Documentatie proiect Two-Factor Authentication (2FA)

Popa Stefan-Eduard 2E1

05.12.2022

1 Introducere

Proiectul include 2 servere (”Some server”,”2FA server”) si 2 cliente (”Some client”,”2FA client”). Some client va incerca sa se logheze la Some server prin credentiale aferente unui anumit cont . Some server va recepta cererea de logare , va incerca sa compare credentialele trimise cu cele existente in baza sa de date si va trimie inapoi clientului un raspuns reprezentativ pentru succesul sau esuarea logarii (procedura obisnuita pentru orice aplicatie multi-user). In cazul in care operatia a avut succes , serverul va pune clientul in situatia in care trebuie sa aleaga intre 2 variante : trimiterea unei notificari in aplicatia de 2FA pentru a confirma identitatea sau introducerea unui cod din aplicatia de 2FA. In cazul notificarii, daca se primeste un raspuns pozitiv de la 2FA server, autentificarea in aplicatia aditionala (Some client si Some Server) va functiona, altfel va esua. In cazul utilizarii mecanismului bazat pe coduri, codul introdus de utilizator in Some client, va fi preluat de Some server si verificat cu serverul 2FA, pentru a decide daca autentificarea s-a realizat cu succes . Important de mentionat este faptul ca acele coduri sunt generate odata la un anumit interval de timp .

2 Tehnologiie utilizate

2.1 Transmission Control Protocol (TCP)

Tehnologia aceasta sta la baza comunicarii intre orice client si server din cadrul proiectului .Acest protocol a fost ales in schimbul UDP din cauza ”sigurantei” modului in care pachetele sunt trimise . TCP foloseste anumite mecanisme de control ce asigura faptul ca pachetele nu sunt pierdute, stricate , schimbate , dublicate sau trimise in alta ordine decat cea cronologica.Un dezavantaj ar fi viteza de transmitere si marimea pachetelor , insa acestea nu sunt prioritare in contextul de fata.

2.2 Threads

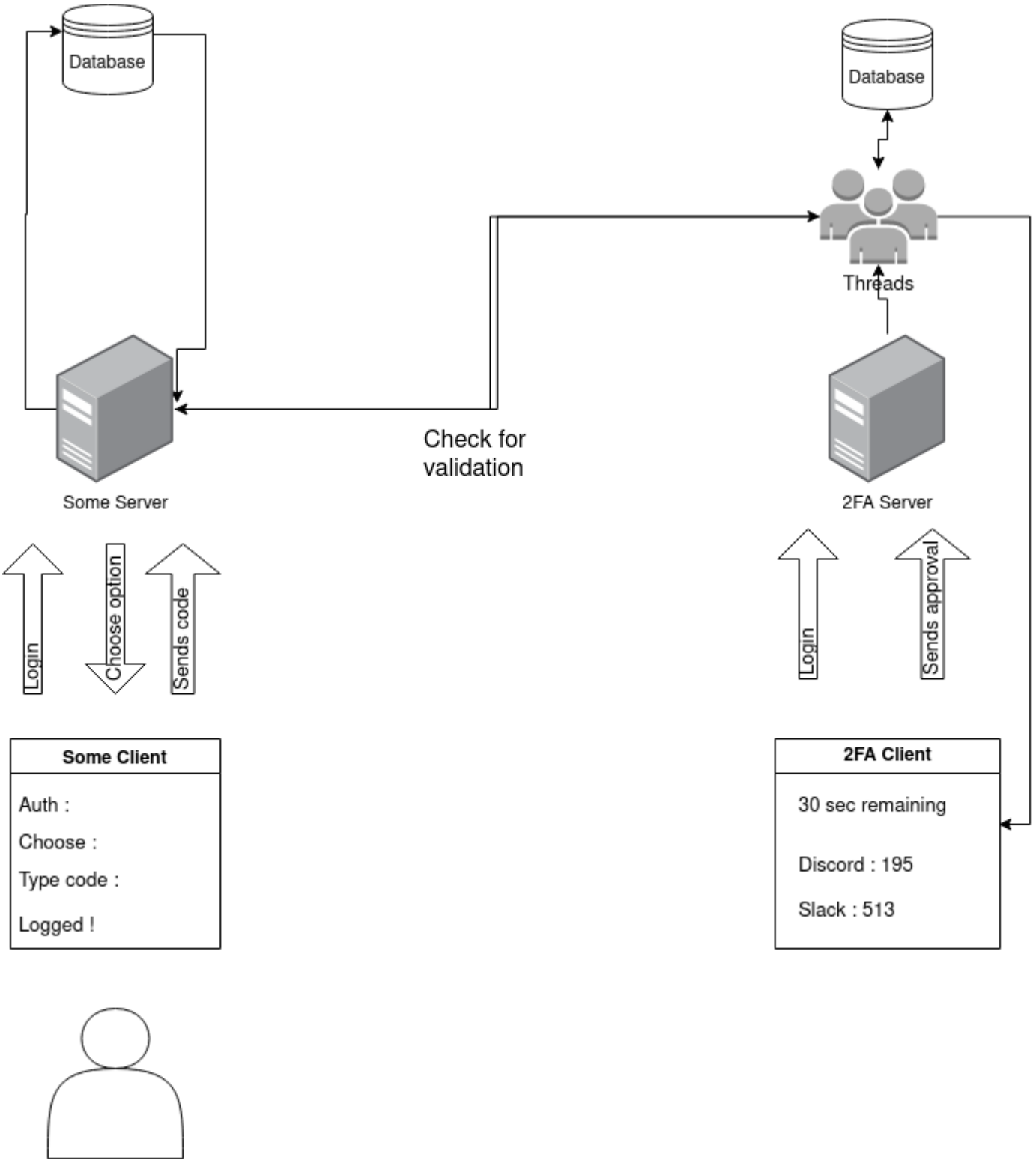
Acestea sunt folosite la serverul 2FA si ii permite indeplinirea anumitor slujbe in mod asincron (raspunderea la cereri , cititul si scrisul la baza de date , generarea de coduri etc.)

2.3 Mutex

Acesta restrictioneaza accesul la baza de date la maxim un singur thread . Scopul este evitarea ”ciocnirilor” intre operatiile de Read si Write ce ar putea produce inconsistenta informatiilor trimise la cererea clientilor serverului 2FA sau diverse erori.

3 Arhitectura aplicatiei

Diagrama ar fi aceasta :



Serverul 2FA este concurent si este bazat pe TCP ca protocol de comunicare . Acesta serveste clientii prin crearea unui thread pentru fiecare client . Pe langa acele threaduri ce sunt responsabile cu servirea clientilor , mai este unul facut pentru regenerarea codurilor din baza de date . Accesul la baza de date este limitat printr-un mutex . Informatiile din baza de date , cererile sunt decriptate iar raspunsurile sunt criptate .
Clientul 2FA se conecteaza la 2FA server , utilizatorul se logheaza si daca nu au fost probleme atunci clientul incepe sa poata cere regulat update-uri pentru userul logat (coduri sau cereri de aprobare pornite de Some client) si sa trimita raspunsuri unde este cazul.

4 Detalii de implementare

Acesta este structul folosit pentru incapsularea descriptorului intors de accept si id-ul thread-ului .

```
typedef struct thData
{
    int idThread; // id-ul thread-ului tinut in evidenta de acest program
    int cl;           // descriptorul intors de accept
} thData;
```

Un scenariu de utilizare este atunci cand se accepta un client si se face un nou thread pentru a-l servi .

```
/* servim in mod concurent clientii...folosind thread-uri */
while (1)
{
    int client;
    thData *td; // parametru functia executata de thread
    int length = sizeof(from);

    printf("[server]Asteptam la portul %d...\n", PORT);
    fflush(stdout);

    /* acceptam un client (stare blocanta pina la realizarea conexiunii) */
    if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
    {
        perror("[server]Eroare la accept().\n");
        continue;
    }

    /* s-a realizat conexiunea, se astepta mesajul */
    td = (struct thData *)malloc(sizeof(struct thData));
    td->idThread = i++;
    td->cl = client;

    pthread_create(&th[i], NULL, &treatReq, td);


}
```

Mesajele de comunicare sunt transmise sub forma unui pachet de lungime fixa :

```
#ifndef PROTOCOL_H
#define PROTOCOL_H

#define SIZEOF_PACKET 1112
#define MAX_SIZE_OF_CONTENT 1000
typedef struct Packet{

    unsigned int from;
    unsigned int operation;
    unsigned int respone_code;
    char userID[50];
    char appID[50];
    char content[MAX_SIZE_OF_CONTENT];
}Packet;

#define FROM_2FA_SERVER 1
#define FROM_2FA_CLIENT 2
#define FROM_SOME_SERVER 6
#define FROM_SOME_CLIENT 4

#define OPERATION_QUIT 1
#define OPERATION_GET_CODES 2
#define OPERATION_LOGIN 3
#define OPERATION_LOGOUT 4
#define OPERATION_UPDATE 5
#define OPERATION_CHECK_CODE_REQ 6
#define OPERATION_CHECK_APPROVAL 7
#define OPERATION_APPROVE_REQ 8
#define OPERATION_DECLINE_REQ 9

#define RESPONSE_ERROR 0
#define RESPONSE_SUCCESS 1
#define RESPONSE_ACCEPTED 2
#define RESPONSE_DECLINED 3
#define RESPONSE_WAIT 4

#endif
```

Acest pachet contine informatii relevante precum aplicatia autor , operatia ce se vrea a fi facuta , rezultatul operatiei , identificatorul utilizatorului , numele aplicatiei si un buffer ce poate fi folosit pentru trimiterea informatiilor ce nu au o structura fixa ,specifica sau care nu se incadreaza in una din cele deja mentionate . Pachetu este folosit in orice transmitere de informatie intre 2FA server si clientii sai (Some server si clientul 2FA) . Un exemplu concret de utiizare este atunci cand se trimite un mesaj :

```
Packet response;
response.from = FROM_2FA_SERVER;
response.operation = OPERATION_GET_CODES;
```

```
response.respone_code = op_response ? RESPONSE_SUCCESS : RESPONSE_ERROR;
memcpy(response.content,(&uinfo),sizeof(uinfo));

if (write(tdL.cl, &response, SIZEOF_PACKET) <= 0)
{
    printf("[Thread %d] ", tdL.idThread);
    perror("[Thread]Eroare la write() catre client.\n");
}
else
    printf("[Thread %d]Mesajul a fost trasmis cu succes.\n", tdL.idThread);
```

Accesarea bazei de date de catre threaduri este limitata printr-un mutex ce permite accesarea bazei de catre cel mult un singur thread pe moment.

```
int accessDatabase(struct ContextForAccessingDatabase context,
struct userInfo *userinfo){

    pthread_mutex_lock(&lock);
    if( DEBUG_LV > 20 ) printf("Locked mutex\n");

    int result = 1;

    if(context.operation == OPERATION_GET_CODES){

        *userinfo = Search(context,&result);
    }
    else if(context.operation == OPERATION_APPROVE_REQ ||
        context.operation == OPERATION_DECLINE_REQ){

        Search(context,&result);
    }
    else if(context.operation == OPERATION_CHECK_CODE_REQ){
        Search(context,&result);
    }
    else if(context.operation == OPERATION_CHECK_APPROVAL){
        Search(context,&result);
    }
    else if(context.operation == OPERATION_UPDATE){
        Update();
    }
    else{
        result = RESPONSE_ERROR;
        printf("!!!NU EXISTA ACEST REQUEST");
    }

    if(result == RESPONSE_ERROR) {
        printf("!!!EROARE LA ACEST REQUEST!!! \n");
    }
```

```
    pthread_mutex_unlock(&lock);
    if( DEBUG_LV > 20 ) printf("Unlocked mutex\n");

    return result;
}
```

O alta bucata relevanta de cod este cererea de primire a update-urilor facuta de clientul 2FA pentru utilizatorul cu care este logat , citirea raspunsului si reprezentarea lor .

```
Packet request;
request.from = FROM_2FA_CLIENT;
request.operation = OPERATION_GET_CODES;
memcpy(request.content,name,sizeof(name));

/* trimiterea mesajului la server */
if (write(sd, &request, MAX_SIZE_OF_CONTENT) <= 0)
{
  perror("[client]Eroare la write() (Packet request) spre server.\n");
  return errno;
}
printf("S a trimis requestul \n");
/* citirea raspunsului dat de server
  (apel blocant pina cind serverul raspunde) */

Packet response;
if (read(sd, &response, SIZEOF_PACKET) < 0)
{
  perror("[client]Eroare la read() de la server.\n");
  return errno;
}
/* afisam mesajul primit */
if(response.respone_code == RESPONSE_SUCCESS){

    struct userInfo userinfo = *((struct userInfo*)response.content);

    printf("userinfo : %d , %s \n",userinfo.appsCount ,  userinfo.userID);

    for(int j = 0 ; j < userinfo.appsCount ; j++){

      printf("{ app : %s , code : %d } ; ", userinfo.apps[j].name ,
      userinfo.apps[j].code);

    }
    printf("\n");
}
else printf("Nasol :( \n");

sleep(TIME_TO_UPDATE);
```

5 Concluzii

O imbunatatire ar fi impartirea pachetelor in 2 parti , header si content . Header sa aiba o lungime mica si fixata ce reprezinta informatii generale legate de raspunsuri si cereri , iar contentul sa fie de lungima variabila .
Alta imbunatatire ar putea fi folosirea unui semafor in loc de mutex . Astfel am putea permite citirea din baza de date (cea mai folosita operatie) a mai multor thread-uri in acelasi timp , in loc de maxim 1 cum este in cazul folosirii unui mutex.

References

[1] https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c

[2] LNCS Homepage, http://www.springer.com/lncs.