My2FA(B)

Baciu-Ciongradi Tudor

Facultatea de informatică , Universitatea Alexandru Ioan Cuza Iași
tudorbc23@gmail.com

Abstract. Acest raport prezintă implementarea unei perechi client/server care suportă autentificare, executarea de comenzi la nivelul server-ului şi a clientului şi returnarea rezultatelor către client. Raportul urmăreşte structura LNCS şi oferă detalii privind arhitectura, tehnologiile şi soluţiile alese, precum şi scenarii de utilizare.

Keywords: Autentificare 2FA · Client-Server · Protocol TCP · C++ 20 · LNCS

1 Introducere

În această secţiune vom prezenta viziunea generală şi obiectivele proiectului. Scopul proiectului este devoltarea unui prototip de sistem 2FA. Sistemul constă într-un server central (Auth Server) care gestionează autentificarea utilizatorilor și metodele cele două metode de verificare (Notificări Push și Coduri TOTP). Obiectivele includ:

- Implementarea unui protocol de comunicare special bazat pe text.
- Gestionarea conexiunilor concurente folosind I/O Multiplexing.
- Implementarea a două metode de verificare (Notification și Code).

2 Tehnologii Aplicate

Pentru implementarea proiectului s-a optat pentru urmatoarele tehnologii: Limbajul C++ 20 a fost ales pentru siguranța strictă a tipurilor de date, performanță și pointerii inteligenți care gestionează automat memoria în cadrul fabricii de comenzi `CommandFactory`. TCP a fost ales ca strat de transport pentru siguranța ridicată comparativ cu UDP. I/O Multiplexing (select) a fost ales pentru gestionarea concurenței. Serverul utilizează `select()` pentru a gestiona simultan mai multe conexiuni de tip client sau server, păstrând o implementare simplă fără complexitatea multi-threading-ului.

3 Structura Aplicației

1. **Auth Server (AS):** Serverul central și generatorul de coduri. Acesta menține un registru al utilizatorilor conectați (`UUID` → `Socket`) și rutează mesajele între servere și clienți.

2. **Auth Client (AC):** Clientul prin care se pot cere coduri sau prin care se pot accepta notificări.
3. **Dummy Server (DS):** Acest server ține locul unui backend al unei aplicații și face legatura între Clientul Aplicație (DC) și Serverul de Autentificare (AS).
4. **Dummy Client (DC):** Client care simulează aplicația utilizatorului care necesită autentificare.

3.1 Concepte de Modelare

Logica internă se baseaza pe Factory Design Pattern. Fiecare comandă internă este incapsulată într-o clasă care este moștenită printr-o interfață de bază `Command`. `CommandFactory` este responsabil pentru parsarea datelor brute și instanțierea unui obiect specific comenzii, astfel izolând logica de parsare de logica de execuție.

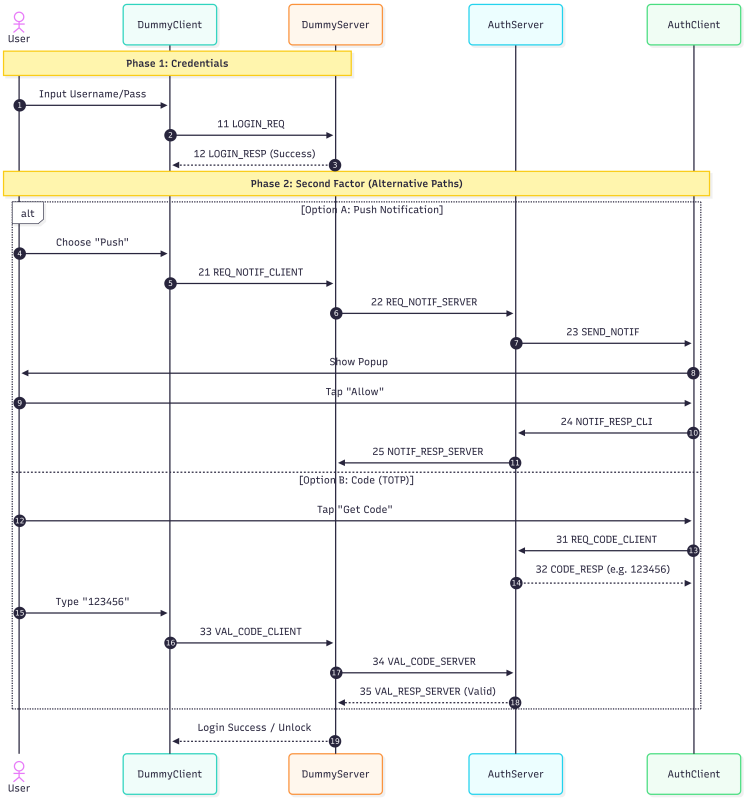

Fig. 1. Diagrama de Structură a Proiectului

4 Protocolul de Comunicare

Protocolul la nivel de aplicație este bazat pe text, utilizând caracterul ; ca delimitator. Formatul general este: "ID_COMANDĂ;ARGUMENT1;ARGUMENT2;...".

Table 1. Specificația Comenzilor Protocolului

ID	Nume Comandă	Argumente	Descriere
1	CONN	Type, AppID	Handshake inițial
3	ERR	Code, Msg	Transmitere erori
11	LOGIN_REQ	User, Pass	Verificare credențiale
12	LOGIN_RESP	Resp, UUID	Rezultat credențiale
21	REQ_NOTIF_CLIENT	UUID	DC cere auth prin Notificare Push
22	REQ_NOTIF_SERVER	UUID, AppID	DS cere AS să notifice userul
23	SEND_NOTIF	AppID	AS trimite notificare către AC
24	NOTIF_RESP_CLIENT	Resp, AppID	AC trimite Accept/Refuz
25	NOTIF_RESP_SERVER	Resp, UUID	AS trimite rezultatul la DS
31	REQ_CODE_CLIENT	UUID, AppID	AC cere cod TOTP lui AS
32	CODE_RESP	Code	AS trimite codul generat
33	VALIDATE_CODE_CLIENT	Code, UUID	DC trimite codul lui DS pentru validare
34	VALIDATE_CODE_SERVER	Code, UUID, AppID	DS cere validarea codului lui AS
35	VALIDATE_RESP_SERVER	Resp, UUID, AppID	AS trimite raspunsul lui DS
36	VALIDATE_RESP_CLIENT	Resp, UUID	DS trimite raspunsul lui DC

5 Scenarii de Utilizare

5.1 Scenariu Succes 1: Autentificare prin Notificare Push

Utilizatorul inițiază logarea folosind metoda "Notificare".

1. DC Trimite `REQ_NOTIF_CLIENT` către DS.
2. DS Trimite `REQ_NOTIF_SERVER` către AS.
3. AS Caută UUID-ul utilizatorului în tabela de rutare, găsește socket-ul AC și trimite `SEND_NOTIF`.
4. AC Afișează notificarea. Utilizatorul apasa "allow", apoi AC trimite `NOTIF_RESP_CLIENT` (True).
5. AS Rutează rezultatul către DS prin `NOTIF_RESP_SERVER` (True).
6. DS Permite accesul aplicației DC ("Login Success").

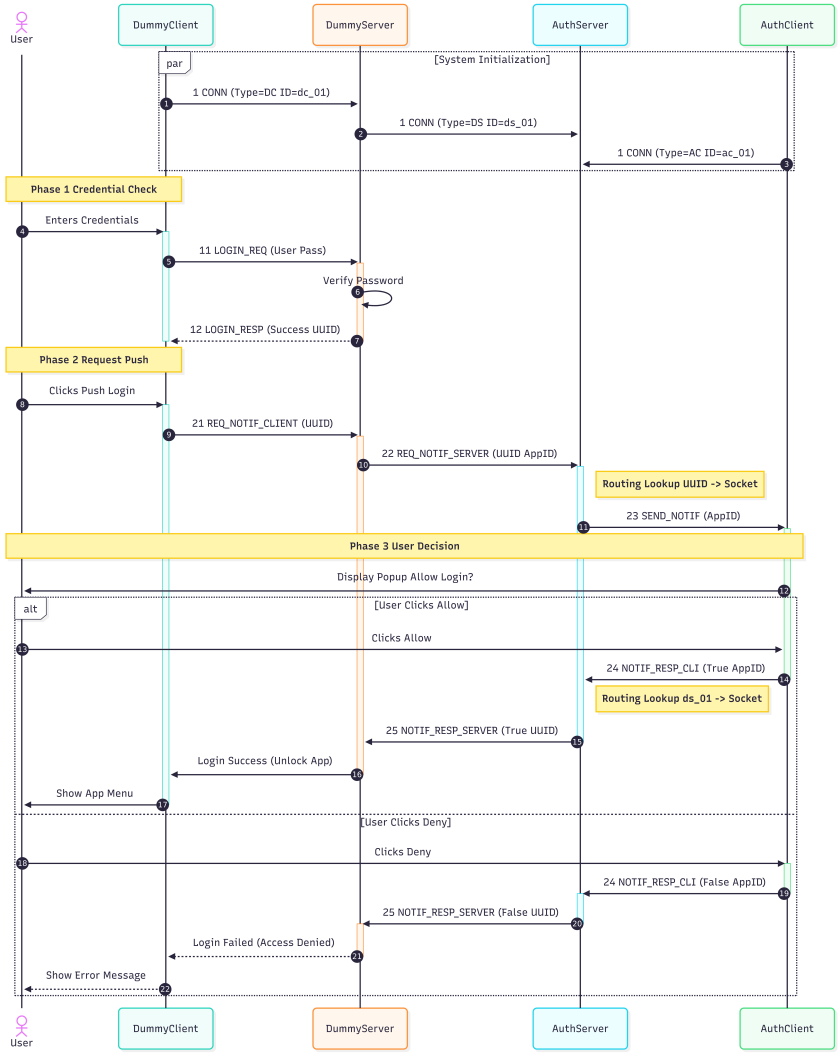

Fig. 2. Diagrama Scenariului Login prin Notificare

5.2 Scenariu Succes 2: Autentificare prin Cod TOTP

Utilizatorul solicită un cod pe telefon pentru a-l introduce manual.

1. AC Trimite `REQ_CODE_CLIENT` către AS.
2. AS Generează codul 123456 și răspunde cu `CODE_RESP`.
3. Utilizatorul introduce codul în DC.
4. DC Trimite `VALIDATE_CODE_CLIENT` către DS.

5. DS Redirecționează comanda către AS prin `VALIDATE_CODE_SERVER`.
6. AS Validează codul și returnează `VALIDATE_RESP_SERVER` (True).

5.3 Scenariu Succes 3: Sistem Handshake

La inceputul unei conexiuni, serverul/clientul se va "prezenta"

1. DS Se conectează la AS pe portul 27701.
2. DS Trimite `CONN;DUMMY_SERVER;ds_01`.
3. AS Mapează socket descriptor-ul la ID-ul ds_01 pentru rutarea viitoarelor comenzi.

5.4 Scenariu Eșec 1: Notificare Respinsă

Utilizatorul refuză Notificarea.

1. AC Trimite `NOTIF_RESP_CLIENT` (False).
2. AS Rutează `NOTIF_RESP_SERVER` (False) către DS.
3. DS Trimite "Access Denied" către DC, oprind logarea.

5.5 Scenariu Eșec 2: Cod Invalid

Utilizatorul introduce greșit codul OTP.

1. DC Trimite `VALIDATE_CODE_CLIENT` cu codul 999999.
2. DS Redirecționează codul pentru verificare către AS.
3. AS Compară 999999 cu codul generat 673123. Fals!
4. AS Trimite `VALIDATE_RESP_SERVER` (False), iar utilizatorul primește eroare in DC.

5.6 Scenariu Eșec 3: Pachet Greșit

Clientul trimite date corupte sau formatate greșit conform protocolului.

1. DC Trimite șirul "Code" (fără ID valid) lui DS.
2. `CommandFactory` nu reușește parsarea ID-ului și returnează `nullptr`.
3. DS Semnalează eroarea "Invalid Format" și ignoră pachetul pentru a sanitiza stream-ul I/O și pentru a preveni o posibilă buclă infinită.

6 Concluzii

Soluția prezentată a implementat cu succes un sistem de autentificare 2FA. Utilizarea pattern-ului Command a modularizat structura proiectului, facilitând actualizari simple pe viitor. Posibile îmbunătățiri:

– **Claritate și Redundanță**: O revizuire a structurii comenzilor ar putea reduce complexitatea comenzilor și ar îmbunătății lizibilitatea codului.

- **Securitate**: Criptarea traficului folosind TLS pentru a preveni interceptarea pachetelor.
- **Persistență**: Integrarea unei baze de date SQL pentru stocarea credențialelor, înlocuind structurile deja existente. Această abordare merge perfect cu conceptul de modularizare pe care l-am menționat.
- **Timeout**: Implementarea unui mecanism de expirare pentru cererile de notificare push care rămân fără răspuns mai mult de 30 de secunde.

Referințe Bibliografice

References

1. Springer LNCS Guidelines: `https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines`
2. C++20: `https://en.wikipedia.org/wiki/C%2B%2B20`
3. TCP: `https://en.wikipedia.org/wiki/Transmission_Control_Protocol`
4. Factory Pattern Design: `https://refactoring.guru/design-patterns/factory-method`