Documentatie MySSH

Albert Alexandru

Decembrie 2023

Universitatea "Alexandru Ioan Cuza" Iasi
Facultatea de Informatica

1 Introducere

Scopul acestui proiect este realizarea unui mecanism de comunicare securizata intre o aplicatie client si un server. Clientul va putea folosi terminalul de pe server pentru orice fel de comenzi bash (inclusiv comenzi multiple legate si redirectii), cu oricate argumente, iar serverul va returna outputul comenzii, intr-un format care nu poate fi interpretat in cazul in care ar fi interceptat in timpul transmisiei. Serverul va putea accepta si deservi mai multi clienti concomitent, acesta fiind unul concurent.

2 Tehnologii Aplicate

Pentru a asigura o comunicare eficienta si sigura intre cele doua aplicatii la nivelul retelei si al transportului, vom opta pentru o conexiune stabila si securizata in ceea ce priveste transmiterea datelor. Avand in vedere ca rapiditatea primirii raspunsurilor nu este un factor critic (ceea ce ar justifica utilizarea protocolului UDP) si ca integritatea datelor transmise este absolut esentiala, vom alege protocolul TCP pentru transmiterea datelor la nivelul transport.

In cadrul proiectului MySSH, integritatea datelor pe parcursul transportului este vitala. Oricare mica eroare in ordinea primirii pachetelor sau in continutul datelor ar incapacita comunicarea intre client si server. De exemplu, o cheie AES sau RSA care se corupe in timpul transportului ar face decriptarea oricarui mesaj imposibila. Acelasi lucru se aplica si pentru mesajele care se corup in timpul transmisiei, deoarece acestea nu ar mai putea fi decriptate.

3 Structura Aplicatiei

Arhitectura acestei aplicatii este construita pe modelul Client/Server concurent. Serverul are capacitatea de a accepta un numar definit de clienti, creand o instanta separata pentru fiecare. Aceste instante vor functiona simultan pentru a servi clientii.

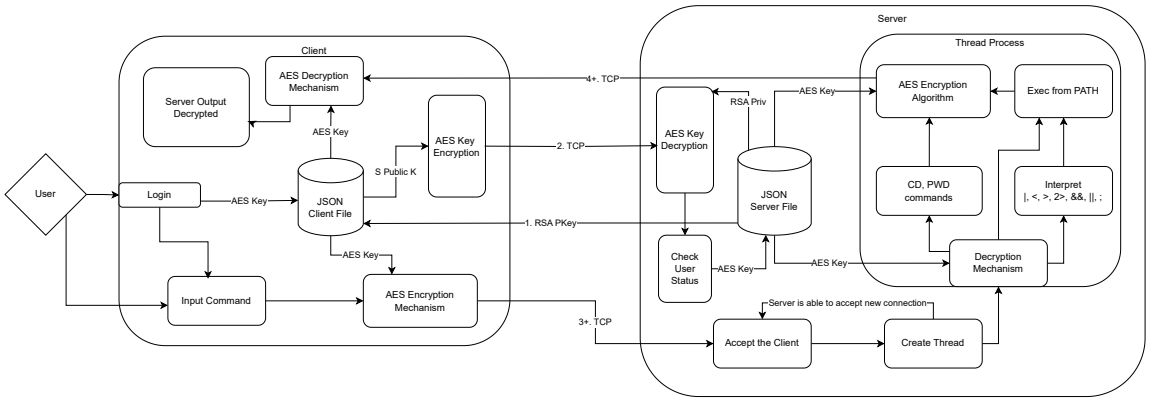

Figura 1: Diagrama MySSH

4 Aspecte de Implementare

Implementare generala:

In ceea ce priveste stocarea datelor, fiecare aplicatie foloseste un fisier JSON. Fisierul JSON al aplicatiei client va contine username-uri, parole si cheia AES unica pentru fiecare instanta a clientului. Pe de alta parte, fisierul JSON al serverului va stoca username-urile, cheile AES corespondente si cheile RSA ale serverului, atat cea publica, cat si cea privata.

Comunicarea initiala intre client si server se va face in mod asimetric folosind cheile RSA. La conectarea unui client nou, serverul va trimite cheia sa publica RSA, nefiind criptata. Dupa autentificarea utilizatorului in aplicatie, se va genera o cheie AES. Clientul va folosi cheia publica RSA pentru a cripta cheia AES si apoi o va trimite serverului. Serverul va folosi cheia sa privata RSA pentru a decripta cheia AES si o va stoca.

Ulterior, comunicarea va trece la criptare si decriptare simetrica, pentru eficienta sporita. De asemenea, fiecare mesaj criptat va fi codificat in baza 64 inainte de a fi trimis prin retea. La fel, serverul va decoda mesajele din baza 64 apoi le va decripta.

Pentru criptare si codificare, utilizam biblioteca OpenSSL 3. Cheile AES folosite au o lungime de 256 de biti, iar cele RSA de 1024 de biti.

Structura Pachetului:

Structura pachetului trimis de cele doua aplicatii este urmatoarea: Header-ul contine lungimea mesajului si este necriptat, pentru a permite alocarea dinamica a memoriei pentru continutul informatiei. Partea urmatoare este formata din informatia propriu-zisa, criptata.

Primul mesaj trimis de client, cel care contine cheia AES, va avea in fata cheii, concatenat, numele utilizatorului, pentru a permite serverului sa actualizeze corect cheia AES.

Mesajele primite de la server vor avea, similar, in fata output-ului, concatenat path-ul curent, pentru afisarea acestuia de catre client.

Executarea Comenzilor:

Pentru executarea comenzilor primite de la client, serverul foloseste functia ‘execlp‘. Pentru ‘pwd‘, se returneaza stringul cu path-ul curent al serverului. Pentru ‘cd‘, se verifica existenta folderului tinta si, daca exista, se executa schimbarea folosind ‘chdir()‘.

In cazul primirii de comenzi multiple sau redirectionari (|, <, >, 2 >, &&, ||, ;), serverul le va

trata. Pentru pipe-uri, se creeaza doua procese copil. Prima comanda se executa si output-ul ei este directionat catre pipe folosind ‘dup2()‘. Al doilea proces copil executa comanda urmatoare, redirectionand output-ul catre un string returnat clientului.

Ceilalti operatorii sunt implementati similar. Comenzile sunt transformate, initial, in forma postfixata, apoi se construieste un AST, unde fiecare comanda este o frunza, iar fiecare operator este un parinte.

Ordinea de precedenta a operatorilor se imparte in doua categorii: cei care trebuie executati imediat (ca < si 2 >), si cei care pot executa si comenezi procesate, cum ar fi |, >, &&, ||. Operatorul ; este procesat primul, determinand numarul de arbori creati.

Deoarece unii operatori au nevoie de statusul executiei comenzii precedente, fiecare functie care proceseaza un operator va returna o structura in care se va afla outputul operatiei, statusul operatiei, si o variabila bool care va stoca starea in care se afla nodul arborelui (procesat/neprocesat). Aceasta functie este necesara, spre exemplu, pentru o inaltuire de pipe-uri. Daca operatorul pipe nu ar stii ca fiul stang a fost executat deja, acesta ar incerca sa-l introduca ca argument intr-un execlp, producand o eroare.

Situatii Reale de Utilizare:

Proiectul emuleaza un protocol existent si foarte util, oferind utilizari diverse si concrete. Este ideal pentru accesarea si operarea unui calculator performant de la distanta, facilitand lucrul cu servere sau computere personale fara a fi fizic prezent. Acest lucru este extrem de util pentru administratorii de sistem care trebuie sa monitorizeze sau sa gestioneze computere si servere.

De asemenea, protocolul simplifica transferul de fisiere intre client si server, fiind eficient pentru schimbul rapid de documente si informatii. In contextul suportului tehnic, permite rezolvarea problemelor de sistem de la distanta, ceea ce este vital pentru companiile cu angajati in diverse locatii sau pentru cei care lucreaza de acasa.

Interpretarea Comenzilor

```
std::vector<std::string> tokens = tokenize(command);
std::vector<std::vector<std::string>> sub_commands = split_commands(tokens);

for (const auto& sub_command : sub_commands) {
    if (!contains_bash_operator_vec(sub_command)) {
        command_output += execute_cmd_without_operators(sub_command, path);
    }
    else {
        std::vector<std::string> postfix = convert_to_postfix(sub_command);
        TreeNode* root = construct_AST(postfix);
        CommandResult output = traverse_and_execute(root, path);
        command_output += output.output;
    }
}
```

Schita pentru algoritmul RSA:

```
void generate_RSA_keys(int& public_key, int& private_key, int& n) {
    srand(time(0));

    int p = generate_prime();
    int q = generate_prime();

    n = p * q;

    int phi = (p - 1) * (q - 1);

    public_key = rand() % (phi - 2) + 2;
    while (gcd(public_key, phi) != 1) {
        public_key = rand() % (phi - 2) + 2;
    }

    private_key = modInverse(public_key, phi);
}
```

5 Posibile Imbunatatiri

Securitate:

Pentru o securizare sporita a transportului, mai multe adaptari se pot implementa:

In primul rand, pentru ambiguizarea interpretarii continutului mesajului, deoarece acesta inca poate fi interceptat in retea (dar nu si decodat), acesta se poate combina cu un sir random de caractere. In acest caz va trebui adaugat un al doilea header care sa contina lungimea stringului generat aleatoriu. Serverul si algorimul de analiza a mesajului din server va trebui modificat la randul lui.

O a doua imbunatarire a securitatii ar putea fi adusa prin introducerea unui nou segment in structura packetului transmis intre cele 2 aplicatii. In acest caz, la sfarsitul packetului va fi adaugat un cod generat de algorimul HMAC-SHA256. Acest cod poate asigura autentificarea si integritate pachetului, pentru a preveni manipularea si coruperea datelor, proces care poate fi intampinat in retea. **Functionalitate:**

O functie aditionala ar putea permite comunicarea intre useri. Deoarece serverul contine cheile AES a tuturor userilor, acesta ar putea fi folosit pentru comunicare intre useri sau pentru redirectionari. Ex: Userul 1 execute o comanda in server iar aceasta apare in clientului 1 dar si in clientului catre care a redirectionat mesajul.

Referinte Bibliografice

[1] Mental Outlaw, How SSH Works `https://www.youtube.com/watch?v=5JvLV2-ngCI&ab_channel=MentalOutlaw`

[2] Rob Edwards, SSH Keys `https://www.youtube.com/watch?v=dPAw4opzN9g&t=489s&ab_channel=RobEdwards`

[3] Akamai Developer, SSH Key Authentication — How to Create SSH Key Pairs https://www.youtube.com/watch?v=33dEcCKGBO4&ab_channel=AkamaiDeveloper

[4] Wikipedia, Secure Shell https://en.wikipedia.org/wiki/Secure_Shell

[5] Wikipedia, RSA Algorithm https://en.wikipedia.org/wiki/RSA_(cryptosystem)

[6] Wikipedia, AES https://en.wikipedia.org/wiki/Advanced_Encryption_Standard