Una Introduzione a TLSv1.0

In questa rubrica abbiamo già avuto modo di descrivere protocolli ed applicazioni che sfruttano SSL/TLS (Secure Socket Layer / Transport Layer Security). Vista l'importanza attuale nelle comunicazioni sia in internet che in reti private di questo protocollo, riteniamo utile esaminarlo un poco più in profondità, studiare i suoi punti di forza e le debolezze, le applicazioni più comuni ed i principali problemi di sicurezza. Ovviamente contiamo di affrontare queste tematiche in questo e nei successivi numeri della rivista.

TLSv1.0

Per comprendere meglio gli aspetti di sicurezza è necessario partire da una descrizione almeno sommaria del protocollo, e per questo consideriamo TLSv1.0 descritto nel RFC 2246 del 1999. Questa è l'ultima versione del protocollo, con il numero di revisione 3.1, mentre le precedenti sono SSLv2.0 e SSLv3.0 che, se possibile, non dovrebbero più essere usate principalmente per delle debolezze dal punto di vista della sicurezza; in particolare la versione SSLv2.0 è fortemente sconsigliata.

Il protocollo SSL è stato inizialmente proposto da Netscape Communications per rendere sicure le comunicazioni tra un server ed un browser web. SSL/TLS è però un protocollo generico e può essere applicato a qualunque applicazione che abbia caratteristiche simili dal punto di vista tecnico alla navigazione web. In particolare SSL/TLS è stato costruito assumendo di avere:

1. un client che si connette ad un server
2. il client che vuole autenticare il server, ovvero essere sicuro dell'identità del server
3. opzionalmente il server può richiedere l'autenticazione del client
4. la creazione di un canale cifrato di comunicazione tra il client ed il server.

Inoltre il protocollo è stato inizialmente pensato per:

1. il trasferimento di pochi dati, ovvero brevi sessioni cifrate
2. l'utilizzo di certificati digitali per l'autenticazione
3. essere facilmente implementabile come add-on alle applicazioni.

Questo ultimo punto implica che, al contrario di IPSEC[1], per implementare SSL/TLS non è necessario modificare lo stack TCP/IP del Sistema Operativo, ma solo aggiungere delle nuove funzionalità alle applicazioni. Oggi è possibile aggiungere facilmente SSL/TLS ad una applicazione utilizzando librerie specifiche quali ad esempio *openssl*.

In pratica nello schema ISO/OSI, SSL/TLS si posiziona al di sopra dello stack TCP/IP come nel

1 Per una descrizione di IPSEC si vedano i numeri 16 e 17 di ICT Security, Ottobre/Novembre 2003.

Diagramma 1.[2]

<table>
<tr><th>Layer</th><th colspan="8">Protocols</th></tr>
<tr><td rowspan="2">Application</td><td>TLS Hand-shake</td><td>TLS Change Cipher</td><td>TLS Alert</td><td>HTTPS</td><td>LDAPS</td><td rowspan="2">...</td><td rowspan="2">HTTP</td><td rowspan="2">SMTP</td></tr>
<tr><td colspan="5">TLS RECORD LAYER</td></tr>
<tr><td>Transport</td><td colspan="8">TCP</td></tr>
<tr><td>Network</td><td colspan="8">IP</td></tr>
<tr><td>Data Link</td><td colspan="8">(Ethernet ...)</td></tr>
</table>

Diagramma 1. SSL/TLS nella pila ISO/OSI

SSL/TLS usa TCP come protocollo di trasporto, delegando ad esso tutte le problematiche di affidabilità del trasporto. Questo implica anche che SSL/TLS può essere utilizzato praticamente solo per applicazioni che utilizzano TCP come protocollo di trasporto dei dati. Come indicato nel diagramma, TLS è composto da 4 sotto-protocolli

1. Record Layer: questo protocollo è alla base di TLS ed è incaricato di
    a) formattare+cifrare+inviare i dati in uscita verso TCP
    b) ricevere+decifrare+assemblare i dati in ingresso da TCP

   e ricevere/passare i dati sia ai tre protocolli superiori di TLS che all'applicazione finale
2. Handshake: questo protocollo è incaricato di effettuare lo scambio iniziale di dati che porta alla autenticazione delle parti ed alla creazione di un canale sicuro
3. Change Cipher: questo protocollo, molto semplice, permette di cambiare i *Security Parameters* usati dal Record Layer per de/cifrare i dati di una connessione
4. Alert: questo protocollo permette l'invio di messaggi SSL/TLS, prevalentemente d'errore, tra il server ed il client.

Diamo qui di seguito molto succintamente lo schema logico di una connessione SSL/TLS di un client (ad esempio un browser web) che vuole connettersi ad un server:

1. sul server vi è una applicazione che attiva SSL/TLS ponendo il server-Record-Layer in ascolto su di una porta TCP ed configurando il server-Handshake con i propri parametri di sicurezza
2. una applicazione client attiva il proprio Handshake con le informazioni necessarie per instaurare la nuova connessione, quali gli algoritmi crittografici che accetta, ma anche l'indirizzo del server eccetera

2 Più precisamente SSL/TLS si posiziona a livello 5 o di *sessione*.

3. il client-Handshake attiva il client-Record-Layer per creare una connessione (con il server) utilizzando il *NULL cipher*,[3] ovviamente all'inizio i dati non possono essere cifrati
4. il server-Record-Layer riceve i dati di una nuova connessione e li passa al server-Handshake il quale ha le informazioni necessarie, fornitegli dall'applicazione, per eseguire la fase di autenticazione
5. una volta terminato l'handshake, che descriveremo in dettaglio più avanti, viene attivato il Change-Cipher che istruisce il proprio ed il remoto Record-Layer a cambiare gli algoritmi crittografici ed utilizzare quelli appena concordati; le connessioni sono unidirezionali e cifrate con chiavi diverse, quindi sia il client che il server attivano il proprio Change-Cipher per la connessione che origina da ognuno di essi
6. tutti i dati successivi inviati dall'applicazione, sono trasmessi tramite il Record-Layer cifrati con gli algoritmi crittografici scelti; al termine della transazione l'applicazione attiva il Alert che invia al peer un messaggio di chiusura della connessione
7. anche in caso di errore viene attivato il Alert che invia al peer l'appropriato messaggio di errore
8. in qualunque momento l'applicazione, sia lato server che client, può riattivare l'Handshake per generare nuovi parametri crittografici per la connessione in atto.

Visto così, il protocollo è semplice, il che è sicuramente un punto a favore della sua sicurezza. Per capirlo meglio, dobbiamo però entrare un poco di più nei dettagli.

TLS Record-Layer

Cominciamo a considerare il formato in cui vengono trattati i dati dal Record-Layer il che ci permetterà di capire cosa deve fare l'Handshake. Quando il Record-Layer deve inviare dei dati, esegue le seguenti operazioni:

1. comprime i dati e li divide in blocchi detti *frammenti*
2. calcola l'HMAC[4] del frammento più il numero di sequenza del frammento stesso
3. cifra il frammento e l'HMAC
4. invia i dati cifrati al peer.

Per effettuare queste operazioni sono necessari i seguenti dati:

1. l'algoritmo di compressione (anche nullo)
2. l'algoritmo (simmetrico) di cifratura e la relativa chiave segreta
3. l'algoritmo di hash per HMAC e la relativa chiave segreta.

Le chiavi segrete, ad esempio del client, sono diverse per cifrare e decifrare, ovvero le chiavi sono

3 Il NULL Cipher indica algoritmi di cifratura nulla, i.e. nessuna cifratura, e senza chiavi segrete.
4 L'HMAC è definito nel RFC 2104 e descrive un modo di proteggere un Hash con una chiave segreta.

diverse a seconda della direzione dei dati. Si noti che i dati sono cifrati con algoritmi simmetrici, quindi la chiave usata dal client per cifrare, è usata dal server per decifrare, e viceversa. La presenza dell'HMAC aumenta la sicurezza dei dati cifrati, prevenendo ad esempio attacchi di replay e simili oltre a garantire autenticità e integrità.

L'ultimo punto di interesse è vedere come vengono generate le chiavi segrete usate dal Record-Layer. Il Record-Layer riceve dall'Handshake principalmente 3 dati: una *pre-Master-key* e *due numeri casuali* generati rispettivamente uno dal client ed uno dal server. Combinando questi tre dati, sia il server che il client generano una *Master-key* dalla quale ottengono le varie chiavi segrete di cui necessitano.[5] La divisione fra pre-Master-key, generata dall'Handshake, e Master-key generata dal Record-Layer, rende l'architettura totalmente modulare ed il Record-Layer indipendente dagli algoritmi utilizzati per l'autenticazione e la generazione della pre-Master-key.

TLS Handshake

Come abbiamo visto, l'Handshake deve fornire al Record-Layer i seguenti dati:

1. gli algoritmi di compressione, cifratura e di Hash per HMAC
2. la *pre-Master-key* e due numeri casuali scelti uno dal client ed uno dal server.

Essendo il TLS Handshake la parte più nota del protocollo, ne diamo qui una descrizione succinta. I dati richiesti sono generati con l'invio di 4 messaggi tra il client ed il server:

1. CLT => SRV - *Client Hello*: questo messaggio contiene un numero casuale, un SessionID e la lista degli algoritmi di compressione, degli algoritmi crittografici e di Hash sia per l'autenticazione che per il canale cifrato, che il client accetta
2. SRV => CLT - *Server Hello*: questo messaggio contiene diversi dati a seconda degli algoritmi scelti; in particolare il server invia un proprio numero casuale, la lista degli algoritmi che ha scelto all'interno della proposta del client, il proprio certificato digitale,[6] opzionalmente, a seconda degli algoritmi crittografici scelti, dei parametri necessari a generare la pre-Master-key, ed opzionalmente la richiesta del certificato del client; alcuni dei dati, anche se non cifrati, sono firmati digitalmente dal server usando la chiave privata corrispondente alla chiave pubblica presente nel certificato inviato al client, in questo modo il client può accertarsi che il server possiede veramente la chiave privata legata al certificato inviato
3. CLT => SRV - *Client Key Exchange*: il client invia al server i parametri necessari a completare la creazione della pre-Master-key; se si adotta RSA, la pre-Master-key è scelta dal client ed inviata al server cifrata con la chiave pubblica del server presente nel certificato, se invece si utiliz-

5 Tecnicamente la Master-key agisce come una sorgente entropica per la generazione di chiavi casuali.
6 In realtà questo non è richiesto se si adotta la modalità *anonima* di TLS, che però non discutiamo in questa sede.

za Diffie-Hellman (Ephemeral)[7] il client invia solo il proprio valore pubblico; se richiesto il client invia anche il proprio certificato digitale; a questo punto il client esegue un Change Cipher istruendo sia il proprio che il server Record Layer ad utilizzare gli algoritmi e le chiavi di cifratura appena concordati per tutti i dati futuri; infine il client invia al server (ora in modo cifrato) un Hash costruito in modo particolare della Master-key e di tutti i dati scambiati sino a questo punto

4. SRV => CLT - *Server Finished*: ricevuti tutti i dati dal client, il server costruisce anch'esso la Master-key, verifica che l'Hash di controllo inviato dal client sia corretto, esegue il proprio Cipher Change, ed invia al client il proprio Hash di controllo costruito in modo simile (ma non identico).

Dopo questi 4 messaggi, il client verifica l'Hash di controllo del server e fatto ciò client e server possono scambiarsi i dati cifrati fra le applicazioni. Si noti come l'Hash di controllo finale sia fondamentale per la sicurezza: se qualcuno avesse modificato dei dati in uno dei 4 messaggi (ricordiamo che non sono cifrati) gli hash finali calcolati dal client e dal server sarebbero diversi, la connessione sarebbe chiusa con un errore e nessun dato sarebbe scambiato fra le applicazioni.

Aspetti di sicurezza del protocollo

Dopo aver brevemente riassunto il protocollo, consideriamo alcuni suoi punti di interesse dal punto di vista della sicurezza. Per prima cosa, se paragoniamo il TLS Handshake ad esempio con IKE, notiamo immediatamente la sua semplicità, che è sicuramente un bene, d'altra parte vedremo che la semplicità introduce delle limitazioni.

Ancora più semplice è il modo in cui sono protetti i dati delle applicazioni, di nuovo in paragone ad IPSEC, questa volta al protocollo ESP. In realtà, anche ad un paragone superficiale tra ESP e la modalità in cui TLS protegge i dati degli applicativi, risulta evidente come ESP abbia più livelli di protezione. Questo non vuol dire che TLS non è sicuro, al contrario, ma che soprattutto in caso di trasporto di molti dati o di tunnel cifrati di lunga durata, ESP offre maggiori garanzie anche in caso di debolezza di alcuni degli algoritmi crittografici utilizzati. E' ad esempio per questo motivo che OpenVPN ha adottato un protocollo simile ad ESP per cifrare e trasportare i dati al posto del modo nativo di TLS offerto dal Record-Layer.

Si noti inoltre come TLS preveda dei meccanismi per la rinegoziazione dei parametri crittografici, ma che questi non sono automatici come in IKE, ma lasciati alla decisione dell'applicazione che utilizza TLS.

Queste caratteristiche ovviamente corrispondono agli scopi iniziali di SSL, brevi connessioni per transito di pochi dati per transazioni web, ma che potrebbero mal adattarsi ad altri usi. Il punto che

7 Vedremo più avanti le particolarità di questo algoritmo.

vogliamo sottolineare qui è che TLS, come ogni protocollo crittografico, ha delle caratteristiche particolari che lo rendono ottimale per certe applicazioni ma sconsigliabile per altre. Pertanto, prima di adottare TLS od un altro protocollo crittografico, è sempre necessario comprenderne bene le caratteristiche e valutarne l'adeguatezza alla situazione concreta.

Tornando a considerare le caratteristiche di sicurezza di TLS, guardiamo ora la parte di maggior pregio del protocollo, ovvero il TLS Handshake. Se lo paragoniamo ad IKE, vediamo che mentre IKE prevede una fase non cifrata iniziale ed una fase cifrata utilizzata per la creazione delle chiavi segrete e del materiale crittografico, il TLS Handshake utilizza solo una fase non cifrata con in più lo scambio, questa volta cifrato, dell'Hash finale su tutti i dati scambiati nell'Handshake. Tra l'altro questo Hash finale è costruito utilizzando sia MD5 che SHA1, il che garantisce che anche in caso di debolezza di uno dei due algoritmi, il protocollo continui ad essere sicuro.[8] Mentre IKE *previene* gli attacchi *attivi* sulla creazione del materiale crittografico per il tunnel cifrando i dati scambiati, TLS *scopre* gli attacchi a posteriori verificando l'integrità dei dati. Gli attacchi a cui facciamo riferimento sono del tipo seguente. Supponiamo che un client indichi al server nel primo messaggio sia DES che 3DES quali algoritmi simmetrici per cifrare i dati, con preferenza per 3DES, e che l'attaccante intercetti il primo messaggio e lo modifichi cancellando 3DES; il server, ricevendo solo DES, potrebbe accettarlo, ed il client e server potrebbero allora accordarsi sull'uso di DES se non fosse per l'Hash finale che scoprirebbe la modifica del primo messaggio. Ovviamente è interesse dell'attaccante far scegliere a client e server un algoritmo di cifratura che è in grado di rompere!

Un altro punto che è stato molto discusso recentemente, è il modo in cui è generata la pre-Master-key. Questo, come abbiamo già indicato, dipende dagli algoritmi crittografici scelti per la fase di autenticazione. Riassumendo possiamo dire che vi sono 3 casi per la generazione della pre-Master-key:

1. *si adotta RSA*: il client genera un numero casuale e lo invia al server cifrato con la chiave pubblica del server estratta dal suo certificato digitale
2. *si adotta Diffie-Hellman*: i parametri pubblici di Diffie-Hellman (DH) del server sono estratti dal suo certificato digitale, per il client se esso invia il proprio certificato digitale i parametri sono estratti da questo, altrimenti il client genera al momento i parametri necessari e invia al server quelli pubblici
3. *si adotta Diffie-Hellman Ephemeral*: in questo caso sia il server che il client generano al momento i parametri di Diffie-Hellman indipendentemente dai propri certificati digitali ed inviano all'altro quelli pubblici.

8 Alla luce dei recenti risultati, agosto 2004, sulle debolezze di MD5, questo algoritmo dovrebbe presto essere sostituito. Purtroppo nel febbraio 2005 è stato mostrato che anche SHA-1 ha problemi simili, quindi presto bisognerà sostituirli entrambi.

Il primo modo ha due svantaggi, il primo è che la pre-Master-key è scelta unicamente dal client senza partecipazione del server; il secondo è che questo metodo permette facilmente l'intercettazione di tutto il traffico in caso di *key-escrow*.[9] In altre parole, se qualcuno fosse in possesso della chiave segreta del certificato digitale del server, questi può passivamente intercettare tutto il traffico, decifrare la pre-Master-key con la chiave privata, calcolarsi la Master-key e decifrare tutto il traffico dati. Quindi tutta la sicurezza rispetto ad attacchi passivi (sniffing) risiede unicamente nella segretezza della chiave privata del server.

Il secondo modo è simile al primo per quanto riguarda il *key-escrow* anche se viene adottato Diffie-Hellman. Ricordiamo che in questo caso la pre-Master-key è generata dal client/server utilizzando il proprio parametro segreto di DH ed il parametro pubblico dell'altro. In questo caso la pre-Master-key dipende dai parametri di entrambi e non può essere imposta da nessuno dei due. Come nel caso precedente, chi è a conoscenza del parametro segreto di DH del server può, intercettando i dati scambiati, calcolarsi la Master-key. Analogamente, se anche il client usa un certificato e qualcuno è a conoscenza del suo parametro segreto di DH, costui può ricostruire la Master-key dai dati scambiati.

Il terzo caso è sicuramente il più sicuro, sia il server che il client generano dei parametri di DH casuali ed unici per questa sessione. Pertanto nessun *key-escrow* è possibile in modo permanente, l'unica possibilità è avere accesso diretto al server, o al client, ed ottenere in tempo reale e per ogni sessione un parametro segreto di DH. In questo caso possiamo parlare di *Perfect Forward Secrecy* (PFS). Con questo termine si vuol dire che se un attaccante fosse in grado di ottenere in qualche modo i parametri crittografici di una sessione, incluse le chiavi, e quindi decifrare i dati, le informazioni ottenute non gli saranno d'aiuto per decifrare un'altra sessione, per la quale dovrà ripetere da capo tutto il lavoro. In altre parole, il possesso delle chiavi crittografiche di una sessione non dà alcuna informazione sulle chiavi crittografiche di un'altra sessione. Come è facile immaginare, questo modo ha come fattore negativo un maggiore impiego di risorse di calcolo rispetto agli altri due e quindi in alcune situazioni si tende a non usarlo ovviamente a scapito della sicurezza.

In futuri articoli torneremo sugli aspetti di sicurezza di SSL/TLS da un punto di vista più generale, forti della conoscenza delle basi del protocollo.

Andrea Pasquinucci
pasquinucci@ucci.it

9 Vi sono casi i cui si può essere obbligati per motivi legali o dalle forze dell'ordine a fornire ad altri la chiave privata del server; nel caso peggiore, la chiave privata del server può anche essere rubata.

Riferimenti Bibliografici

[1] SSLv3.0: http://wp.netscape.com/eng/ssl3/

[2] TLSv1.0 descritto in RFC-2246

[3] RSA: http://www.rsasecurity.com/rsalabs/ in particolare gli standard PKCS

[4] W.Diffie M.E.Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, IT-22:644-654, 1976, si veda anche PKCS#3

[5] openssl: http://www.openssl.org/