APPUNTI SICUREZZA

Sicurezza come problema:

- intrusion informatiche
- inserimento di virus
- incertezza sull'identità dell'utente
- Perdita di dati
- Blocco dei sistemi

Sicurezza come soluzione:

- Gestire database utenti
- Gestire parco macchine utenti
- Garanzia dei servizi
- Integrità di server e dati
- Garanzia di privacy

Centralità dell'utente autorizzato

Un servizio informatico deve:

- Essere disponibile (per utenti autorizzati)
- Non essere accessibile a chiunque

L'utente è la figura centrale, è centrale la procedura con cui l'utente si autorizza e il sistema lo riconosce (localmente o in rete), anche il mercato ha spostato la sua attenzione sull'utente autorizzato

Utente autorizzato → ha due componenti:

1) Un identificativo (nome utente) o anche altre informazioni anagrafiche, uno stesso utente fisico può avere più identificativi
2) Credenziali di accesso → permette di eseguire un operazione che può avere due esiti:
    1. L'utente accede al servizio
    2. L'utente non può accedere al servizio

Ci possono essere più credenziali per un utente

Controllo di accesso → verifica delle credenziali

Crea una barriera tra utente e servizio con lo scopo di:

- Evitare utilizzi non autorizzati
- Evitare usi illeciti di utenti autorizzati
- Permettere logging dell'utente

L'accesso dell'utente ha 3 fasi:

- Enrollment: il sistema acquisisce i dati utente e profilo
- Autenticazione: il sistema deve verificare che l'utente sia effettivamente chi sostiente di essere, l'identificativo fornito nella fase di controllo corrisponde all'utente che richiede il servizio
- Autorizzazione: una volta riconosciuto l'utente il sistema deve controllare che l'utente sia abilitato ad utilizzare il servizio

Autenticazione VS Identificazione

Autenticazione:

- L'utente dice chi è
- Il sistema verifica

Identificazione:

- Utente segue una procedura di autenticazione ma non fornisce un identificativo

Lo scopo è lo stesso: in ambi i casi il sistema concede o meno l'accesso. Modalità e procedure sono diverse

Ci sono 3 tipi di autenticazione:

1) Qualcosa che l'utente possiede → smartcard o chip
   Vantaggi: possibile autenticazione con solo token (es smartcard), impossibile fare replay, limita la manipolazione dell'hardware

Svantaggi: il token può essere perso o rubato, bisogna fisicamente fabbricare i token (costi)

2) Qualcosa che l'utente è → Biometria
   Vantaggi: non può essere rubato o (in teoria) copiato, difficile fare il reply
   Svantaggi: imprecisa, falsei negativi o falsi positivi
3) Qualcosa che l'utente sa → password
   Vantaggi: più diffusa, facile da implementare, disponibole su tutti i sistemi
   Svantaggi: soggetta a reply, sniffing, guessing, cracking, phishing

Protocolli di autenticazione

Client e server comunicano tramite protocollo di autenticazione che garantisce la sicurezza, l'utente è riconosciuto solo se le credenziali sono corrette, non ci dev'essere intercettazione delle credenziali, l'accesso procede in modo sicuro per tutta la sessione

Debolezze dei cifrari simmetrici:

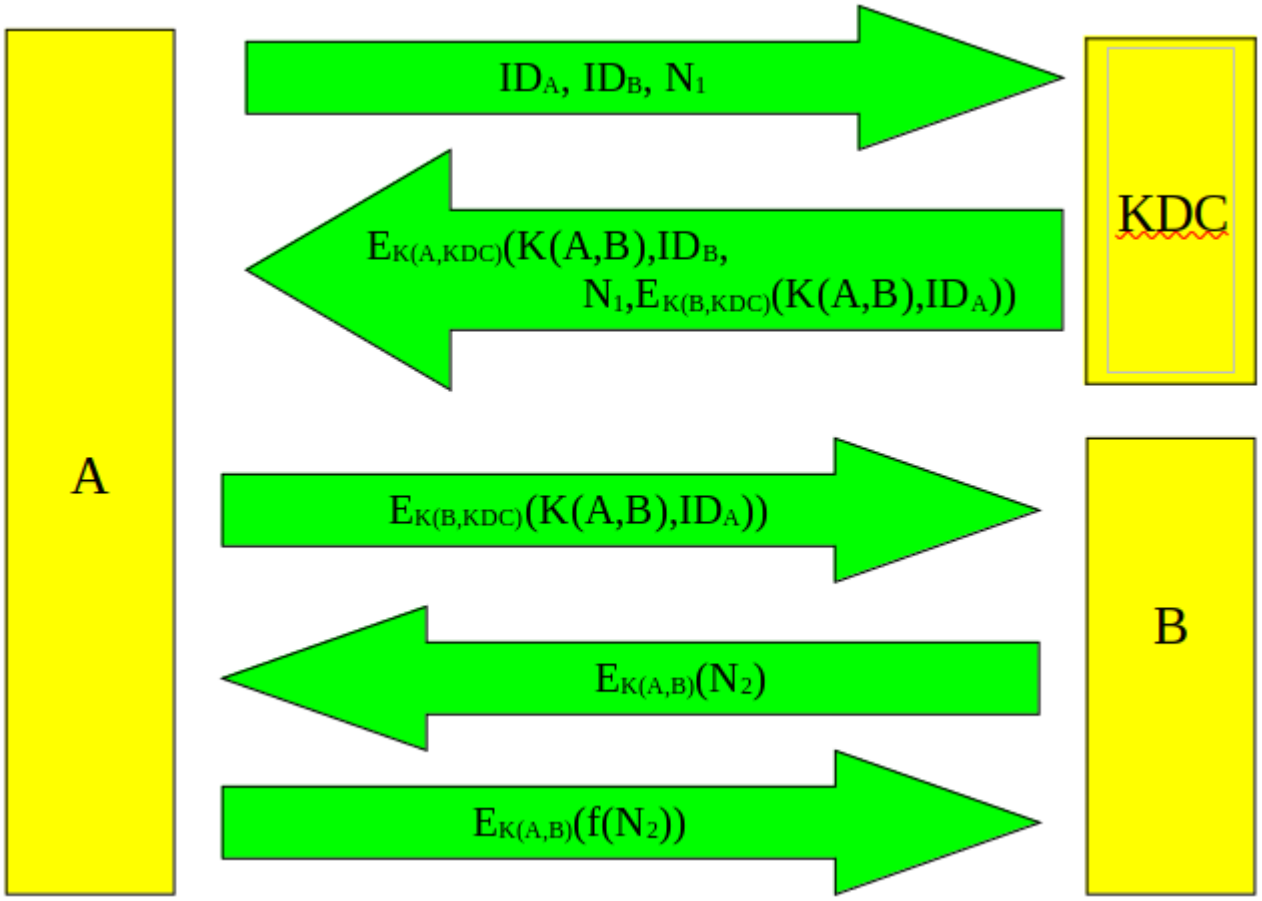
- Uso la conoscenza di parti di testo fisse
- Debole a critoanalisi statistica

Protocolli di autenticazione:

1) Password con validità limitata nel tempo
   Il sistema di autenticazione richiede delle credenziali, le credenziali hanno validità limitata nel tempo (es 3 mesi)
   Problema: possibile reply per il periodo di durata delle credenziali
2) Password generata da token come funzione di data e ora
   Il sistema genera un token in base a un segreto condiviso tra client e server e si unisce a data e ora, la password può cambiare ogni minuto
   Problema: il replay è comunque possibile in quella piccola finestra di validità della password
3) Autenticazione one time password
   Server e client condividono una password. La criptano n volte. Per ogni k sessione utilizzano la n-kesima password. Una volta terminate le n password si sceglie una altra password

0 = P → ultima password utilizzata
1 = H(P)
2 = H(H(P)) = H^2(P)
…
N = H^n(P) → prima password utilizzata

4) Needham/Schroeder
È un protocollo a chiave simmetrica, evita il replay perché la chiave è one timeevita il session hijacking perché non posso intercettare le chiavi. L'unico problema è distribuire le chiavi. Le distribuisce un Key Distribuition Center, che possiede una chiave simmetrica condivisa con ogni utente, uso queste chiavi per comunicare le chiavi utenti (session keys)



i. A comunica a KDC che vuole comunicare con B inviando il proprio ID (IDa), l'ID di B (IDb) e un nonce (N1)
ii. KDC genera casualmente la chiave condivisa tra a e b K(a,b) di 256 bit
iii. KDC invia un messagigo ad A, criptato con K(kdc,a), contenente la chiave K(a,b), l'IDb, il nonce (N1) u un ticket, criptato con K(kdc,b) contentente K(a,b) e l'Ida
iv. A ora conosce la chiave condivisa con B (K(a,b)) e inoltra a B il ticket (A il ticket non può leggerlo dato che è cifrato con K(kdc, b))

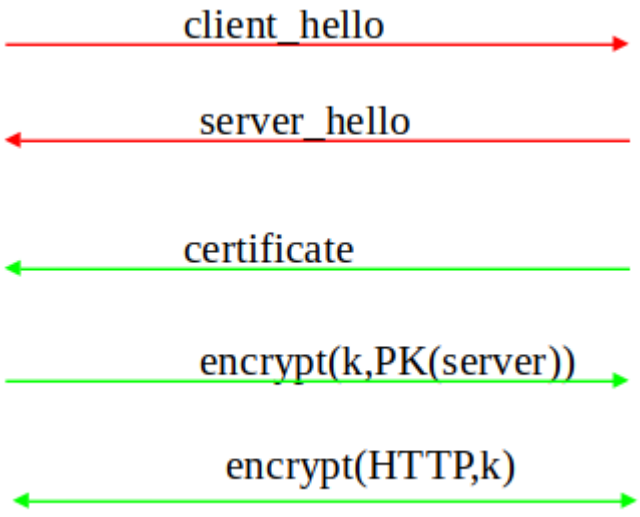
v. B apre il ticket e conosce K(a,b)

Ora A e b condividono un segreto, la sessione applicativa inizia, però prima A e B dimostrano di “essere vivi”

vi. B si inventa un N2 e lo cifra con K(a,b) e lo invia ad A
vii. A ottiene N2 decifrandolo e lo trasforma con una funzione f, lo cifra con K(a,b) e lo reinvia a B
viii. B capisce che A è viva e conosce la chiave

5) Kerberos
   È un controllo di accesso centralizzato basato su crittografia convenzionale. È usato da Microsoft Active Directory

6) SSL – secure socket layer



HTTP

È a livello applicativo ed è basato su tcp

Ha un livello di sicurezza intermedio, le informazioni sono passate in chiaro

Proxy

Oggetto in rete che so comporta da server verso il client e da client verso il server.

Funge da cache, se il client invia una richiesta e il proxy la possiede gli risponde, altrimenti la chiede al server e successivamente risponde. Questo permette di migliorare l'efficienza della connessione.

Funziona anche da firewall "mascherando" ciò che gli sta dietro

Un proxy lato server si chiama reverse proxy

Gateways

Oggetto intermediario tra client e server che però cambia il protocollo. Ad esempio comunico con il gateway tramite http e il gateway comunica con il server in maniera custom (utilizzando logica applicativa)

Tunnels

Funziona come un proxy ma senza memoria, inoltra semplicemente la richiesta.

In alcuni casi un proxy si può comportare da tunnel, se il server si accorge del proxy può impedire che venga memorizzato il contenuto (ad esempio nel caso di streaming live)

Formati HTTP

http ha diversi formati per risposta e richiesta, e per la richiesta di divide ulteriormente in GET e POST. La differenza tra GET e POST è che nel metodo GET i parametri son passati tramite la Uri, nel metodo POST i parametri sFono passati nel corpo della richiesta.

Ci sono diversi tipi di Header, che sono delle coppie <attributo, valore>:

1) General header: c'è sempre, contiene la data e l'ora della richiesta
2) Request header: sono solo nella richiesta e sono
    - If-modified-since : il client comunica al server che vuole la risposta solo se è "stata modificata da" la data indicata
    - Referer: indica la URI da cui l'utente proviene. Utile per tracciare l'utente, può essere usata per l'autenticazione.

- User-Agent: indica il software client utilizzato
- Autorization

3) Response header: sono solo nella risposta
    - Location: indica l'absolute-URI per la redirezione
    - Server: indica il software server del client
    - WWW-Authenticate
4) Entuity header: è presente solo se è presente l'entity body
    - Content-Encoding: codifica entity body
    - Content-Type: type/subtype dell'entity body
    - Content-Length: numero di bytes del Body
    - Expires: data da dopo la quale non si fa cacheing
    - Last-Modified: data dell'ultima modifica

Mime

Espresso come "tipo/sottotipo", è nato prima di http in ambiente mail anche per sapere il tipo degli allegati per poterli visualizzare

http eredita il mime

nella PEC si usa il Smime

Basic Authentication

È insicura se basata su http → i parametri vengono passati in chiaro

Richiede nome utente e password. Utilizza 4 messaggi se il browser è "fresco", 2 se è già stato utilizzto

Come funziona:

- Il server richiede l'autenticazione tramite:
    - Un Entity Body vuoto
    - Uno stato 401 (unauthorized)
    - Una risposta con l'header

      WWW-Authenticate: Basic realm="token"
- Il server richiede le credenziali
- Il client fornisce le credenziali utilizzando l'header e un basic-cookie

  Authorization: Basic basic-cookie

  ➔ È l'utente e la password codificati in base 64

- Il server riceve la richiesta con basic-cookie
- Se le credenziali combaciano allora il server invia la risposta richiesta in origine, se no il server risponde con l'errore 403 (forbidden) e l'entity body vuoto

Cookies
Sono una forma debole di autenticazione, servono per gestire le sessioni.
Ci sono due tipologie:
- Il server richiede il cookie per tutta la durata della sessione
- Il server richiede solo all'inizio e poi lo salva

Virtual Hosting
Sullo stesso server web sulla stessa macchina con stesso indirizzo IP possono girare più siti web. In base al nome simbolico presente nell'header della richiesta il server web fa matching con il nome del virtual server. Ogni virtual server ha un numero di porta a identificarlo e una diversa document root.
Per farlo funzionare basta includere nelle configurazione di apache il file di configurazione dei virtual host

HTTPS
È l'evoluzione sicura di HTTP, poiché il traffico tra server e client è cifrato.
Si usa SSL (attualmente tecnicamente è TLS, ma la dicitura ssl rimane) e necessitano di un certificato e una chiave pubblica almeno sul lato server (lato client non è necessaria)

Innanzitutto il server deve possedere un certificato firmato da una certification authority e una coppia di chiavi RSA (pubblica e privata)
Istruiamo apache dove trovare la chiave privata e il certificato, sia quello del server firmato sia quello della Certification Autority

Per creare questa serie di certificati utilizziamo OpenSSL

OpenSSL

Offre SSL, TSL, algoritmi di crittografia simmetrici e assimmetrici

Costituito da 3 parti:

- libssl.a : libreria che contiene l'implementazione e le funzioni utili al corretto funzionamento delle varie versioni SSLv2, SSLv3 TLSv1, sia lato client che lato server
- libcrypto.a: libreria che contiene:
    - funzioni crittografiche
    - funzioni di gestione dei certificati
    - cifrari simmetrici
    - cifrari assimmetrici
    - funzioni di hash
    - funzioni per creare, leggere, gestire certificati x509; funzioni per la gestione di una Certification autority
- openssl: tool da linea di comando che permette di usare la maggiorparte delle funzioni presenti in libcrypto.a

Public Key Infrastructure

La gestione delle chiavi e delle certificazioni richiede programmi specifici e una vera e propria infrastruttura

Perché utilizzare chiavi pubbliche?

- Necessarie per la firma elettronica (non per lo iam (Identity and Access Management))
- Nello iam la ragione è che sono più facili da distribuire
  Es. se abbiamo n soggetti con chiavi segrete avremmo bisogno di n^2 chiavi.
  Con le chiavi pubbliche ne abbiamo semplicemente 2n. Quindi per gestire tanti utenti è meglio avere chiavi pubbliche o Key Distribuition center

Perché utilizzare le chiavi pubbliche al posto del Key Distribuition Center?

- Il key distribuition center deve essere online e funzionante ogni volta che necessario
- Con la chiave pubblica il certificato vale sempre, la PKI può essere offline

Problema per la revoca dei certificati?

Distribuzione di chiave pubblica

Se A deve parlare con B, A deve inviare la chiave a B.

Problema: man in the middle → falsificazione della chiave

Soluzione:

- Certification Autority
    - A e B richiedono il proprio certificato
    - CA lo genera e gliero da
    - Se A e B vogliono parlare si scambiano i certificati senza che la CA intervenga

  Il certificato è compostoi dal nome del soggetto e la chiave pubblica, il tutto firmato dalla CA

- Public Key Directory → Directory chiavi pubbliche (terza parte fidata): raccolgo le chiavi e controllo l'autenticità. È un server (reso sicuro da HTTPS, SSL/TLS) a cui i client richiedono nome e chiave di chi vogliono contattare.
    - L'utente A fornisce la propria chiave pubblica a PKD tramite un canale sicuro
    - PKD memorizza la chiave associata ad A
    - Quando B vuole parlare con A, richiede la chiave pubblica a PDK
    - PKD fornisce chiave di A a B tramite un canale sicuro
- Public Key Authority: si comporta come KDC, a differenza della CA fornisce un database centralizzato con coppie <nome, chiave pubblica>. La correttezza della chiave è a carico della PKA
    - A fornisce la propria chiave pubblica a PKA
    - B vuole parlare con A: invia un messaggio con la richiesta della chiave pubblica di A e un nouns a PKA
    - PKA invia un messaggio a B contentente: la richiesta di B, il nons, la chiave pubblica di A, il tutto cifrato con la chiave privata di PKA

Chiave compromessa

Quando:

- Un soggetto non è più sicuro di essere l'unico a possedere il proprio certificato

Soluzione: Si associa al certificato un intervallo di validità

Scaduto l'intervallo l'utente può richiedere il rinnovo del certificato (non invia il certificato vecchio alla CA, ne richiede solo il rinnovo)

I certificati sono realizzati con lo standard .x509 (è supportato dal browser e prevede delle catene di certificati)

Struttura x.509:

- Nome soggetto
- Chiave pubblica del soggetto
- Firma della CA
- Intervallo Validità
- Altri campi di qualifica

Catena di certificati

Un certificato può essere certificato da un altro certificato

X.509 → mutua autenticazione client-server

Tecnologia assimmetrica, versione semplificata di SSL/TLS

1) A manda a B:
    - Il certificato di A
    - Il timestamp
    - Un rendom
    - Il nome di B
    - Dati
    - Chiave di sessione e chiave pubblica di B, il tutto cifrato
    - Firma di A
2) B manda ad A:
    - Il certificato di B
    - Il timestamp
    - Un random
    - Il nome di A
    - Il random precedentemente inviato da A
    - Dati
    - La chiave di sessione e la chiave pubblica di A, cifrati
    - La firma di B

3) A per dimostrare che è vivo invia:
    - Random precedentemente inviato da B e la firma di A

La sessione qua inizia, ogni messaggio è cifrato con la chiave di sessione

Revoca dei certificati

Può essere revocato il certificato prima della scadenza → debolezza dell'algoritmo, compromissione sicurezza chaive

Per mantenere tutto autormatico la CA espone una lista dei certificati revocati

Formato della lista dei certificati revocati (CRL)

È una lista dei certificati revocati, contiene i certificati e la loro data di revoca. È da scaricare.

3 problemi:

- Efficienza -> fare verifica della lista richiede tempo
- Online → se il cliente è offline la verifica non può avvenitre
- Aggiornamenti → tra un aggiornamento e l'altro il sistema è vulnerabile

Soluzione: OCSP

- Non scarico la lista
- Mando ID del certificato alla CA che risponde la validità
- Ho risposta immediata
- OCSP può essere configurato sul browser

PKI Architecture

È un sistema integrato di servizi e persone che forniscono servizi.

Al centro di tutto il sistema c'è la CA che emette i certificati, solo su pc controllati e sistemi adatti. I certificati vengono creati rispettando la norma tecnica e legale vigente nel paese. La CA deve avere un server OCSP sempre aggiornato.

RA – registration authority

La RA è l'intermediario tra utente e CA, l'utente va a chiedere alla RA il certificato, la RA identifica gli utenti. La RA non ha chiavi private della CA. l'utente scarica direttamente dalla CA il certificato. L'utente richiede alla RA la revoca o il rinnovo del certificato.

I software vendor si interfacciano direttamente con la CA e il loro software ha il ruolo di PKI.

Vulnerabilità PKI:

- Attacco al webserver della CA: come ogni altro attacco a un webserver. È complicato perché bisogna interrompere il servizio
- Attacco alla RA: più facile rispetto all'attacco alla CA. Si potrebbe compromettere il canale sicuro tra RA e CA e inviare richieste false di certificati. È anche il punto più esposto della PKI Architecture a errore umano
- Problemi con le revoche: Gli attacchi possono avvenire nel momento di aggiornamento delle liste, in quel momento non sono disponibili i certificati appena revocati da aggiornare per cui un utente malevolo potrebbe utilizzarli e farli funzionare. Viene risolto con OCSP.
- Errore da parte dell'utente: l'utente accetta certificati di dubbia origine

SSL & TLS

4 Handshake Protocol

L'obbiettivo principale è inizializzare la chiave di sessione che cifrerà tutto il traffico

1) *Fase 1*: Client_hello – server_hello
   Client_hello:
   - Versione
   - Random
   - Session id
   - Metodi di compressione disponibili
   - Cipher suite

   Server_hello:
   - Versione
   - Metodo compressione selezionato
   - Cipher suite

   Cipher suite → contiene:

   1) Cipher spec: algoritmi di encryption utilizzati
   2) Metodi di scambio delle chiavi

   I metodi di scambio delle chiavi sono:

   - RSA → encryption a chiave pubblica
   - Diffie-Hellman: versione base soggetta a man-in-the-middle
     - Anonymus → chiavi anonime e non atuenticazione
     - Ephimeral → chiavi pubbliche firmate con chiavi private RSA (autenticazione)
     - Fixed → le chiavi sono presenti nei certificati x.509 inviati dal client e dal server

   Server può richiedere la client_hello per rinegoziare le chiavi, lo fa tramite il messaggio hello_request

2) *Fase 2*: (fa tutto il server)
   il server invia la propria catena di certificati x.509. In base al metodi di scambio chiavi selezionato avviamo varie situazioni.
   - Fixed Diffie-Hellman: la chiave è presente nel certificato. Termina qui la fase di scambio chiavi

- Con RSA non si manda il successivo messaggio *server_key_excange*
- Negli altri casi si procede con lo scambio delle chiavi dopo essersi scambiati i certificati

Un server non anonimo può andare a richiedere i certificati al client tramite l'invio del messaggio *certificate_request*.
Il server chiude la fase due mandano il *messagio server_done.*

3) *Fase 3*: (fa tutto il client)
   Il client ha ricevuto il certificato lato server.
   Il client invia il proprio certificato se richiesto (dal server tramite il messaggio *certificate_request*)
   Il client fa *client_key_exchange* inviando al server il Master Secret cifrato con la chiave pubblica del server appena ottenuta (RSA, sta nel certificato x.509 lato server)
   Il client manda un messaggio *certificate_verify* contenente tutti i messaggi scambiati finora firmati con la chiave privata del client

4) *Fase 4*: (parte la sessione applicativa)

   il client invia il messaggio change_cipher_spec → un singolo byte, serve a segnalare il cambio di stato da pending a current

   il client mada il messaggio finished utilizzando il nuovo stato, questo messaggio contiene l'hash calcolato sul master secret e su tutti i messaggi scambiati fino a quel momento

   La stessa cosa è effettuata dal server

   A questo punto il 4HandShake è terminato

Key Generation Method

Durante il 4hs non si scambia il vero Master Secret ma il pre_master_secret da cui si deriverà poi il vero master secret. Il (vero) master secret verrà poi utilizzato per generare successive chiavi segrete

SSL Architecture

SSL si basa su TCP, ma è sotto il livello applicativo, realizza il livello di presentazione (Presentation).

È formato da tre protocolli più il record protocol.

Record Protocol:

Prende i dati dal livello applicativo (l'upper layer), li frammenta, li comprime, li firma con il MAC, i frammenti con il MAC vengono cifrati e viene aggiunto l'header, a questo punto sono pronti per essere inviati e viaggiano su TCP

Come viene creato il MAC: faccio l'hash della chiave del MAC concatenata con pad2 concatenato con l'hash della chiave del MAC concatenato con pad1 e il sequence_number, il tipo, la lunghezza e il numero di frammento

hash(MAC _write_secret |pad2 | hash(MAC _write_secret | pad1 | Seq_num | Type| Length| Fragment))

Type = tipo del livello superirore che ha consegnato i dati

Al termine della fase di handshake ho lo stato current e quindi posso utilizzare le chiavi appena concordate per cifrare. Durante l'handshake non le ho ancora concordate, quindi se client e server si erano già parlati utilizzo le chiavi presenti nell'attuale stato current (ovvero lo stato diventato current nella precedente comunicazione), in caso client e server non si siano mai parlati utilizzo le chiavi vuote e cifro con la funzione identità

1) *Handshake Protocol*: realizza l'autenticazione dei peer tramite la 4hs

2) *Change chiper spec Protocol*: serve al client per comunicare al server il cambio di stato da pending a current

3) *Alert Protocol*: sono messaggi per mandare errori, è un messaggio di 2Byte, un Byte è utilizzato per il livello di alert (fatale o non fatale), l'altro Byte è utilizzato per il codice di errore

Sessione e connessione in SSL

Sessione vs Connessione:

Sessione	Connessione
- È di livello applicativo - Ci sono due soggetti paritari - Necessita di uno schema di presentation - Eventuale cifratura dei dati - Una sessione può contenere più connessioni	- È a livello di trasporto - Non ha connotazione applicativa - Trasferisce solo bit tra client e server - Garantisce che i bit arrivino in ordine - Non sappiamo interpretare il significato dei bit scambiati - Garantisce affidabilità

Entrambe sono associate a uno stato:

Stato Sessione	Stato Connessione
- ID - Peer cetificate - Metodo di compressione - Algoritmi di crittografia utilizzati - Booleano (specifica se la sessione può o meno rimanere viva se apro altre sessioni) - Master secret	- Numeri random (nouns) - Vettore di inizializzazione utilizzato ne CBC encryption della cifratura nel record protocol - Secret number (numerazione pacchetti) - 2 chiavi, una per cifrare e una per certificare (quindi 4 in totale, 2 client e 2 server)

Directories & LDAP

Directory: insieme di record, chiamate anche entry ogni record avente una sua struttura. Ogni record è organizzato in una gerarchia. Ogni entry è un insieme di coppie <attributo, valore>. Il valore ha un certo tipo, ogni attributo può avere più valori

Deve essere possibile fare delle ricerche nella directory

Ogni pezzo di informazione è chiamato record o entry

Le directory sono ottimizzate per ricerca e lettura

Non è tanto importante la consistenza dei dati poiché i dati cambiano raramente

È un database gerarchico

Utilizzo: gestione di dipendenti o prodotti, stilare un inventario per una organizzazione

Si può accedere alle directory da:

- Linea di comando
- Interfaccia grafica tramite web application (interazione integrata con altro software)
- Lato server

Standard più noti:

- X500: complesso e in disuso
- LDAP: standard per le directory, è leggero, open, free

Directory → albero in cui le entry sono i nodi

A differenza dei database sono poco efficienti in scrittura ma molto più efficienti in lettura

Le informazioni sono presenti sia nelle foglie che nei nodi

I nodi che non sono foglie sono dei Container

Le informazioni possono essere distribuite su più server

Info presente in un nodo: <attributo, valore>

Ogni entry ha una Object Class che identifica quali elementi può contenere la entry

Naming Context: sottoalbero

Namespace → spazio dei nomi, evita conflitti

DNS si basa su LDAP

Client Operation

Un client deve poter interrogare un server LDAP

L'accesso avviene in 3 modi:

- Accedo tramite browser (phpldapadmin)
- Accedo tramite "gateway http", il client interagisce con il gateway (che è una pagina http) che risolve le domande interagendo con LDAP
- Client Custom

Il client per fare una ricerca deve specificare al server LDAP:

- Base DN: da che punto dell'albero si vuol far partitre la ricerca
- Scope
- Filtri: combinati tramite &, |, !, %, ecc.
- Parametri opzionali: tipo: attributo, dereferenziare aliases, ecc.

Ci sono parametri che posso impostare nel server

LDAP Protocol

Client:

1) Mi collego al sertver
2) Eseguo una ricerca (search)
3) Ricevo l'entry (una o più) in risposta alla search
4) Chiudo il collegamento

SI basa su TCP ma si può utilizzare anche UDP

I messaggi possono essere operazioni (più frequenti) o controllo o opzioni

Esempi di operazioni:

- Bind: indica a quali DN voglio collegarmi, devo fornire utente e password per accedere alla directory
- Serarch
- Compare
- Delete
- Add: serve per scrivere sulla directory

Schema → c'entra con la tipatura dei dati

È il sistema di tipatura e vincoli su dati della directory (come avviene nei database)

Quindi come sono fatte le entry, ovvero gli attributi

È complicato poiché ogni entry è diversa perché descrive qualcosa di diverso

Alcuni attributi sono obbligatori, altri no

Dobbiamo specificare come sono fatti gli attributi, ovvero la loro sintassi

Ogni volta che aggiungo dati nel server, LDAP controlla che lo schema sia rispettato

Lo schema è memorizzato nel file di configurazione del server oppure all'interno della stessa directory

Lo schema comprende:

- Structure rules
- Name form
- Sintassi

Ogni entry di un albero LDAP ha un attributo Object Class che definisce lo schema di entry

Esistono delle regole di struttura: come una Object Class si relaziona ad altre Object Class → esiste una gerarchia nelle Object Class (gerarchia di tipi)

Dobbiamo dire quali sono gli attributi o di che tipo sono i valori inseriti

Esistono delle matching rules, come si devono comportare i valori

Object Class

L'object class può essere utilizzata delle query

Le object class sono di 3 tipi:

- Astratte: non ereditano (?), servono come definizione astratta
- Standard: sono quelle più utilizzate, sono quelle gerarchiche
- Ausiliarie: classi esistenti in cui si possono aggiungere attributi

Dati delle object class:

- ID numerico
- nome
- Superclass
- Categoria (insieme delle 3 scritte sopra)
- Elenco degli attributi *must* e *may*
- Attributo che identifica la entry

Quando viene creata una entry nella directory dobbiamo associarla a una object class, uso quindi una object class esistente (o una esistente precedentemente inventata da me, è comunque meglio partire da una già esistente)

Management

Il managment riguarda due temi:

- Replica: backup
- Referral: implemento una base di dati LDAP distribuita e/o ridondante

Referral

redirect quando il client richiede una risorsa a un server in cui non è presente, l'informazione arriva quidi da un altro server

viene usato sui sottoalberi

ci sono 2 modi di gestire i referral:

- Demando tutto al server, il client è ignaro

- Il server risponde con l'indirizzo del server contenente il dato, questa informazione (il link) è contenuta nella entry referral sul server, il client allora rieseguirà la stessa richiesta ma al server del link

Possono essere interni ed esterni

Aliases

è simile al referrals ma è per una singola entry nella stessa directory (e non un intero sottoalbero su qualche altro server), deve essere risolto dal server

il client può chiedere al server di dereferenziare gli aliases, escludendoli quindi dalla ricerca

in un albero possono essere presenti sia aliases che referrals

Metadirectories

- Master directory: il client è configurato per fare domande a masterdirectory, la masterdirectory è collegata con il referral
- Sincronizzazione: sincronizzare un'architettura multi-master, il client fa domande alle due o più directory sincronizzate, generalmente l'operazione di sincronizzazione è gestihta da un harvester
- Harvester: Chiede continuamente dati a directory e scrive su directory per mantenere sincronizzazione
- Connector
- Gateway: il client fa domande al gateway (che è vuoto), il gateway interroga le directory e inoltra le risposte al client

Formati di Data Interchange

come si spostano grandi quantità di dati da e verso directory?

Standards: LDIF, DSML