MY SSH

Toderuți Gheorghe Andrei

UAIC INFO IAȘI
andreitoderuti@gmail.com

1 Introducere

Proiectul propus urmărește implementarea unei aplicații de tip server-client, care să permită clientului să ruleze comenzi pe server, iar serverul să returneze rezultatele acestor comenzi.

Obiectivele proiectului sunt: implementarea unui mod prin care comunicarea între server și client se realizează în mod securizat criptând informațiile, implementarea unui sistem de autentificare la nivelul clientului ce se validează la nivelul serverului cu date stocate într-o bază de date asociată, suport pentru orice tip de comenzi, fie că sunt simple sau compuse (legate prin $|, <, >, 2>, \&\&, ||, ;$), capabilitatea de a gestiona clienți multipli simultan și de a gestiona modificarea directorului de lucru pentru fiecare în mod individual, pentru a nu fi afectați de schimbarea directorului (*cd*) într-un alt client.

2 Tehnologii Aplicate

Tehnologiile esențiale utilizate în dezvoltarea aplicației sunt:

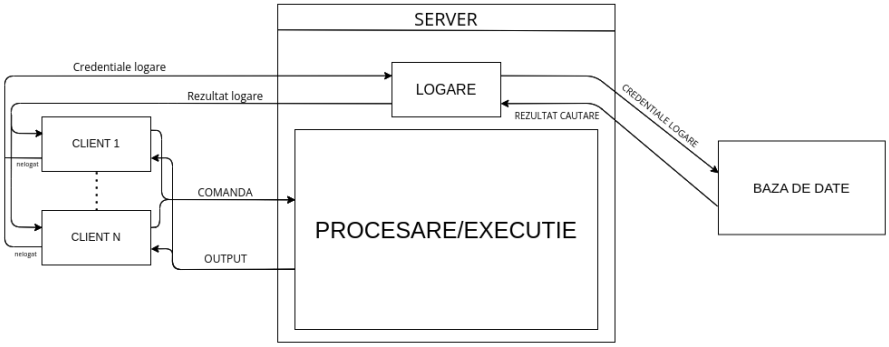
1. TCP
   Comunicarea este realizată utilizând TCP deoarece protocolul suportă conexiuni între două puncte diferite și asigură trimiterea integrală și ordonată a datelor între cele două capete, lucru important atât pentru introducerea comenzilor, cât și pentru transmiterea rezultatului.
2. Multiplexor I/O
   La protocolul TCP am adăugat un multiplexor implementat cu ajutorul funcției *select*() pentru a putea gestiona eficient și cu ușurință lucrul cu multipli clienți în paralel, fără a fi nevoie de multiple threaduri sau fork-uri.
3. Rivest-Shamir-Adleman (RSA)
   Pentru transmiterea sigură a cheilor AES către clienți, am utilizat algoritmul RSA. Acest algoritm de criptare asimetrică permite generarea unei perechi de chei (publică și privată), utilizate pentru criptarea și decriptarea datelor. Cheia publică este trimisă clientului, care o folosește pentru a cripta cheia AES. Serverul utilizează cheia sa privată pentru a decripta această informație. RSA oferă un nivel ridicat de securitate, asigurând confidențialitatea schimbului de chei.

4. Bază de date SQL_LITE
   În baza de date voi stoca informațiile utilizatorilor cu care aceștia se vor loga în aplicație. Am ales să implementez baza de date cu SQL_LITE datorită simplității cu care pot fi stocate datele într-un singur fișier, având și performanțe foarte bune la accesare.
5. Secure Hash Algorithm 256 (SHA256)
   În cadrul aplicației, SHA-256 este utilizat pentru a asigura securitatea parolelor stocate în baza de date. Fiecare parolă introdusă de utilizator este combinată cu un salt generat aleator și apoi trecută prin funcția de hash SHA-256. Rezultatul obținut (hash-ul) este stocat în baza de date în locul parolei propriu-zise. Această abordare asigură că parolele nu sunt stocate în formă clară .
6. Advanced Encryption Standard (AES)
   Criptarea la nivelul programului va fi realizată utilizând acest algoritm standardizat. Acesta conferă posibilitatea de a cripta blocuri de date (chiar și de dimensiuni mari) și funcționează cu o singură cheie atât pentru criptare, cât și pentru decriptare. Am ales această metodă de criptare deoarece este o metodă performantă, eficientă, ușor de implementat și foarte bine documentată.

3 Structura Aplicației

Aplicația funcționează cu un server și minimum un client. După autentificarea userului în client acesta va putea introduce în client orice comandă dorește. Comanda va fi trimisă în server unde va fi executată, răspunsul fiind returnat către client după execuție. Toate datele schimbate între client și server vor fi criptate și ulterior decriptate, atât la nivelul clientului cât și la nivelul serverului.

Diagrama următoare ilustrează funcționarea proiectului:



4 Aspecte de Implementare

Comunicarea se realizează în mod concurent utilizând un multiplexor bazat pe funcția select(), care permite gestionarea mai multor conexiuni simultan pe un singur fir de execuție.

```
bcopy ((char *) &actfds, (char *) &readfds, sizeof (readfds));
if (select (nfds+1, &readfds, NULL, NULL, &tv) < 0) {
    error ("[ERROR] select().\n");
}
if (FD_ISSET (sd, &readfds)) {
    ...
    client = accept(sd, (struct sockaddr *) &from, &len);
    ...
    FD_SET (client, &actfds);
    ...
}
```

Dupa stabilirea unei conexiuni intre un client si server toate comunicatiile se vor realiza in mod securizat. Aplicația utilizează criptarea pentru a proteja comunicațiile între client și server. Cheile AES pentru criptarea simetrică sunt generate individual pentru fiecare client și sunt transmise în siguranță prin criptare asimetrică RSA. Datele schimbate sunt criptate cu AES folosind vectori de inițializare ($init_vector$) unici pentru fiecare mesaj, asigurând confidențialitatea și integritatea comunicației.

```
    unsigned char init_vector[AES_BLOCK_SIZE];
RAND_bytes(init_vector, AES_BLOCK_SIZE);

// Setarea cheii pentru criptare
AES_KEY enc_key;
AES_set_encrypt_key(client_aes_keys[fd].data(),
AES_KEY_LENGTH * 8, &enc_key);

// Criptarea datelor
AES_cbc_encrypt(
    plain_buf.data(),                   // Datele originale
    encrypted_output.data(),            // Buffer pentru datele criptate
    length,                             // Lungimea datelor
    &enc_key,                           // Cheia de criptare
    init_vector,                        // Vector de initializare
    AES_ENCRYPT                         // Modul de criptare
);
...
// Setarea cheii pentru decriptare
AES_KEY dec_key;
AES_set_decrypt_key(client_keys[fd].data(),
```

```
AES_KEY_LENGTH * 8, &dec_key);

// Decriptarea datelor
AES_cbc_encrypt(
    encrypted_command.data(),          // Datele criptate
    decrypted_command.data(),          // Buffer pentru datele decriptate
    length,                            // Lungimea datelor
    &dec_key,                          // Cheia de decriptare
    init_vector,                       // IV folosit pentru criptare
    AES_DECRYPT                        // Modul de decriptare
);
```

Primul contact al utilizatorului cu aplicația necesită autentificare. Acesta are la dispoziție două comenzi: una pentru înregistrare și alta pentru logare. Procesul de autentificare are loc exclusiv pe server. Credentialele utilizatorilor sunt stocate într-o bază de date, iar pentru securitate, parolele sunt stocate sub formă de hash generat cu algoritmul SHA-256.

După autentificare, clientul poate introduce în terminal orice comandă. Aceasta va fi trimisă către server, procesată și executată într-un $fork()$, rezultatul fiind trimis înapoi către client.

Clientul rămâne conectat la server până la introducerea comenzii exit ce cauzează închiderea acestuia, respectiv deconectarea acestuia de la server. Serverul rămâne funcțional și gestionează în continuare clienți (rămași sau nou-conectați).

```
if (strcmp(command, "exit") == 0) {
    bzero(paths[fd].path, 256);
    close(fd);
    FD_CLR(fd, &actfds);
    return;
}
```

De asemenea, serverul implementează o structură care stochează pentru fiecare user conectat la server adresa acestuia. Astfel, în cazul în care un utilizator utilizează comenzi ce modifică directorul curent (precum cd), modificarea va fi resimțită la nivelul fiecărui client în parte, fără a influența ceilalți clienți (deja existenți sau care sunt acceptați ulterior).

```
typedef struct {
    char path[256];
}client_path;

client_path paths[MAX_CLIENTS];
char server_path[256];

int main {
    if (realpath("./", server_path) == NULL) {
```

```
        error("[ERROR] nu se poate server_path\n");
    }
    ...
    while (1) {
        ...
        client = accept(sd, (struct sockaddr *) &from, &len);
        strcpy(paths[client].path, server_path);
        chdir(paths[client].path);
        ...
    }

}
```

Execuția comenzilor este realizată utilizând *fork*(). Comanda este executată într-un proces fiu, redirecționată către pipe prin intermediul *dup*2(), preluată de către părinte și transmisă pe bucăți fiului, gestionând astfel și cantități mari de output.

Datorită stocării path-ului fiecărui client în structura de mai sus, procesele fiu nu trebuie să rămână permanent deschise pentru fiecare client. Acestea sunt create și utilizate doar atunci când este necesar să se execute o comandă, optimizând astfel resursele utilizate ale sistemului și performanța aplicației.

```
    int pipefd[2];
if (pipe(pipefd) < 0 ) {
    error("[ERROR]: PIPE");
}
int pid= fork();
if (pid == 0) {
    close(pipefd[0]);
    dup2(pipefd[1], STDOUT_FILENO);
    ...
    execlp(command,command, args, NULL);
    ...
}
else {
    close(pipefd[1]);
    waitpid(pid, NULL, 0);
    while (status = read(pipefd[0], output, SIZE) > 0) {
            int bytes = strlen(output);
            write (fd, &bytes, sizeof(int));
            write (fd, output, bytes);
            bzero(output, SIZE);
        }
        close(pipefd[0]);
}
```

5 Concluzii

După părerea mea, îmbunătățirea proiectului s-ar putea realiza prin:

- Implementarea execuției comenzilor prin metode precum thread pooling.
- Implementarea unei baze de date mai complexe, ce să gestioneze și directorul curent pentru clienți.
- Crearea unei interfețe ce să suporte accesul comenzilor anterioare (prin arrow keys) și listarea componentelor directorului curent (tasta TAB în bash), poate chiar si alte lucruri originale.

Totodată, proiectul are nevoie de o testare riguroasă pentru a înțelege performanța acestuia în condiții de stres, dar și cât de scalabil este acesta.

References

1. https://edu.info.uaic.ro/computer-networks/files/7rc_ProgramareaInReteaIII_En.pdf
2. https://edu.info.uaic.ro/computer-networks/files/5rc_ProgramareaInReteaI_en.pdf
3. https://edu.info.uaic.ro/computer-networks/files/NetEx/S9/servTcpCSel.c
4. https://edu.info.uaic.ro/computer-networks/files/NetEx/S9/cliTcp.c
5. https://en.wikipedia.org/wiki/Secure_Shell