Relazione del progetto “Secure Online Backup” per il corso di Sicurezza nei Sistemi Informatici

Specifica e analisi del protocollo

Il progetto in questione si basa sulle specifiche dello scenario tre. Le linee guida prevedono la creazione di un’applicazione distribuita secondo il paradigma “client-server”, in cui il server ha una coppia di chiavi, formata dalla sua chiave pubblica e dalla sua chiave privata. Si assume che i client conoscano la chiave pubblica del server, e che condividano con quest’ultimo una password segreta. Il protocollo crittografico deve prevedere che al termine della sua esecuzione venga stabilita una chiave di sessione tra client e server; e che il client possa ritenere che il server disponga della chiave di sessione e viceversa. Il che significa imporre come obiettivi la key authentication e la key confirmation. Una lieve differenza di questo progetto rispetto alle specifiche è data dal fatto che oltre ad una password segreta condivisa tra client e server, vi è anche un username che il client dovrà inserire all’atto dell’autenticazione con il server. Si avranno quindi due segreti condivisi (o un unico segreto condiviso grande quanto la somma delle dimensioni di utente e password). La possibilità di un nome utente è stata aggiunta per dare una maggiore flessibilità all’applicazione.
Passiamo ora all’analisi del protocollo.

Protocollo reale

$$M1. \quad C \rightarrow S \quad E_{k_{S+}}(K_{CS}\,, N_C)$$

$$M2. \quad S \rightarrow C \quad E_{k_{CS}}(N_C + 1\,, N_S)$$

$$M3. \quad C \rightarrow S \quad E_{k_{CS}}(utente\,, password\,, N_S + 1)$$

Il protocollo si compone di tre messaggi. E’ il client a iniziare l’handshake con il server. E lo fa inviando al server, la chiave di sessione K_{CS} e il nonce N_C da lui generati, entrambi cifrati con la chiave pubblica del server (che il client conosce per ipotesi). I nonce generati da client e server sono dei numeri casuali.
Il server decifra con la sua chiave privata il messaggio, e risponde inviando un pacchetto cifrato con K_{CS} contenente un nonce generato dal server (N_S) e N_C incrementato di un’unità.
Infine il client riceve il pacchetto, e lo decifra con la chiave K_{CS} e verifica che il suo nonce sia stato correttamente incrementato dal server. In seguito invia un messaggio con utente, password e N_S incrementato di un’unità; il tutto cifrato con la chiave di sessione K_{CS}. Il server verificherà la correttezza di utente, password e nonce inviati dal client.

Protocollo ideale

$M1. \quad C \rightarrow S \quad \{(C \overset{K_{CS}}{\leftrightarrow} S), N_C\}_{K_{S+}}$

$M2. \quad S \rightarrow C \quad \{N_S, (C \overset{K_{CS}}{\leftrightarrow} S), N_C\}_{K_{CS}}$

$M3. \quad C \rightarrow S \quad \{N_S, (C \overset{K_{CS}}{\leftrightarrow} S), (C \overset{utente}{\leftrightharpoons} S), (C \overset{password}{\leftrightharpoons} S)\}_{K_{CS}}$

Ipotesi

1. $S \mid \equiv C \Longrightarrow C \overset{K_{CS}}{\leftrightarrow} S$
2. $S \mid \equiv C \overset{utente}{\leftrightharpoons} S$
3. $S \mid \equiv (C \overset{password}{\leftrightharpoons} S)$
4. $C \mid \equiv (C \overset{utente}{\leftrightharpoons} S)$
5. $C \mid \equiv (C \overset{password}{\leftrightharpoons} S)$
6. $C \mid \equiv \#(C \overset{K_{CS}}{\leftrightarrow} S)$
7. $C \mid \equiv \#(N_C)$
8. $S \mid \equiv \#(N_S)$
9. $C \mid \equiv \overset{K_{S+}}{\mapsto} S$
10. $S \mid \equiv \overset{K_{S+}}{\mapsto} S$

Obiettivi

1. $C \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S)$
2. $S \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S)$
3. $C \mid \equiv S \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S)$
4. $S \mid \equiv C \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S)$

Dimostrazione

L'obiettivo al punto 1 è già stato raggiunto per l'ipotesi 6, dato che è il client stesso a generare la chiave di sessione K_{CS}.
Dopo M1 (per message meaning rule):

$$\frac{S \rhd \{ C \overset{K_{CS}}{\leftrightarrow} S \}_{K_{S+}}, \; S| \equiv \overset{K_{S+}}{\mapsto} S}{S \mid \equiv C \mid\sim C \overset{K_{CS}}{\leftrightarrow} S}$$

Dopo M2 (per message meaning rule e per nonce verification rule):

$$\frac{C \rhd \{C \overset{K_{CS}}{\leftrightarrow} S\}, \; C \mid \equiv C \overset{K_{CS}}{\leftrightarrow} S}{C \mid \equiv S \mid\sim C \overset{K_{CS}}{\leftrightarrow} S}$$

$$\frac{C \mid \equiv S \mid\sim C \overset{K_{CS}}{\leftrightarrow} S, \; C \mid \equiv \#(C \overset{K_{CS}}{\leftrightarrow} S)}{C \mid \equiv S \mid \equiv C \overset{K_{CS}}{\leftrightarrow} S}$$

Dopo M3 (per nonce verification rule e per jurisdiction rule):

$$\frac{S \mid \equiv C \mid\sim (C \overset{K_{CS}}{\leftrightarrow} S), \; S \mid \equiv \#(C \overset{K_{CS}}{\leftrightarrow} S)}{S \mid \equiv C \mid \equiv C \overset{K_{CS}}{\leftrightarrow} S}$$

$$\frac{S \mid \equiv C \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S), \; S \mid \equiv C \Rightarrow (C \overset{K_{CS}}{\leftrightarrow} S)}{S \mid \equiv (C \overset{K_{CS}}{\leftrightarrow} S)}$$

Progettazione

L'applicazione ha come scopo quello di offrire all'utente una sorta di servizio di cloud computing molto semplice. Nello specifico, la possibilità di salvare dei file su un computer remoto in confidenzialità, e di effettuare su di essi diverse operazioni. Essendo un'applicazione di rete e quindi di tipo distribuito, consta di un client e di un server. Per avviare il server occorre specificare la porta su cui quest'ultimo deve rimanere in attesa di eventuali richieste dal client. Mentre per eseguire il client occorre specificare l'ip e la porta di ascolto del server. Tutta l'applicazione è stata sviluppata in ambiente Mac Os X (Unix based) e compilata con GCC. Il linguaggio di programmazione scelto è il C. Il server supporta il multi thread, il che gli consente di servire più client in "contemporanea". La strategia multi thread utilizzata è quella del pool di thread, ovvero, la creazione di un certo numero di thread all'avvio, che resteranno in attesa di nuove richieste dai client. Data la semplicità

dell'applicazione (e il suo fine didattico) si è deciso di utilizzare un'interfaccia utente a riga di comando (CLI), e di utilizzare dei file per la memorizzazione di dati permanenti, al posto di un database. Visto l'utilizzo dei thread e della crittografia, nel progetto verranno ampiamente utilizzate funzioni presenti nelle librerie "pthread" e "openssl".

Il progetto è suddiviso sostanzialmente in 3 moduli: modulo client ("backup-client.c"), modulo server ("backup-server.c") e modulo di utilità ("utility.h") . Il modulo client è costituito da istruzioni utili per stabilire una connessione con il server e fornire un'interfaccia all'utente. Dualmente, nel modulo server vi sono istruzioni per la creazione di thread e l'attesa di nuove connessioni. Sia il modulo client che quello server sono molto legati al modulo utilità, nel quale sono implementate le funzionalità di sicurezza e quelle per la gestione ed esecuzione corretta dei thread. In particolare, per quanto riguarda la sicurezza, sono state implementate diverse funzioni facilmente richiamabili da client e server, in modo da snellire il codice in tali moduli.

Le più importanti, e le più utilizzate, sono la "inviaMessaggio", e la "riceviMessaggio". Queste funzioni vengono utilizzate dopo che il client ha inviato la chiave simmetrica di sessione al server. La "inviaMessaggio" dati in ingresso un socket, un buffer, la dimensione di tale buffer e una chiave di cifratura; cifra il buffer (testo o binario) con la chiave in ingresso e lo invia tramite il socket indicato, occupandosi di tutti i passaggi intermedi per il compimento di tale funzione (allocazione/deallocazione del contesto di cifratura, invio delle dimensioni dei dati…). Dualmente la "riceviMessaggio" dati in ingresso un socket e una chiave, si occupa di ricevere un buffer cifrato di dimensione non nota a priori, decifrare tale buffer e restituire il puntatore al buffer in chiaro.

L'instaurazione di un canale sicuro tra client e server avviene tramite il metodo della digital envelope: il client invia al server la chiave simmetrica di sessione, da lui generata, cifrata con la chiave pubblica del server. Tutte le comunicazioni successive verranno cifrate con tale chiave simmetrica. Per ciò che riguarda la cifratura asimmetrica è stato impiegato l'algoritmo RSA, mentre per la cifratura a chiave simmetrica è stato usato l'algoritmo DES ECB. Le chiavi pubblica e privata del server (preesistenti) sono memorizzate rispettivamente in 2 file ("pub.pem" e "priv.pem"). Il file contenente la chiave pubblica del server è a disposizione del client mentre la chiave privata è a disposizione del server. Per consentire l'invio e la ricezione della chiave simmetrica di sessione generata dal client, sono state implementate 2 funzioni: "inviaChiaveSim" e "riceviChiaveSim". La prima genera e cifra con la chiave pubblica del server, la chiave di sessione e la invia. La seconda riceve, decifra con la chiave privata del server, e restituisce la chiave di sessione.

Una volta che il client è riuscito a stabilire un canale sicuro con il server, l'utente dovrà inserire il proprio nome utente e la propria password per accedere al servizio. Il modulo client si occuperà di eseguire opportuni controlli su formato e dimensione degli input immessi dall'utente prima di inviarli al server. Quest'ultimo controllerà che tali credenziali siano corrette confrontandole con quelle contenute in un apposito file opportunamente formattato. L'applicazione non offre un servizio di registrazione di un nuovo account. Pertanto tale servizio dovrà essere svolto da un'applicazione terza (ad esempio un'applicazione web) che dovrà provvedere ad un opportuno controllo dei nomi utente e delle password scelte dall'utente all'atto della registrazione (controlli su dimensione, e su presenza o meno di

caratteri non alfanumerici). Tale applicazione terza dovrà inoltre memorizzare su file tali dati nel formato “utente\npassword\n\n”.

Se username e password inserite dall’utente sono corretti, verrà presentato il menù principale contenente tutte le possibili funzioni da richiamare, con affianco il numero corrispondente ad una determinata funzione. Le funzioni disponibili sono:

- Invio di un file
- Visualizzazione dell’elenco dei file caricati
- Ricezione di un file precedentemente caricato
- Rimozione di un file precedentemente caricato
- Rimozione di tutti i file precedentemente caricati
- Terminazione del programma

Inserendo il numero corrispondente alla funzione che si desidera richiamare, verranno eseguite tutte quelle operazioni che serviranno per esaudire la richiesta dell’utente. Nel caso dell’invio di file, ad esempio, l’utente dovrà inserire il percorso in cui si trova il file che desidera inviare al server. Sul server ogni utente avrà a disposizione una cartella (con nome pari allo username scelto) dove avranno luogo tutte le operazioni richieste da uno stesso utente. E’ evidente che non è ammessa la presenza, nella stessa directory, di file con nome uguale. Pertanto nel caso in cui l’utente invii un file con nome pari al nome di un file già presente sul server, quest’ultimo verrà sovrascritto.
Per quanto riguarda la funzione di visualizzazione dell’elenco dei file caricati, una volta che il server riceve tale direttiva esegue il comando “ls” sulla cartella dell’utente che ha fatto la richiesta, e salva il suo output su un file. Subito dopo il server legge il file e invia il contenuto cifrato al client, il quale provvederà a decifrare il messaggio e a stamparlo a schermo.
Se viene richiamata la funzione di rimozione di un file, il server prima controlla che tale file sia presente, in caso positivo provvede a rimuoverlo richiamando il comando “rm”. In caso negativo provvede a notificare l’assenza del file al client. Nel caso di rimozione di tutti i file, il server rimuove completamente la directory dell’utente, e ne ricrea una nuova vuota. Il trasferimento di un file avviene per mezzo di buffer di dimensioni limitate e fisse, che viene ripetutamente riempito e svuotato, in modo che sia possibile trasferire file di qualsiasi dimensione senza eccedere nell’allocazione di memoria dinamica.

Come detto in precedenza, si suppone che i client conoscano la chiave pubblica del server. Per realizzare tale ipotesi, vista l’assenza di certificati, si potrebbe fare in modo che il client venga distribuito assieme al file contenente la chiave pubblica del server. Chi distribuirà il software dovrà fare un hash del pacchetto, e renderlo noto, in modo che il client possa verificare l’integrità della chiave pubblica e dell’eseguibile, e che quindi possa essere certo di collegarsi al server autentico.
Il server dovrà essere eseguito da un’entità sicura, che protegga il file dove sono memorizzate le credenziali dei client da accessi non autorizzati.