

# Implementarea unui Terminal SSH

Busuioc Corina-Ștefania

Facultatea de Informatică, Universitatea “Alexandru Ioan Cuza”

## 1 Introducere

Proiectul are ca obiectiv crearea unui mediu pentru client care să simuleze funcțiile unui terminal de tip SSH, permițând executarea comenzilor simple sau compuse prin redirectionarea intrării/ieșirii standard și utilizarea pipe-urilor sau operatorilor logici (**and**, **or**). Comunicarea criptată joacă un rol crucial în asigurarea securității rețelei. Partea de autentificare, care folosește un fișier JSON, are ca scop folosirea unor fișiere mai complexe decât fișierele txt folosite majoritar la nivelul începător.

## 2 Tehnologii Aplicate

### 2.1 TCP

TCP garantează livrarea completă și ordonată a datelor, o cerință esențială în criptare și autentificare. Serverul gestionează doar conexiuni active, fără a bloca procesul principal.

### 2.2 Multiplexare

Multiplexarea permite gestionarea eficientă a mai multor conexiuni într-un singur fir de execuție, economisind resurse. Fiecare client este asociat unui descriptor de fișier.

### 2.3 OpenSSL

OpenSSL este utilizat pentru criptarea și decriptarea datelor folosind algoritmul AES-128 în mod CBC. AES-128 este o metodă de criptare rapidă și sigură, fiind un standard pentru securizarea comunicațiilor. OpenSSL simplifică utilizarea acestor algoritmi. Serverul generează perechi unice de chei (key și IV) pentru fiecare client folosind RAND\_bytes. Cheile sunt transmise clientului după criptare cu o cheie publică predefinită, pentru un schimb de chei inițial.

### 2.4 JSON

JSON este un format flexibil și ușor de utilizat pentru stocarea și manipularea datelor structurate. Datele utilizatorilor sunt stocate într-un fișier JSON. Biblioteca cJSON simplifică extragerea informațiilor (nume și parolă). Autentificarea utilizează o combinație de verificare a utilizatorilor și parolă în memoria serverului. JSON este mai ușor de gestionat decât alte formate precum XML.

- ### 3 Structura Aplicației

- Structuri de date personalizate: Structurile user, keys și node organizează datele legate de utilizatori (nume, parolă, clientul care e conectat), informațiile criptografice (chei și vectori de inițializare - IV) și structura comenzii.
- Codul este împărțit în funcții clare: read\_json, transform, encrypt\_decrypt, priority, add, build, execute, comcd și mesaj.
- Criptare simetrică AES (Advanced Encryption Standard). Cheie publică și privată: Serverul folosește o cheie simetrică comună (stocată în key și iv) pentru criptarea cheilor de sesiune generate pentru fiecare client. Securitatea canalului de comunicație: Cheile de sesiune ale fiecărui client sunt criptate folosind cheia simetrică comună înainte de a fi transmise
- Analiza comenzilor shell: Comenzile introduse de utilizator sunt analizate pentru a detecta caracteristici speciale: &&, ||, |, <, >, 2>.
- Construirea unui arbore de comenzi pentru a ține cont de prioritatea operatorilor, în funcție de operatorul care este rădăcina fiecărui subarbore se va executa subarborele stâng și drept.
- Execuție izolată: Comenzile sunt executate în procese copil.

## 4 Aspecte de Implementare

### 4.1 Criptare/Decriptare cu AES-128 CBC

```
int encrypt_decrypt(unsigned char key[16], unsigned char iv
[16],
                    const char* text, const int l, char* rez,
                    int sau) {
    EVP_CIPHER_CTX *ctx;
    int lrez = 0, total_l;

    if ((ctx = EVP_CIPHER_CTX_new()) == NULL) {
        perror("Eroare la crearea contextului");
        return 0;
    }
    EVP_CipherInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv,
sau);
    EVP_CipherUpdate(ctx, rez, &lrez, text, l);
    total_l = lrez;
    EVP_CipherFinal_ex(ctx, rez + total_l, &lrez);
    total_l += lrez;
    if (sau == 0) rez[total_l] = '\0';
    EVP_CIPHER_CTX_free(ctx);
    return total_l;
}
```

### 4.2 Protocol de Schimb de Chei Simetrice

```

RAND_bytes(cbc[client].key, 16);
RAND_bytes(cbc[client].iv, 16);
int l_key = encrypt_decrypt(key, iv, cbc[client].key, 16,
    key_encrypt, 1);
int l_iv = encrypt_decrypt(key, iv, cbc[client].iv, 16,
    iv_encrypt, 1);
write(client, &l_key, sizeof(int));
write(client, key_encrypt, l_key);
write(client, &l_iv, sizeof(int));
write(client, iv_encrypt, l_iv);

```

### 4.3 Parsare comandă şi verificare

```

while(str != NULL)
{
    int ok = 0;
    for(int k = 0; k < 7; k++)
        if(strcmp(str, oper[k]) == 0)
        {
            if(i % 2 == 0 && j == 0)
            {
                int bytes = encrypt_decrypt(cbc[fd].key, cbc[
                    fd].iv, "Comanda gresita\n", 16,
                    encrypted, 1);
                write(fd, &bytes, sizeof(int));
                write(fd, encrypted, bytes);
                l = 4; write(fd, &l, sizeof(int));
                write(fd, "DONE", 1);
                return 0;
            }
            args[i][j] = NULL; i++;
            args[i][0] = str; i++; j = 0; ok = 1;
        }
    if (ok == 0)
    { args[i][j] = str; j++; }
    str = strtok(NULL, " ");
}

```

### 4.4 Exucutare comandă cu redirecţionare output

```

//In function execute
if (strcmp(root->opr, ">") == 0)
{
    char path[1024];
    strcpy(path, client_dir[fd]);
}

```

```

    strcat(path, "/");
    strcat(path, root->right->com[0]);
    int fd_out = open(path, O_WRONLY | O_CREAT | O_TRUNC,
        0644);
    if (fd_out == -1)
    {
        char encrypted[50];
        int l = encrypt_decrypt(cbc[fd].key, cbc[fd].iv, "Eroare
            deschidere fisier output\n", 33, encrypted, 1);
        write(fd, &l, sizeof(int));
        write(fd, encrypted, l);
        return 1;
    }
    int outp = dup(STDOUT_FILENO);
    dup2(fd_out, STDOUT_FILENO);
    close(fd_out);
    execute(root->left, -1, 0, fd);
    dup2(outp, STDOUT_FILENO);
    close(outp);
    return 0;
}

```

## 4.5 Protocol la Nivelul Aplicației

### Inițializare Conexiune

- Serverul ascultă conexiunile pe portul 2703.
- Când un client se conectează, serverul inițializează cheile de criptare simetrică pentru acel client și le trimite.

### Autentificare

- Clientul trimite comanda "Conectare", urmată de numele utilizatorului și parola.
- Serverul validează utilizatorul și parola. În caz de succes, stabilește clientul ca autentificat.

### Transmiterea Comenzilor

- Comenzile sunt criptate pe client și decriptate pe server.
- Serverul analizează comanda și o descompune în comenzi simple și operatori, verifică dacă ordinea acestora este corectă apoi creează aborele comenzii pe care îl execută în ordine înfixată, conform regulilor operatorilor care sunt în rădăcină, comenzile simple se execută cu ajutorul unui proces copil, fork, și execvp.

## 4.6 Scenarii Reale de Utilizare

### Gestionare de Utilizatori

- Utilizatorii se autentifică folosind un sistem bazat pe un fişier JSON.
- Permite urmărirea utilizatorilor autentificaţi şi prevenirea autentificării multiple simultane.

### Executare Comenzi

- Identificarea şi parsarea comenzilor.
- Un operator este mereu între două comenzi sau între o comandă şi un fişier în cazul redirectionărilor, excepţie face comanda ; care se poate găsi la sfârşit fără a avea o comandă în dreapta, orice nu respectă structura asta este considerată comandă greşită sau incompletă şi nu se execută.
- Redirecţionarea output-ului către procesul părinte care trimite prin socket la client, excepţie fac comenzi care au output-ul redirecţionat spre un fişier (>) sau strerr-ul redirecţionat (2>).

## 4.7 Funcţii

- `read_json` şi `tranform`: citesc, parsează şi stochează informaţii despre utilizatori.
- `encrypt_decrypt`: Criptare/Decriptare.
- `priority`, `add`, `build`: construiesc arborele comenzii bazat pe prioritatea operatorilor.
- `execute`: execută arborele, comenzile simple prin `fork` si `execvp`, în cazul unui operator redirecţionează input-ul, output-ul, `stderr`-ul, verifică reuşita unei comenzi.
- `comcd`: funcţie specială pentru comanda `cd`.
- `autentificare` (în client): încearcă autentificarea până când datele sunt validate de server şi trimite mesajul de Autentificare cu succes.
- `mesaj`: autentifică clientul şi parsează imputul, validându-i structura (comandă operator comandă operator comandă ...).

## 5 Concluzii

Proiectul realizează o implementare a unui terminal SSH criptat, utilizând tehnologii precum TCP, OpenSSL şi JSON. Îmbunătăţirile posibile sunt realizarea unei interfeţe grafice care să se prezinte cu un meniu al comenzilor şi al argumentelor şi posibilitatea de adăugare a unui utilizator nou în cazul în care se încearca autentificarea unui utilizator care nu există.

## 6 Referințe Bibliografice

- <https://www.geeksforgeeks.org/cjson-json-file-write-read-modify-in-c/>
- <https://github.com/rbtylee/tutorial-jsonc>
- [https://docs.openssl.org/1.1.1/man3/EVP\\_EncryptInit/](https://docs.openssl.org/1.1.1/man3/EVP_EncryptInit/)
- <https://www.pandasecurity.com/en/mediacenter/what-is-aes-encryption/>
- <https://edu.info.uaic.ro/computer-networks/files/NetEx/S9/servTcpCSel.c>
- <https://edu.info.uaic.ro/computer-networks/files/NetEx/S9/cliTcp.c>