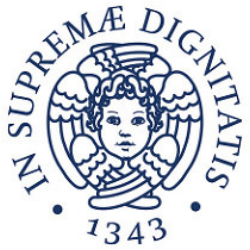

UNIVERSITÀ DEGLI STUDI DI PISA

Computer Engineering

Progetto di Cybersecurity

GERARDO ALVARO

DANIELA COMOLA

RICCARDO POLINI

a.a. 2018/2019

INDICE

1. INTRODUZIONE

Il progetto consiste in un'applicazione client-server che permette il download e l'upload di file in maniera sicura. È stato utilizzato il linguaggio di programmazione C++ e la libreria di OpenSSL per gli algoritmi di cifratura. La comunicazione, che avviene mediante protocollo di trasporto UDP, è confidenziale, autenticata e resistente a diversi dei più comuni tipi di attacchi informatici.

2. DESCRIZIONE APPLICAZIONE

2.1 Repository

La repository del progetto è organizzata come segue:

- **`/certif`**: contiene tutti i certificati e le chiavi private dei client e del server, il certificato di Simple Authority e la lista dei certificati revocati, tutto in formato PEM.
- **`/download:`** contiene la lista dei file in possesso del client che può caricare nel server.
- **`/serv_files`**: contiene la lista dei file in possesso del server che possono essere scaricati dai client autorizzati.
- **`/src:`** contiene i codici sorgente dell'applicazione.

Nell'applicazione sono già configurati i seguenti client:

- Gerardo: client autorizzato con certificato valido.
- Riccardo: client autorizzato con certificato revocato.
- Daniela: client non autorizzato con certificato valido.

2.2 Comandi

Questa è la lista dei comandi disponibili per l'applicazione:

- **`!help:`** mostra a video la lista di comandi disponibili.
- **`!upload:`** consente al client di caricare un file nel server, il file caricato deve avere un nome diverso da quelli dei file già presenti nel server.
- **`!get:`** consente al client di scaricare un file dal server.
- **`!list:`** consente al client di ottenere una lista dei file disponibili nel server.
- **`!quit:`** disconnette il client.
- **`!squit:`** chiude sia client che server.

3. SCELTE IMPLEMENTATIVE

Analizziamo ora alcune delle più significative scelte che abbiamo effettuato nella realizzazione dell'applicazione:

- L'applicazione gestisce i dati in input e output come un continuo stream di byte che termina con la chiusura della sessione.
- Le operazioni di download e upload sono *memory-efficient* infatti ogni file, la cui dimensione può raggiungere i 4GiB, quando trasmesso viene suddiviso in "CHUNK" di grandezza di 512KiB.
- Sia il server che i client sono autenticati con un certificato pubblico rilasciato da un'autorità certificata.
- Il server, oltre a verificare la validità del certificato ricevuto, controlla che il client appena collegato sia presente in una lista privata di client autorizzati presso di lui. In caso contrario chiude la connessione.
- Il client, una volta verificata la validità del certificato del server con cui si è appena collegato, ne visualizza a schermo il nome e può decidere se proseguire o meno con la connessione, se lo reputa affidabile.
- La confidenzialità della comunicazione è garantita dal metodo di cifratura AES128 in modalità CFB8 (Cipher Feed-Back a 8bit). Grazie a questa modalità non è richiesta l'aggiunta del *padding* poiché i messaggi verranno cifrati un byte per volta. Lo schema della modalità CFB è riportato in Figura 1.

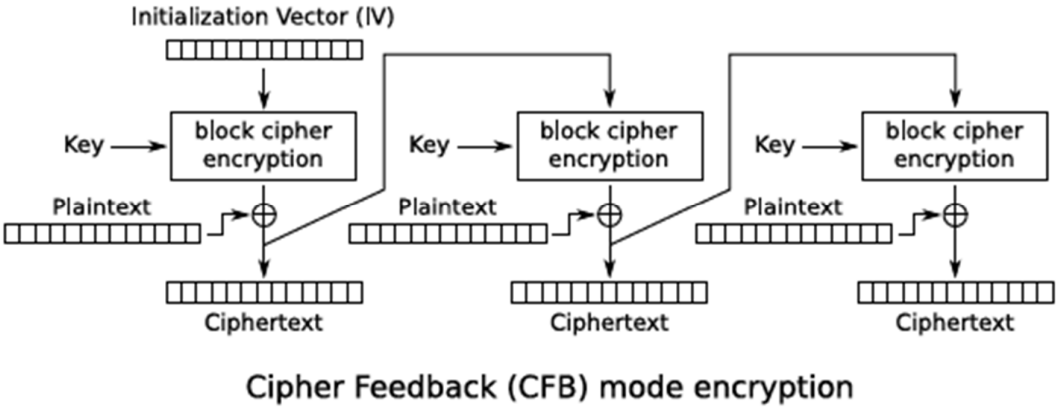

Figura 1: Schema CFB mode

- La grandezza della chiave di sessione per la cifratura e del vettore di inizializzazione sono prestabiliti dal metodo di cifratura AES128 su 16 byte.

- L'autenticazione dei messaggi è ottenuta grazie alla modalità di autenticazione HMAC; i digest, su 32 Byte, sono ottenuti dall'esecuzione della funzione hash SHA-256 sul messaggio al quale è stato preventivamente concatenato il relativo numero di sequenza.
- Il numero di sequenza relativo ad ogni messaggio è su 4 byte consentendo così lo scambio di un massimo di 2^{32} messaggi con numero di sequenza univoco per ogni sessione. Al raggiungimento di UINT32_MAX la sessione viene chiusa.

4. PROTOCOLLO SCAMBIO CHIAVI

4.1 Handshake

Per stabilire una sessione sicura è stato implementato un handshake a 3 vie mostrato in Figura 2.

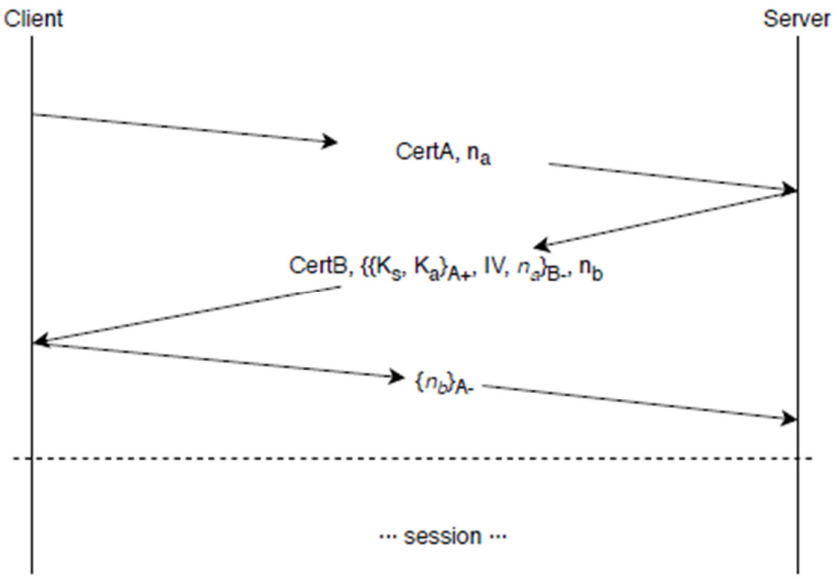

Figura 2: Handshake iniziale

- ***certA*** è il certificato del client rilasciato da un'entità pubblica fidata per la creazione dei certificati (Simple Authority).
- n_a è il nonce del client per garantire la freschezza della sessione e proteggere la comunicazione da attacchi di tipo Replay.
- ***certB*** è il certificato del server.
- K_s e K_a sono le chiavi simmetriche usate durante la sessione per cifrare e autenticare i messaggi rispettivamente; vengono generate dal server in maniera randomica e vengono inviate al client criptandole con la chiave pubblica del client A^+.
- ***IV*** è il vettore di inizializzazione usato nella prima iterazione della cifratura in modalità CFB usata per cifrare la sessione.
    - le chiavi simmetriche K_s e K_a, il vettore di inizializzazione e il *nonce* del client, vengono firmate digitalmente con la chiave privata del server B^-.
- n_b è il nonce del server.
    - infine il client invia il *nonce* del server firmato con la propria chiave privata A^-

4.2 Dimostrazione con Ban Logic

La correttezza dell'handshake esposto sopra viene dimostrata con l'utilizzo della Ban Logic.

Gli obiettivi del protocollo sono l'autenticazione e la freschezza delle chiavi di sessione che in termine di Ban Logic possono essere definite come segue:

$$A \models A \overset{k_s, k_a}{\longleftrightarrow} B$$
$$A \models \#(A \overset{k_s, k_a}{\longleftrightarrow} B)$$

Possono essere fatte le seguenti assunzioni riguardo:

- chiavi

$$A \models \overset{CA^+}{\longmapsto} CA$$
$$B \models \overset{CA^+}{\longmapsto} CA$$
$$B \models A \overset{k_s, k_a}{\longleftrightarrow} B$$

- "fiducia"

$$A \models B \Rightarrow A \overset{k_s, k_a}{\longleftrightarrow} B$$
$$A \models B \Rightarrow \#(A \overset{k_s, k_a}{\longleftrightarrow} B)$$
$$A \models CA \Rightarrow (cert_A, cert_B)$$
$$B \models CA \Rightarrow (cert_A, cert_B)$$

- Freschezza

$$A \models \#(N_A)$$
$$B \models \#(N_B)$$
$$B \models \#(A \overset{k_s, k_a}{\longleftrightarrow} B)$$

Nell'handshake visto nella sezione precedente l'unico passaggio che merita una dimostrazione con Ban Logic è il secondo, quello dal server verso il client, poiché il primo avviene in chiaro non trasportando informazioni critiche e l'ultimo rappresenta solamente la conferma dell'avvenuta ricezione del secondo passaggio.

La via di handshake considerata (M2) può essere idealizzata nella seguente maniera:

$$M2 : B \to A : \{cert_B\}_{CA^-}, \{IV, N_A, \#(A \overset{k_s,k_a}{\longleftrightarrow} B), \{A \overset{k_s,k_a}{\longleftrightarrow} B\}_{A^+}\}_{B^-}$$

Dimostrazione.

Postulato 1.

$$\frac{A \models \overset{CA^+}{\longmapsto} CA, A \triangleleft \{cert_B\}_{CA^-}}{A \models CA \mid\sim cert_B}$$

Postulato 2.

$$\frac{A \models \#(certB), A \models CA \mid\sim cert_B}{A \models CA \models certB}$$

La freschezza di certB è garantita dal suo range di validità verificato dalla lista dei certificati revocati.

Postulato 3.

$$\frac{A \models CA \models cert_B, A \models CA \Rightarrow cert_B}{A \models cert_B}$$

Ricordiamo che la chiave pubblica di B è parte di certB e consideriamo come $payload_{M2}$ il contenuto inviato dal server al client.

Postulato 1.

$$\frac{A \models \overset{B^+}{\longmapsto} B, A \triangleleft \{payload_{M2}\}_{B^-}}{A \models B \mid\sim payload_{M2}}$$

Postulato 2.

$$\frac{A \models \#(payload_{M2}), A \models B \mid\sim payload_{M2}}{A \models B \models payload_{M2}}$$

La freschezza del $payload_{M2}$ è garantita dal nonce del client; visto che il $payload_{M2}$ contiene le due chiavi di sessione e la freschezza delle stesse possiamo affermare che:

$$A \models B \models A \overset{k_s,k_a}{\longleftrightarrow} B$$

$$A \models B \models \#(A \overset{k_s,k_a}{\longleftrightarrow} B).$$

Postulato 3.

$$\frac{A \models B \models A \overset{k_s,k_a}{\longleftrightarrow} B, A \models B \Rightarrow A \overset{k_s,k_a}{\longleftrightarrow} B}{A \models A \overset{k_s,k_a}{\longleftrightarrow} B}$$

Postulato 3.

$$\frac{A \models B \models \#(A \overset{k_s,k_a}{\longleftrightarrow} B), A \models B \Rightarrow \#(A \overset{k_s,k_a}{\longleftrightarrow} B)}{A \models \#(A \overset{k_s,k_a}{\longleftrightarrow} B)}$$

C.V.D.

5. RESISTENZA AGLI ATTACCHI

L'applicazione è stata progettata per evitare attacchi di Eavesdropping, Oracle e Replay. Non è invece garantita, poiché non richiesta, la Perfect Forward Secrecy. Infine grazie alla funzione *regex_match()* lato server viene controllato che i nomi dei file inviati dal client contengano solo caratteri consentiti.

5.1 Eavesdropping

Un avversario potrebbe provare a spiare la conversazione tra client e server.

Nell'applicazione da noi proposta, la conversazione è confidenziale in quanto ogni messaggio è cifrato con l'algoritmo AES-128 in modalità Cipher Feed-Back (CFB). Pertanto, anche se l'avversario memorizzasse ogni messaggio cifrato, non riuscirebbe comunque a decifrarli senza la chiave di sessione.

5.2 Oracle attack

Un'applicazione è sensibile a questo tipo di attacco qualora decifri il messaggio ricevuto prima di verificarne la sua integrità. Quindi per essere resistenti a Oracle attack la decifratura del messaggio viene eseguita solo qualora l'autenticazione del messaggio dia esito positivo, ovvero, l'hmac ricevuto è uguale all'hmac calcolato sul messaggio corrispondente.

5.3 Replay Attack

Un avversario potrebbe memorizzare alcuni messaggi inviati da uno dei due peer all'altro e, successivamente, potrebbe inviarli nuovamente facendo credere al peer ricevente che sia il peer legittimo a inviarglieli

Per contrastare gli attacchi di tipo replay, l'applicazione contrassegna ogni messaggio con un numero di sequenza incrementale. Se il numero di sequenza atteso dal peer destinatario non corrisponde con quello del messaggio arrivato la connessione viene chiusa.

Il numero di sequenza in questione viene implementato come un contatore e qualora durante una sessione venga raggiunto il suo valore massimo la sessione viene chiusa.

5.4 Perfect Forward Secrecy

L'applicazione proposta non garantisce la Perfect Forward Secrecy: qualora un utente malevolo entrasse in possesso della chiave privata del client, sarebbe in grado di decifrare le chiavi di sessione, criptate in fase di handshake con la chiave pubblica del client, e di conseguenza tutte le sessioni di messaggi precedenti tra il server e il client di cui ha ottenuto la chiave privata.

La garanzia di questa proprietà esula dallo scopo di questo lavoro tuttavia, qualora si volesse estendere questa applicazione in futuro per garantire la PFS, le chiavi di sessione dovrebbero essere cifrate con una chiave simmetrica effimera che venga eliminata al termine del suo utilizzo.