MyFileTransferProtocol

Bogdan George-Alexandru

B3
Facultatea de Informatica - Universitatea Alexandru Ioan Cuza din Iași

Profesori: Alboaie Lenuta, Panu Andrei

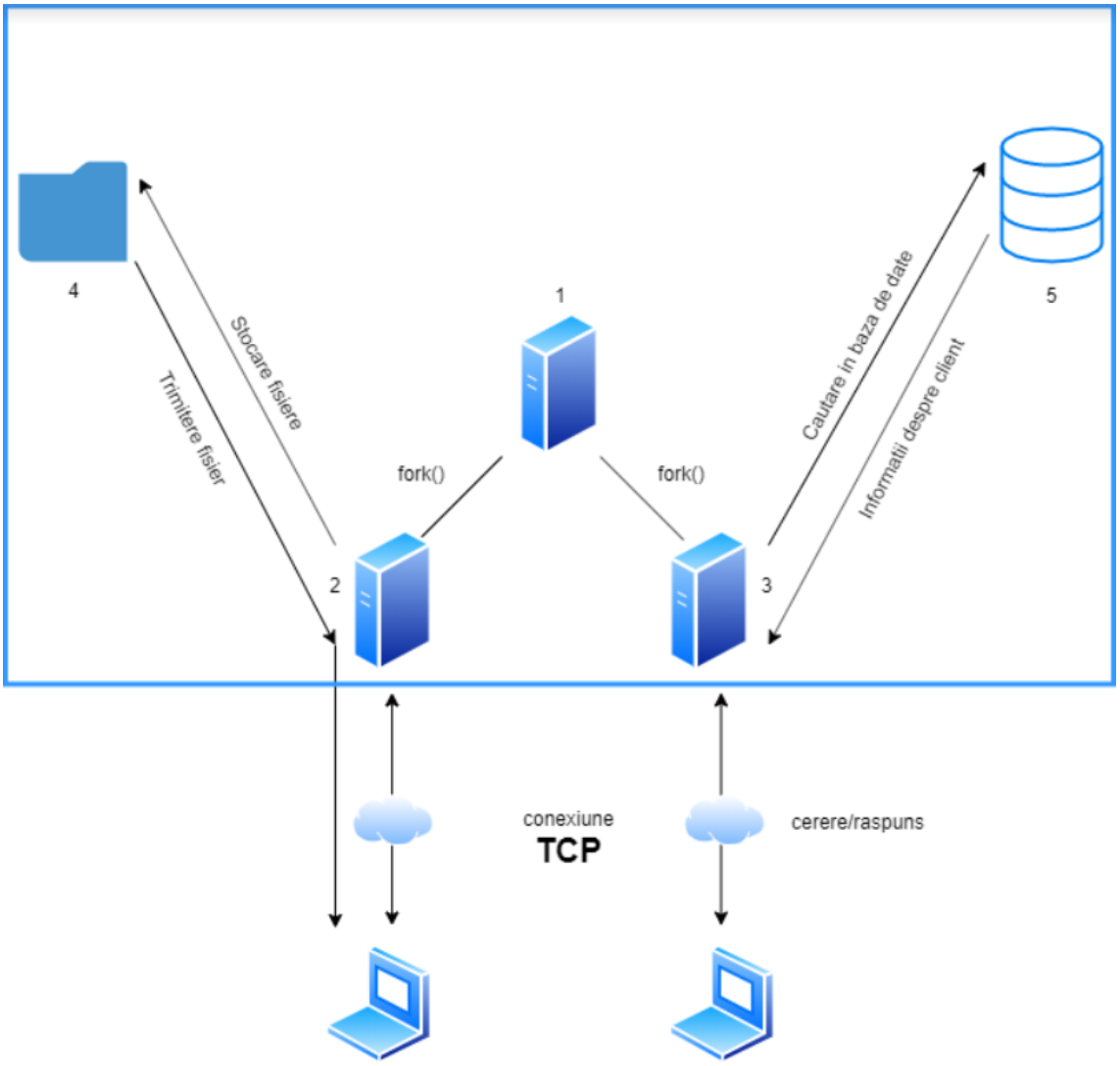
Introducere

Sa se implementeze o aplicatie client/server ce permite transferul de fisiere intre clienti si server. Serverul va pune la dispozitia clientilor un numar minim de comenzi ce permit autentificarea, operarea cu directoare si cu fisiere. De asemenea, va trebui implementat un mecanism de autorizare (whitelist/blacklist) pentru conturile utilizatorilor si un mecanism de transmitere securizata a parolei la autentificare.

Tehnologiile utilizatei

In realizarea proiectului am ales sa utilizez protocolul TCP (Transmission Control Protocol). TCP este un protocol optimizat pentru livrarea datelor fara pierdere de pachete intrucat expeditorul asteapta confirmarea destinatarului. Trasnferul de fisiere intre client si server implica si necesitatea utilizarii unui protocol ce garanteaza integritatea transmiterii fisierului. File Transfer Protocol utilizeaza protocolul TCP.

Arhitectura aplicatiei



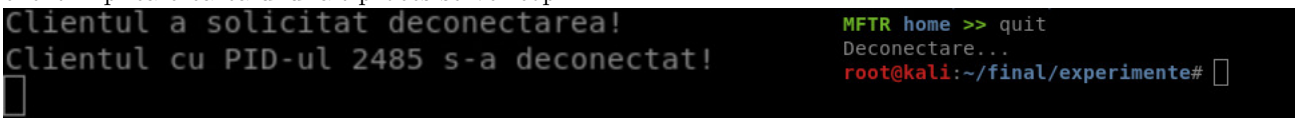
Serverul serveste cererile clientilor in mod concurent prin intemediul proceselor copil. Exista un unic proces copil (2) pentru fiecare client in parte.
Baza de date (5) a serverului contine informatiile fiecarui client:

- Username
- Parola (criptata folosind chiar cheile RSA publice ale serverului)
- Blacklisted (0 sau 1 in functie de caz)
- Super-Utilizator (0 sau 1 in functie de caz)

Clientii se pot deplasa liber in interiorul directorului (4) unde sunt stocate fisierele la care au acces:

- Acces la orice subdirector
- Acces la statisticile fiecarui fisier (dimensiune, data ultimei modificari)
- Din motive de securitate nici un client nu are acces la directorul ”parinte” al directorului 4
- Clientii nu pot modifica sau sterge fisiere

Clientul interactioneaza numai cu procesul copil ce ii este atribuit, iar fiecare proces copil ii raspunde cererilor avand acces la baza de date cat si la toate fisierele "descarcabile".
Deconectarea normala se face prin comanda "quit" transmisa de client, socketul este inchis din ambele capete iar procesul server-copil este asteptat de parinte pana la incheiere folosind un handler. Reconectarea unui client implica crearea unui alt proces server-copil.



Daca aplicatia client este inchisa in mod neasteptat serverul isi inchide capatul de socket si asteapta procesul copil pana se termina.

Daca aplicatia server nu mai raspunde aplicatia client isi inchide capatul de socket si se inchide.

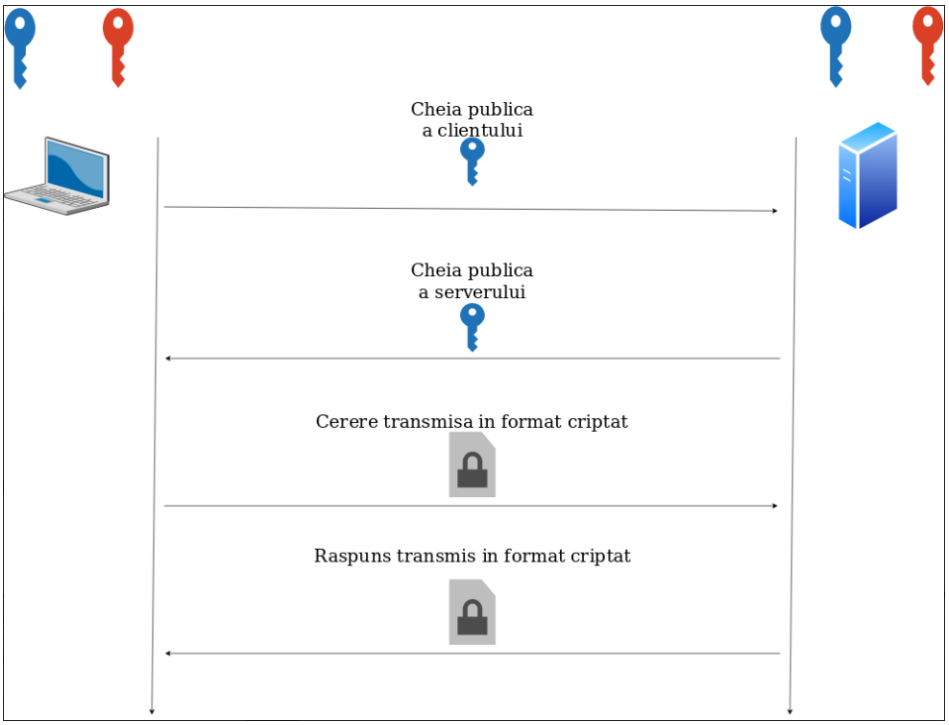
Detalii de implementare

Protocolul stabilit de catre mine in transmiterea mesajelor este:
Mesajele trimise de client catre server sunt de forma [comanda] [optiune]
Raspunsurile serverului la comenzile punctuale sunt de forma [calecurenta]$[raspuns]
Unde [calecurenta] reprezinta calea relativa din directorul (4) in care clientul se afla la momentul respectiv. Astfel incat clientul stie mereu in ce subdirector al directorului (4) se afla la un anumit moment.

Comunicatia dintre client si server este securizata utilizant algoritmul RSA.

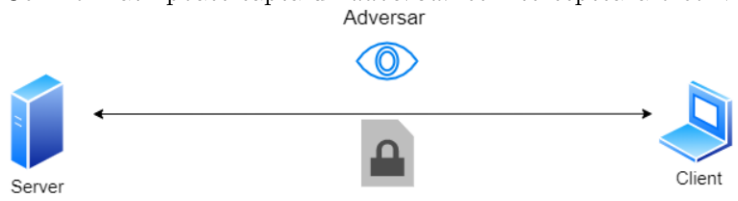
Descrierea algoritmumui de criptare RSA

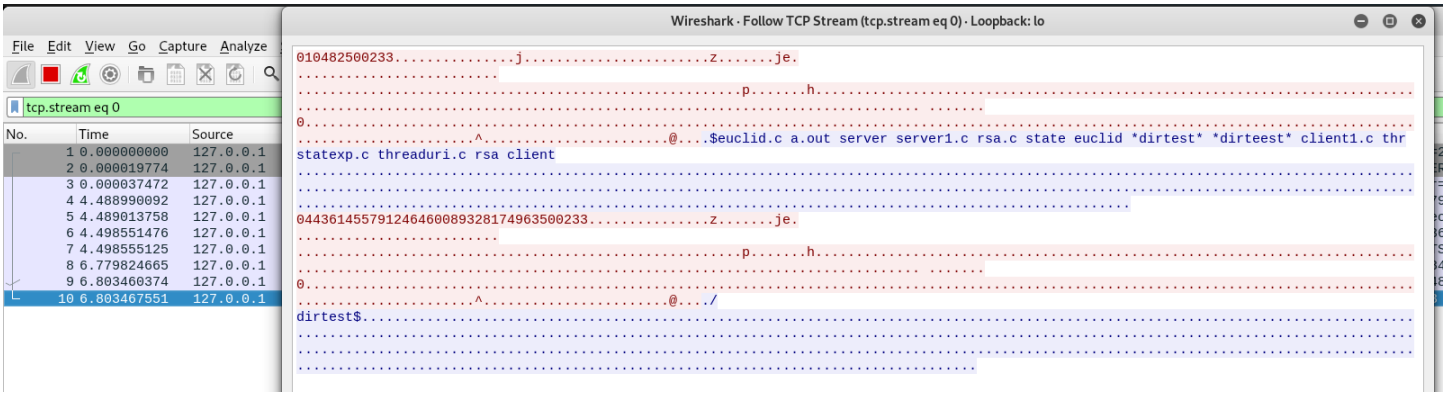
Atat serverul cat si clientul aleg doua numere prime p si q, se calculeaza $n = p \cdot q$ si $\Phi = (p-1) \cdot (q-1)$. Se alege e a.i. $cmmdc(e, \Phi) = 1 \Rightarrow \exists d$ a.i $e \cdot d \equiv 1 (mod\, \Phi)$ ce poate fi calculat eficient cu algoritmul extins al lui Euclid. Cheia publica este perechea (n, e) iar cheia privata este (n, d). Stabilirea legaturii dintre client si server consta intr-un schimb de chei publice. Cheia privata este folosita la decriptare si nu este niciodata transmisa.



Rezultatele implementarii algoritmumui de criptare RSA

Ce informatii poate capta un *adversar* ce intercepteaza o conversatie server-client?





Pentru a simula un astfel de *adversar* am utilizat "WireShark", captand pachete TCP transmise intre client si server in urma unei solicitari de tip "ls", lansate de client.

In imagine clientul transmite mesaje criptate iar raspunsurile serverului sunt in clar.
Primul mesaj (evidentiat cu rosu) este trimis de client si reprezinta comanda "ls", este indescifrabil pentru cineva ce nu are acces la cheia privata a serverului.
Al doilea mesaj (evidentiat cu albastru) este raspunsul dat de catre server, ce poate fi vizibil oricarui *adversar*.
Evident, in stadiul final atat comunicatia client-server cat si server-client va fi criptata.
Singura slabiciune a sistemului este reprezentata de captarea schimbului de cheai publice transmise in clar.
Odata captate securitatea mesajelor depinde numai de cat de greu de factorizat sunt numerele n alese de parti.

Login/Register

Dupa schimbul de chei publice, pentru a avea acces la comenzile si fisierele disponibile serverul ii solicita clientului Username-ul si Parola. Acesta le trimite (evident in format criptat) si asteapta confirmarea serverului.
Serverul verifica in baza de date existenta clientului, parola cat si daca eventual clientul este marcat ca blacklisted (caz in care conexiunea nu ii este acceptata).
Un client poate opta si pentru optiunea register, ce ii permite sa isi seteze un username (ce va fi verificat de server astfel incat sa fie unic) cat si o parola. Orice client proaspat inregistrat nu o sa fie marcat ca blacklisted.
Doar un client marcat ca Super-Utilizator poate marca un alt client in blacklist.

Operarea cu directoare si cu fisiere

Odata autentificat, clientul are la dispozitie o serie de comenzi:

- cd cu optiunile . .. [cale]
- ls (optional poate fi adaugata si o cale)
- stat [fisier]
- quit
- download [cale]
- upload [numefisier]
- blacklist [nume] 1 (numai pentru Super-Utilizatori) adauga utilizatorul pe blacklist
- blacklist [nume] 0 (numai pentru Super-Utilizatori) elimina utilizatorul de pe blacklist

Transmiterea fisierelor

Pentru transmiterea unui fisier expeditorul deschide fisierul respectiv si il transmite in binar prin socket. Transmiterea fisierelor se face prin intermediul aceluiasi socket prin care clientul transmite comenzi catre server.
Expeditorul transmite pentru inceput dimensiunea fisierului ce urmeaza sa fie trimis, dupa care transmite un caracter #.
Destinatarul citeste caractere din socket atata timp cat acestea sunt cifre. Primul caracter transmis de expeditor ce nu este cifra este chiar caracterul #. Expeditorul incepe sa transmita fisierul in binar, iar destinatarul citeste atat timp cat a citit mai putin decat dimensiunea fisierului.
Pentru a evita suprascrierea unor fisiere sunt utilizate prefixe.
Spre exemplu daca in folderul unde se descarca fisierul text.txt exista deja un fisier numit text.txt, acest nou fisier o sa primeasca numele de text1.txt, urmatorul text2.txt si tot asa. Acest prefix este pozitionat inaintea posibilei terminatii a fisierului (.txt .c .cpp .zip). Astfel fisierele mai mari trebuie impartite si trimise prin mai multe pachete.

Use case-uri

Sectiune dedicata situatiilor neasteptate ce pot aparea pe parcursul unei comunicari intre client si server
Un client poate solicita descarcarea unui fisier de pe server doar daca acest fisier exista in folderul curent in care clientul a navigat. In caz contrar clientul este avertizat ca nu a solicitat un fisier valid.
Asemenea un client poate transmite catre server un fisier din folderul local al aplicatiei client.
O deconectare normala si recomandata a clientului de la server se face prin comanda "quit".
O deconectare a clientului rezultata in urma intreruperii aplicatiei client in oricare alt mod decat cel precizat anterior nu afecteaza capacitatea serverului de a-si continua comunicarea cu alti clienti.
O intrerupere a aplicatiei server in timpul unei conversatii dintre server si minim un client duce la inchiderea aplicatiei client in urma executarii oricarei comenzi.
Atat intreruperea aplicatiei client cat si intreruperea aplicatiei server in timpul unei trasmiteri de fisier duc la transmiterea incompleta a datelor. In urma intreruperii aplicatiei client in timpul unei transmiteri de fisier server - client variabile errno din aplicatia server ia valoarea EPIPE, caz in care serverul intrerupe transmiterea iar procesul copil alocat respectivului client se incheie.
In urma intreruperii aplicatiei server in timpul unei transmiteri clientul o sa primeasca doar o parte din date, acesta este avertizat la sfarsit ca datele nu au fost descarcate complet deoarece aplicatia client simuleaza un potential diferit client ce intentioneaza sa se conecteze la server pentru a se asigura ca serverul raspunde.

Concluzii

In aceasta sectiune prezint unele imbunatatiri ce pot fi aduse atat pe partea de server cat si pe partea de client fara sa exclud posibilitatea de a implementa unele din ele:

- Transmiterea fisierelor in format criptat
- Stocarea securizata a parolelor utilizand Hash-uri
- Interfata grafica pentru utilizatori
- Stocarea securizata a cheilor RSA
- Stabilirea unor chei RSA mai dificil de factorizat
- Utilizarea sistemului de criptare OpenSSL
- Un client conectat ce este marcat in blacklist sa fie imediat deconectat
- Stocarea IP-ului fiecarui client in baza de date
- Blacklist IP
- Fiecare client sa aiba un folder personal si sa poata stoca fisiere, asemeni unui *Cloud*

Bibliografie

[1] https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
[2] https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
[3] http://web.cs.wpi.edu/ cs542/f09/slides/SQLC.pdf
[4] https://www.geeksforgeeks.org/rsa-algorithm-cryptography/
[5] https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/
[6] https://www.geeksforgeeks.org/c-program-for-basic-and-extended-euclidean-algorithms-2/
[7] https://www.youtube.com/watch?v=12F3GBw28Lg