Il Protocollo Handshake di TLS (fino alla versione 1.2)

Basato sulla presentazione "TLS Handshake Protocol"

Indice

1 Il Protocollo Handshake di TLS (fino a v1.2)

Questa analisi si concentra sul protocollo TLS (Transport Layer Security) fino alla versione 1.2. È importante notare che **TLSv1.3 è significativamente diverso** e semplifica molti di questi passaggi.

1.1 Quando avviene l'Handshake?

L'handshake è il processo fondamentale che avviene in diverse situazioni:

- **Negoziazione iniziale**: Quando un client si connette per la prima volta a un server, usano l'handshake per:
  - Autenticarsi reciprocamente (o più comunemente, solo il server viene autenticato).
  - Concordare gli algoritmi crittografici da utilizzare (la "cipher suite").
  - Scambiare valori casuali (nonces) per prevenire attacchi di tipo "replay".
  - Scambiare o calcolare segreti condivisi (le future chiavi di sessione).
- **Ripresa della sessione (Session Resumption)**: Se client e server hanno già comunicato, possono usare un handshake abbreviato per riprendere una sessione precedente senza rieseguire l'intero processo di autenticazione e scambio chiavi.
- **Re-keying**: Per "rinfrescare" i segreti (le chiavi di sessione) dopo un certo periodo di tempo o quantità di dati scambiati, pur mantenendo la stessa sessione.

1.2 Obiettivi dell'Handshake

Gli obiettivi primari dell'handshake sono:

1. **Negoziazione sicura dei segreti condivisi**: Le chiavi di sessione simmetriche non vengono *mai* trasmesse in chiaro. Vengono derivate utilizzando la crittografia asimmetrica durante l'handshake.
2. **Autenticazione (opzionale)**: L'handshake può autenticare sia il client che il server. In pratica, **l'autenticazione del server è quasi sempre richiesta** (il client deve sapere con certezza con chi sta parlando). L'autenticazione del client è più rara (spesso gestita a livello applicativo, es. con login e password).
3. **Robustezza agli attacchi Man-in-the-Middle (MITM)**: L'intero processo è progettato per resistere a un utente malintenzionato che intercetta e modifica la comunicazione.
4. **Negoziazione affidabile**: Un attaccante non può manomettere (tamper) la comunicazione per alterare l'esito della negoziazione (ad esempio, forzando l'uso di algoritmi più deboli) senza essere scoperto.

Un punto fondamentale è che la negoziazione è intrinsecamente **vulnerabile ad attacchi di downgrade**, che sono la minaccia principale che l'handshake deve contrastare.

2 Flusso dei Messaggi dell'Handshake (Schema Completo)

Questo è il flusso completo dei messaggi scambiati tra Client e Server durante un handshake completo (non abbreviato).

Client → Server Client Hello (Propone versioni TLS, cipher suite, nonce del client)

Server → Client Server Hello (Sceglie la versione e la cipher suite, invia il nonce del server)

Server → Client Certificate* (Invia il suo certificato digitale, contenente la chiave pubblica)

Server → Client Server Key Exchange* (Messaggio opzionale, usato per algoritmi come Diffie-Hellman)

Server → Client Certificate Request* (Opzionale: il server chiede al client di autenticarsi)

Server → Client Server Hello Done (Il server ha finito la sua parte iniziale)

Client → Server Certificate* (Opzionale: il client invia il suo certificato, se richiesto)

Client → Server Client Key Exchange (Il client invia il "pre-master secret" (cifrato) o i suoi parametri DH)

Client → Server Certificate Verify* (Opzionale: se il client ha inviato un certificato, firma i messaggi)

Client → Server Change Cipher Spec (Avviso: "da ora in poi cifro tutto")

Client → Server Finished (*Primo messaggio cifrato*: un MAC di tutti i messaggi precedenti)

Server → Client Change Cipher Spec (Avviso: "anch'io da ora in poi cifro tutto")

Server → Client Finished (*Primo messaggio cifrato dal server*: un MAC di tutti i messaggi)

Entrambi Application Data (La comunicazione sicura (es. HTTP) inizia)

(= Messaggi opzionali e/o contestuali)*

2.1 Nota sulla Segmentazione TCP

È importante ricordare che TLS opera sopra TCP. Un singolo *TLS Record* (che può contenere uno o più messaggi Handshake) può essere frammentato in più *segmenti TCP* per la trasmissione. Non c'è una corrispondenza 1:1.

3 Fase 1: Negoziazione Iniziale (Hello)

Questa fase stabilisce i parametri di base della connessione.

3.1 Client Hello

È il primo messaggio, inviato in **plaintext**. Contiene:

- **Versione**: La versione TLS/SSL più alta supportata dal client.
- **Random (32 byte)**: Un nonce composto da 4 byte di timestamp e 28 byte casuali.
- **Session ID**: Se il client desidera riprendere una sessione precedente, include qui il vecchio ID. Altrimenti è vuoto.
- **Cipher Suites**: Una **lista** di tutte le combinazioni di algoritmi crittografici supportate dal client, in ordine di preferenza.
- **Algoritmo di Compressione**: Oggi sempre nullo (la compressione TLS si è rivelata vulnerabile ad attacchi come CRIME).

3.2 Cipher Suite

Una "cipher suite" è una stringa che definisce un set di algoritmi. La nomenclatura standard (fino a TLS 1.2) è: `TLS_AAAAAAA_WITH_BBBBBBBB_CCCCCCCC`

- **AAAAAAA** (Algoritmo di Scambio Chiave): Definisce come il server si autentica e come viene scambiata la chiave (es. `RSA`, `DHE_DSS`).
- **BBBBBBBB** (Algoritmo Simmetrico): L'algoritmo di cifratura per i dati (es. `3DES_EDE_CBC`, `AES_128_GCM`).
- **CCCCCCCC** (Algoritmo Hash): Usato per il Message Authentication Code (MAC) (es. `SHA`, `SHA256`).

3.3 Server Hello

In risposta, anch'esso in **plaintext**. Contiene:

- **Versione**: La versione **più alta supportata da entrambi**. (Se il Client offre 1.2 e il Server supporta solo 1.1, il Server risponderà 1.1).
- **Random (32 byte)**: Un *nuovo* valore casuale generato dal server, diverso da quello del client.
- **Session ID**: Se il server accetta la ripresa della sessione, conferma il Session ID, altrimenti ne genera uno nuovo.
- **Cipher Suite**: **Una singola** cipher suite, scelta dal server dalla lista offerta dal client.
- **Algoritmo di Compressione**: Selezionato dalla lista del client (oggi, nullo).

4 Interludio: Uso della Crittografia Asimmetrica

La crittografia simmetrica (dove la stessa chiave cifra e decifra) è molto veloce, ma ha un problema: come si fa a scambiare la chiave in modo sicuro? Includerla nel codice di un'app sarebbe un disastro di sicurezza.

La soluzione è la **crittografia asimmetrica** (o a chiave pubblica/privata).

- Si usa una coppia di chiavi: una **Chiave Pubblica** (K_{ENC}) e una **Chiave Privata** (K_{DEC}).
- K_{ENC} può essere distribuita a chiunque. K_{DEC} deve rimanere segreta.
- I dati cifrati con la Chiave Pubblica possono essere decifrati *solo* con la Chiave Privata corrispondente.
- È **computazionalmente impossibile** risalire alla Chiave Privata conoscendo solo la Chiave Pubblica.

La crittografia asimmetrica è però molto lenta (4-5 ordini di grandezza più lenta della simmetrica).

Soluzione Ibrida (il metodo di TLS): Usare la lenta crittografia **asimmetrica** *solo* per scambiare o concordare una chiave simmetrica (chiamata "master secret"). Successivamente, usare la veloce crittografia **simmetrica** (con la chiave appena concordata) per cifrare tutti i dati della comunicazione.

4.1 Approcci allo Scambio Chiavi (fino a TLS 1.2)

1. Trasporto della Chiave (Key Transport) - es. RSA

1. Il Server invia la sua Chiave Pubblica (nel suo certificato).
2. Il Client genera un segreto casuale (il "pre-master secret").
3. Il Client *cifra* il pre-master secret con la Chiave Pubblica del server e glielo invia.
4. Solo il Server (con la sua Chiave Privata) può decifrare il messaggio e ottenere il pre-master secret.
5. Ora entrambi possiedono lo stesso segreto.

2. Accordo sulla Chiave (Key Agreement) - es. Diffie-Hellman (DH)

1. Client e Server generano entrambi una coppia di chiavi DH (privata e pubblica).

2. Si scambiano *solo* le loro chiavi pubbliche DH (es. $g^x \mod p$ e $g^y \mod p$).
3. Combinando la propria chiave privata con la chiave pubblica ricevuta, entrambi possono *calcolare indipendentemente* lo stesso identico segreto condiviso ($g^{xy} \mod p$).
4. Il segreto non viene mai trasmesso, nemmeno cifrato.

5 Attacchi di Downgrade

Questa è una vulnerabilità critica che l'handshake deve prevenire.

- Un attaccante Man-in-the-Middle (MITM) intercetta il `Client Hello` (che, ad esempio, propone "TLS 1.2" e cipher suite robuste).
- L'attaccante modifica il messaggio prima di inoltrarlo al server, sostituendo la versione con "SSL 2.0" (una versione vecchia e rotta) e rimuovendo tutte le cipher suite robuste.
- Il Server riceve il `Client Hello` manomesso e crede che il client supporti solo SSL 2.0.
- Il Server risponde con un `Server Hello` che accetta SSL 2.0.
- L'attaccante inoltra questa risposta al client. Client e Server sono stati ingannati e negoziano una connessione debole che l'attaccante può rompere.

6 Fase 2 e 3: Autenticazione e Scambio Chiave

Questa è la fase centrale dell'handshake, che utilizza la crittografia asimmetrica e previene gli attacchi di downgrade.

6.1 Fase 2: Il Server si Autentica

- **Certificate**: Il server invia il suo certificato X.509 (o una catena di certificati). Questo certificato lega l'identità del server (es. `www.google.com`) alla sua Chiave Pubblica ed è firmato da un'Autorità di Certificazione (CA) fidata.
- **Server Key Exchange** (Opzionale): Usato se la chiave nel certificato non è adatta al trasporto (es. è solo per firmare). È fondamentale per **Ephemeral Diffie-Hellman (DHE)**. In DHE, il server genera una chiave DH *temporanea* (effimera) solo per questa sessione e la invia *firmata* con la sua chiave privata (RSA o DSS) del certificato.
- **Certificate Request** (Opzionale): Il server chiede al client di inviare un certificato per autenticarsi.
- **Server Hello Done**: Un messaggio vuoto che segnala la fine dei messaggi del server.

6.2 Fase 3: Il Client Risponde

- **Client Key Exchange**: Questo è il cuore dello scambio.
  - **Se si usa RSA**: Il client genera il *pre-master secret*, lo cifra con la Chiave Pubblica del server (presa dal certificato) e lo invia.
  - **Se si usa DH**: Il client invia qui la sua chiave pubblica DH (es. $g^y \mod p$).
- **Certificate Verify** (Opzionale): Se il server ha richiesto l'autenticazione del client (con `Certificate Request`), il client usa la sua chiave privata per *firmare* un hash di tutti i messaggi dell'handshake scambiati fino a quel momento. Questo prova che il client *possiede* la chiave privata associata al certificato che ha inviato.

6.3 Come si previene il Downgrade (nel caso RSA)?

Quando il client invia il `Client Key Exchange`, il *pre-master secret* che cifra include i primi due byte che indicano la **versione TLS nativa del client** (es. "3.3" per TLS 1.2). L'attaccante MITM non può leggere o modificare questo contenuto, perché è cifrato con la chiave pubblica del server. Quando il server riceve il `Client Key Exchange` e lo decifra, confronta la versione *interna* (es. TLS 1.2) con quella negoziata nell'`Server Hello` (es. SSL 2.0). Se non corrispondono, rileva l'attacco di downgrade e interrompe la connessione.

7 Fase 4: Conclusione e Attivazione Cifrari

Questa fase ha due scopi fondamentali:

1. Attivare i nuovi parametri di sicurezza negoziati.
2. Verificare che l'handshake non sia stato manomesso da un MITM.

7.1 Change Cipher Spec (CCS)

Questo è un messaggio speciale, che tecnicamente appartiene a un protocollo separato. È un singolo byte (con valore 0x01) che notifica all'altra parte: "Tutto ciò che invierò da questo momento in poi sarà cifrato e autenticato con le chiavi che abbiamo appena generato".

7.2 Finished

Questo è il **primo messaggio inviato con la nuova cifratura**. Contiene un MAC (calcolato usando una PRF) di *tutti i messaggi dell'handshake* scambiati fino a quel punto (da `Client Hello` in poi).

- Il client invia il suo `Finished` (cifrato).
- Il server invia il suo `Finished` (cifrato).

Se un attaccante avesse modificato anche un solo bit di un messaggio precedente (come la lista delle cipher suite nel `Client Hello`), l'hash calcolato dal client e quello calcolato dal server sarebbero diversi. Il server non riuscirebbe a verificare il `Finished` del client (o viceversa), e l'handshake fallirebbe.

Questo messaggio **autentica l'intero handshake** e conferma che entrambe le parti hanno calcolato gli stessi segreti.

8 Gerarchia e Calcolo delle Chiavi

TLS utilizza una gerarchia di chiavi per garantire che, anche se un set di chiavi di sessione viene compromesso, il segreto principale rimane al sicuro.

1. **Pre-Master Secret**: Il segreto iniziale scambiato (via RSA) o calcolato (via DH).
2. **Master Secret**: Un segreto fisso di 48 byte, derivato dal *Pre-Master Secret* e dai *nonce* del client e del server.
3. **Key Block (Connection State Keys)**: Il *Master Secret* (che non viene mai usato direttamente) e i nonce vengono usati come input per una **Pseudo Random Function (PRF)** per "espandere" il segreto e generare tutte le chiavi di sessione necessarie (fino a 6):
    - Chiave MAC del Client
    - Chiave MAC del Server
    - Chiave di Cifratura del Client
    - Chiave di Cifratura del Server
    - IV (Initialization Vector) del Client (per cifrari a blocchi)
    - IV (Initialization Vector) del Server

Questo processo è noto come paradigma **Extract-then-Expand**.

8.1 Handshake Abbreviato (Session Resumption)

Se un client si riconnette e invia un `Session ID` valido (Fase 1), il server può accettarlo.

- Il server recupera il **Master Secret** originale associato a quel Session ID.
- **Si saltano le Fasi 2 e 3**: non c'è scambio di certificati né crittografia asimmetrica.
- Si scambiano **nuovi nonce** (nel `Client Hello` e `Server Hello`).
- Si usano il *vecchio Master Secret* e i *nuovi nonce* per generare un **nuovo set di chiavi di sessione**.
- Si passa direttamente alla Fase 4 (`Change Cipher Spec` e `Finished`).

Questo è molto più veloce, ma non offre *Perfect Forward Secrecy (PFS)* (segretezza futura), poiché se un attaccante ruba il Master Secret (o la chiave privata del server in caso di RSA), può decifrare tutte le sessioni (passate e future) basate su di esso. Gli algoritmi effimeri (DHE) prevengono questo.

8.2 Le Funzioni Pseudo-Casuali (PRF) in TLS

La PRF è il "motore" che genera il materiale crittografico.

- **TLS 1.0/1.1**: Usava una PRF *hard-coded* che combinava MD5 e SHA-1 (ormai deboli). L'idea era che un attaccante dovesse rompere entrambi gli hash.
- **TLS 1.2**: Ha introdotto una PRF *negoziabile*, permettendo l'uso di hash più robusti. L'impostazione predefinita è basata su SHA-256.
- **TLS 1.3**: Sostituisce la PRF con **HKDF** (HMAC-based Key Derivation Function), un moderno standard per la derivazione di chiavi, provabilmente sicuro.