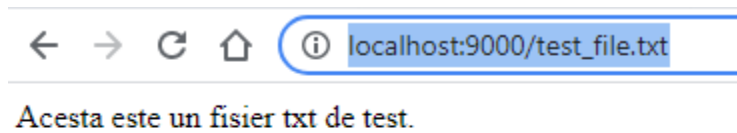


Figura 5. 29 Accesarea unui fisier de test prin HTTP

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

```
Introdu portul pe care sa asculte server-ul: 9000
Alege protocolul folosit (TLS pentru 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. 30 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. 31 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`

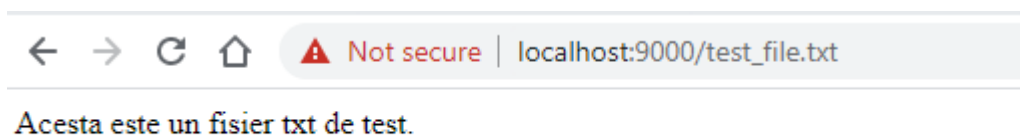


Figura 5. 32 Comunicatie nesigura cand folosim certificate care nu sunt de incredere

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” că și în cazul certificatelor folosite în realizarea acestui proiect, sau acesta este semnat dar nu este considerat de încredere, nefacand parte dintr-un container de chei pe care serverul să îl considere de încredere

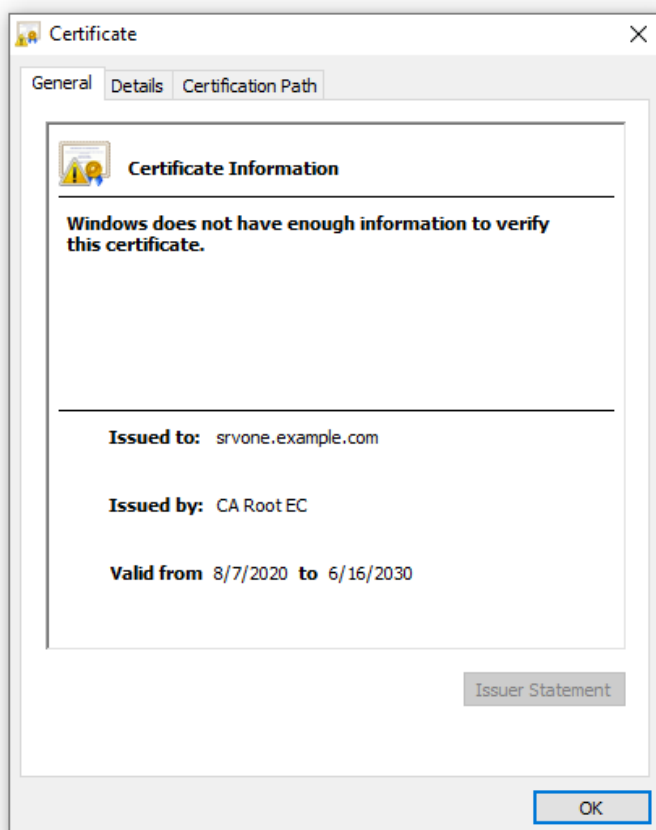


Figura 5. 33 Certificat nevalid

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. 34 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. 35 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. 36 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. 37 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. 38 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. 39 Adaugarea unui certificat intr-un "truststore"

Concluzii și Direcții de Viitor

Proiectul propune implementarea unor aplicații care să permită schimbul de date între ele folosind comunicații criptate dar și necriptate.

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:

- Crearea unor aplicații care să permită schimbul de mesaje în mod criptate și care să accepte conectarea mai multor utilizatori.
- Crearea unor aplicații Web care să permită afisarea de pagini HTML.

Bibliografie

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

```

