Appunti di Internet Security

Giuseppe Criscione

Versione 1.0 25/06/2019

Indice

1 Perché è importante la sicurezza informatica

La sicurezza informatica è l'informatica. Non esiste un campo dell'informatica dove non è presente la sicurezza. Il motivo è semplice: creando servizi bisogna garantirne la sicurezza.

- **App sicure**. Bisogna avere fiducia delle app scaricate perché quello che si scarica può fare quello che vuole nel nostro sistema (es. trojan). Nasce il concetto di **permessi**. I sistemi operativi hanno una gerarchia di permessi o profili autorizzativi o tecniche per il controllo d'accesso. Forcing dei permessi giusti. E' fondamentale che l'app abbia due requisiti:

  1. L'utente capisca i permessi che sta dando (tipicamente no)
  2. I permessi siano definiti in maniera non minimalista (legge europea RGDP 679/2016 art. 25)
  3. Policy coerente ma lista dei permessi incoerente

  Inoltre, è importante per l'utente il **diritto all'accesso**. L'utente ha il diritto di chiedere al titolare del trattamento tutti i dati dell'utente di cui sono in possesso.

- **Browser sicuro**. L'applicativo di maggior diffusione è senza dubbio il browser. Esso gioca quindi un ruolo fondamentale nella sicurezza di un sistema. Ad esempio, i moderni browser gestiscono direttamente i certificati dei siti web, azione che in passato era svolta dal sistema operativo. Quindi i browser scendono più a basso livello.

- **Pagamenti sicuri**. Utilizzo di nuove tecnologie come la Blockchain. Il suo funzionamento si basa su due principi:

  1. Appendere alla catena è semplice
  2. Rimuovere è computazionalmente impossibile

2 Tassonomia degli attacchi

Vediamo quali sono le tipologie di attacchi che esistono:

- **Attacchi criminali**: frodi, appropriamento indebito di capitali, attacchi distruttivi come ad esempio rovinare un sistema, cancellare l'hard disk, negazione del servizio

- **Pirateria**: Furto di proprietà intellettuale, che si accresce verso il furto di identità (appropriazione indebita di identità) e furto di marchi.

- **Attacchi di tipo privacy**:Gli attacchi di tipo privacy, sono classificabili come: spionaggio industriale, attacchi a database su servizi online, analisi del traffico, spionaggio industriale su larga scala.

- **Monitoraggio del traffico**:Lo spionaggio esiste a tutti i livelli, sia industriale sia su vasta scala sia su tutto il traffico. Ma com'è possibile controllare tutto il traffico?
  Essendo la rete gerarchica, questo è possibile grazie ai servizi di routing, che gestiscono il traffico.
  Esistono programmi come `Echelon` che [..]

- **Attacchi a Database**: Oggigiorno i database sono ovunque. Dietro ogni singolo login, registrazione, ecc c'è una database. Oggi gli attacchi ai database sono molto in voga. La GDPR parla esplicitamente di pseudoanonimizzazione come primo strumento per garantire l'anonimato o il pseudoanonimato sui database. [ARX].

- **Attacchi a scopo pubblicitario**: Gli attacchi di negazione del servizio hanno un riscontro pubblicitario sul servizio, in questo caso in maniera negativa perché distrugge l'immagine e il servizio (in maniera temporanea).

- **Attacchi sui sistemi leali**: ad esempio il parere di un esperto in tribunale.

3 Proprietà di sicurezza

Mentre la tassonomia degli attacchi e degli attaccanti, è una tassonomia chiacchierata, la tassonomia delle proprietà è il nostro principale strumento di lavoro. Anche se le proprietà di sicurezza sono innumerevoli, è possibile definire quelle che sono le proprietà di base, ovvero: **segretezza (confidenzialità)**, **autenticazione** e **integrità**. Proprietà di base scritte pure sullo standard ISO 27001 dove si parla anche di **disponibilità del servizio**.
Avendo una base solida, è possibile, tramite un procedimento gerarchico definire delle proprietà sempre più specifiche. Al secondo livello, infatti, troviamo le proprietà di **non ripudio** e **disponibilità**. Continuando, al terzo livello parliamo di **valuta elettronica**, **equilibrio privatezza/legge**, **livelli di segretezza** e **controllo**.
In cima, troviamo le proprietà che possiamo definire noi.

3.1 Proprietà di livello 1

Segretezza (confidenzialità)

Definizione: *L'informazione non sia rilasciata ad entità non autorizzate a conoscerla.*
In genere, il segreto è qualcosa di personale, ma esistono segreti condivisi. Il segreto *partiziona* i partecipanti i due gruppi: quelli che lo possono sapere e quelli che non lo possono sapere. Questo è cruciale: non ha senso parlare di segretezza senza definire un insieme di autorizzati a conoscere. Il segreto non è assoluto, la confidenzialità è tra un certo gruppo di partecipanti rispetto ad un altro gruppo.
Due esempi di misure di segretezza: **crittografia** e **steganografia**.
Il watermarking è una tecnologia di alterazione di un'informazione (tipicamente un'immagine) in maniera tale che questa non sia percepibile dal fruitore ufficiale. Un'applicazione è la steganografia. Ad esempio, cambiando il bit meno significativo di un immagine (il bit più a destra) il risultato non è percepibile all'occhio umano. Ad esempio, se ho un'immagine a 4MP, per ciascuno di ogni pixel potrei cambiare il singolo bit meno significativo, avendo così a disposizione 4 milioni di bit. Posso così fare una codifica di un messaggio ed inserirla nell'immagine. Così facendo, l'immagine avrà modifiche impercettibili, ma starà trasportando delle informazioni.
Gli obiettivi del watermarking possono essere due:

- **Scenario 1**. Realizzare una comunicazione segreta tra due individui A e B. Usando un'immagine a 4MP ho 500.000 caratteri a disposizione. Quindi l'obiettivo in questo caso è la trasmissione di un messaggio segreto. Quindi in questo caso la proprietà di sicurezza necessaria è la **segretezza**.

- **Senario 2**. Inserimenti di un messaggio che esprima la proprietà intellettuale da parte del proprietario. In questo caso, la proprietà necessaria è la **integrità**.

Non abbiamo però una tecnologia scalabile di watermarking da garantire le proprietà di sicurezza dei due scenari dovuto proprio ai limiti di queste tecnologie. Si parla appunto di limiti sull'attuazione delle proprietà di sicurezza.
Negli ultimi tempi infatti il concetti di watermarking è meno in voga, a conferma di quello detto finora.
Un'altra tecnica di steganografia si chiama Chaffing&Winnowing, sviluppata da Ronald Linn Rivest (La R di RSA). La tecnica è usata nello scenario dove A e B comunicano, condividendo una chiave segreta. A dividerà il messaggio in bit inviandone uno alla volta in pacchetti che contengono anche un numero seriale necessario per poter ricostruire l'ordine del messaggio. Oltre a queste informazioni, viene inviato un codice di autenticazione del messaggio (MAC) crittografato tramite la chiave condivisa. Questi messaggi saranno inviati a C (fidato) tramite un canale sicuro. Quest'ultimo li invia a B tramite un canale non sicuro, mescolando i pacchetti di A con dati casuali con stesso numero seriale dei pacchetti di A, ma con i bit di contenuto invertiti ed un numero casuale al posto del MAC. Quando B riceverà i pacchetti, setaccerà i messaggi per scoprire se si tratta di informazioni rilevanti o meno. L'utente malevolo,se dovesse intercettare il flusso non potrebbe distinguere ammesso che non conosca la chiave.
La steganografia è l'arte di nascondere, mentre la crittografia è l'arte di trasformare in una direzione e ritrasformare nell'informazione d'origine.

Autenticazione

Definizione: *Le entità siano esattamente chi dichiarano di essere.*
Alcune misure possono essere la **conoscenza**, **possesso**, **biometria**.

Integrità

Definizione: *l'informazione non sia stata modificata da entità non autorizzate.*
È una proprietà di sicurezza in quanto ci può essere un attaccante che può trarre beneficio dall'alterazione. Alcune misure possono essere la **firma elettronica** e il **checksum**.
Il checksum è un controllo di integrità. Restituisce uno o più bit calcolati come funzione di un insieme di altri bit, cioè i dati che voglio trasportare. L'alterazione dei dati provocherà un fallimento del controllo del checksum. Perché due dati sorgenti differenti generano due checksum differenti. Il checksum però non garantisce l'integrità perché l'attaccante può sostituire il messaggio e checksum. In questo modo il destinatario non riscontrerebbe anomalie. Il checksum ha senso solo per superare i limiti della rete, ma non garantisce integrità. La firma digitale invece, non solo garantisce l'integrità, ma anche l'autenticità del documento.

Privatezza

Definizione: *Diritto di un'entità di rilasciare o meno i propri dati personali ad altre entità.*
La privacy troppo spesso viene scambiata con la sicurezza. La privacy è un diritto o un insieme di diritto. È un insieme di cose che l'individuo deve essere messo in condizione di fare. La confusione si genera con il concetto di riservatezza. La privacy è un diritto di segretezza. Ma qual è allora il nesso tra segretezza e privacy? La privacy è un diritto a queste cose, è una meta proprietà. La privacy è un insieme di diritti, in particolare un diritto alla riservatezza.

Anonimato

Definizione: *Diritto dell'iniziatore di una transazione di rilasciare o meno la propria identità ad altre entità.*
La privacy è circoscritta sulla sola identità personale. Se posso occultare la privacy della mia identità allora resto anonimo.
L'anonimato è un concetto assoluto, teorico. È come la randomicità. Un numero è random se la proprietà di scoprirlo tende a 0. Se non sappiamo la lunghezza del numero random (ad esempio di un numero random binario) dobbiamo enumerare ovvero provare tutte le combinazioni. La probabilità di trovare un numero di lunghezza n è $\frac{1}{2^n}$. La probabilità diminuisce in maniera esponenziale all'aumentare della lunghezza del numero. Significa che la probabilità di indovinare il segnreto **non è mai nulla**, se il segreto sia di lunghezza finita.
L'attaccante potrebbe non sapere la lunghezza del segreto da indovinare. In tal caso, la probabilità è ancora più bassa. Il messaggio è chiaro: il numero random non implica che l'attaccante non possa indovinarlo, vuol dire che l'attaccante ha una bassissima probabilità di indovinarlo.
Non esiste l'anonimato assoluto. Se faccio una certa transazione, avrò una tupla nel database che però è resa "anonima". Ma non esiste l'anonimato assoluto, esiste lo **pseudo-anonimato**. Lo pseudo-anonimato lo si fa creando istanze separate nei database. Un'istanza fisica è di fatto una copia fisica del database. Così come non esiste la randomicità assoluta non esiste l'anonimato assoluto.

La navigazione anonima non ci rende anonimi sulla rete, ma semplicemente rende anonimi nei confronti della macchina su cui stiamo operando perché non registra dati di sessione.
Esistono server anonimizzatori (Anonymous Web Proxy), che applicano una maschera. Questi server rimbalzano le connessioni, mettendoci la faccia. Oppure utilizzo delle VPN (Virtual Private Network), ovvero un tunnel di comunicazione sicuro con l'endpoint (segretezza e integrità). Un modo di fare VPN è IPsec o OpenVPN. La VPN ci rende anonimi perché quando chiediamo un sito web, quest'ultimo viene chiamato dall'IP della VPN. VPN però richiede un'autenticazione utente e utilizza una cifratura asimmetrica.
Un'altra misura di sicurezza della rete è la **segmentazione** operata attraverso l'uso di una VLAN, che oltre a risolvere il problema della saturazione degli indirizzi, funge da segmentazione.
Esistono servizi di anonimizzazione commerciali che si possono acquistare, ma tipicamente non sono altro che intermediari a livello applicazione. Se significa che forzare il server anonimizzatore rileverebbe la nostra identità.
Esistono soluzioni più sicure, che operano invece a livello di routing, come ad esempio **Tor**. Normalmente, nelle reti, quando si richiede un sito web, ci sarà una serie di nodi che verranno chiamati in causa per partecipare all'inoltro del traffico nel momento in cui il servizio richiesto inizia la comunicazione con la nostra macchina. Quindi c'è il problema del routing, ovvero nel minimizzare questi inoltri. Tor, invece, prevede che il routing sia forzato lungo un gran numero di nodi, in modo che aumenti la probabilità del chiamante (mittente) di rimanere anonimo. Perché chiunque intercetti il traffico in un qualunque hop della rete per risalire ad una certa identità non basta che rompa il singolo server anonimizzatore, ma deve essere in grado

di trovare tutti gli hop a ritroso. Il che è difficile, ma non impossibile. [**Tor vulnerability**].

3.2 Proprietà di livello 2

Non ripudio

Definizione: *L'entità non possa negare la propria partecipazione ad una transazione con uno specifico ruolo.*
Il non ripudio serve nei casi in cui un'entità potrebbe negare la partecipazione. Spesso il non ripudio è confuso con l'autenticazione, ma ci sono delle differenze. Es: autenticare Mario Rossi vuol dire avere evidenza che il mio interlocutore dall'altra parte del sistema sia Mario Rossi. La parola *negare la partecipazione*, nella definizione di autenticazione non c'era perché non c'era il problema di negare un tentativo di negare. Potrei autenticare Mario Rossi, ma il fatto che lo autentichi per il riconoscimento della password, mi permette di impedirgli di negare la sua partecipazione. Quindi una cosa è riconoscere l'interlocutore (cioè autenticare), un'altra cosa è avere qualcosa (un'evidenza) che mi permetta di smascherare eventuali falsità da parte del mio interlocutore, ad esempio il non aver partecipato ad una transazione.
Il non ripudio, è una proprietà più forte, che senza autenticazione non avrebbe senso. L'autenticazione è una condizione necessaria ma non sufficiente. Se non ho autenticazione non ho il non ripudio.

Autenticazione, anonimato e non ripudio

Relazioni logiche:

1. L'autenticazione non fornisce l'anonimato
2. Autenticazione e anonimato sono una l'opposto dell'altra
3. Se ho l'autenticazione ho il non ripudio? **No**.
4. Se ho il non ripudio ho l'autenticazione? **Si**.
5. Se ho l'anonimato non posso avere il non ripudio. Con l'anonimato il non ripudio perde di significato in quanto se ho l'anonimato ho la non autenticazione; con la non autenticazione ho il non non ripudio.
6. Se ho il non ripudio, sicuramente il soggetto non è anonimo.

3.2.1 Il problema dell'esame (protocollo WATA)

I protocolli e gli schemi di sicurezza sono dappertutto, dal momento in cui ci possono essere intenzioni malevole. Anche un esame è un problema di sicurezza e quindi necessita di un protocollo di sicurezza. Una delle proprietà essenziali è l'autenticazione dello studente. Così come l'anonimato. L'anonimato non è dello studente nei confronti del docente, ma del suo compito in fase di correzione. Quindi l'anonimato del compito in fase di correzione è una proprietà importante. Quindi come realizzare un protocollo per un esame scritto tale che ciascun compito sia autenticato ma anonimo?
Tale protocollo deve garantire le seguenti proprietà:

- Autenticazione del compito per prevenire imbrogli dello studente (a tutela del docente)
- Anonimato del compito per prevenire votazione iniqua (a tutela dello studente)

La soluzione è il protocollo **WATA** (Written Authenticated Though Anonymous).
Il protocollo prevede l'estrazione di un numero n di domande su un database di N domande, generando un numero x di compiti diversi. Ogni domanda è posta in un singolo foglio con un codice a barre uguale che collega logicamente i tre fogli tra di loro. Nel primo foglio vi è una regione di spazio dedicata all'autenticazione, staccabile dal compito, detto token, con lo stesso codice a barre presente sugli altri compiti. Lo studente compila il token con la propria anagrafica, il l'invigilator passa ed autentica lo studente tramite un documento. Lo studente, prima di andare via, stacca il token diventando l'unico repositorio dell'associazione di sicurezza.
Possibili attacchi:

- Scambiarsi i compiti. L'invigilator ha il compito di impedire che ciò accada.
- Falsificare il codice a barre, scambiando il proprio con il codice uno studente bravo. Questo apre lo scenario della messa in sicurezza del token, cioè un problema di sicurezza fisico. Il problema di sicurezza è garantire l'integrità del codice ovvero garantire un'associazione sicura tra anagrafica e codice. L'invigilator, allora, pone una firma grafometrica (o un timbro) a cavallo tra il codice e l'anagrafica.

- Problema della randomicità dell'assegnamento. Bisogna garantire che i compiti siano assegnati in maniera casuale, per evitare che chi distribuisce i compiti possa avere dei conflitti di interesse. Per garantire la randomicità deve essere lo studente a scegliere il compito.
- Randomicità della consegna. Analogamente all'assegnamento, bisogna garantire che al momento della consegna non ci siano possibilità di risalire all'identità della persona (ad esempio conservando il compito in cima alla lista). Anche in questo caso il problema viene risolto facendo inserire allo studente il compito in maniera casuale nella pila dei compiti.
- L'invigilator potrebbe memorizzare le domande del compito, al momento dell'autenticazione. Risolvibile comprendo il compito, lasciando solo il token.
- Contrassegno di un compito. In questo caso si avrebbe la deanonimizzazione di uno studente a caso. Questo, su grandi numeri non ha molto senso. Su piccoli numeri non ha senso l'anonimato.

Descrizione del protocollo:

1. **Preparation**.
   (a) Estrazione random dal database che genera la tupla di domande
   (b) Estrazione random di lunghezza n dell'identificativo del test (codice a barre)
   (c) Inserimento nel database dei voti la tabella `(it_test, voto)` dove il voto sarà vuoto
   (d) Stampa del test con id del test, modulo di autenticazione (compreso un altro id del test), le domande e il modulo per le risposte (righe)
   (e) Costituzione del test firmato tramite firma grafometrica (o timbro) sul test.

   Finita la parte di preparazione, la pila dei test firmati passa all'invigilator. Se l'invigilator e l'examinator sono la stessa persona il passaggio è vacuo.
2. **Testing**.
   (a) Un test firmato arriva randomicamente al candidato
   (b) Il candidato riempie il modulo di autenticazione del test firmato
   (c) L'invigilator passa per l'autenticazione
   (d) Il candidato fornisce il documento e il test con l'operazione di copertura del compito.
   (e) L'invigilator controlla il documento e verifica che il candidato sia registrato e abbia la qualifica giusta per sostenere l'esame
   (f) Il candidato svolge l'esame
   (g) Quando il candidato finisce realizza il compito finito
   (h) Il candidato taglia il token e consegna il test in maniera casuale

   Terminata questa fase, l'invigilator passa i compiti all'esaminatore.
3. **Marking**.
   L'esaminatore accede al database dei voti, fa una query sull'id del test, prende la tupla ottenuta ed inserisce il voto. Finito il marking si passa alla fase di notification.
4. **Notification**.
   Lo studente fornisce il token, che viene scansionato. Il sistema fa una query sul database dei voti e ritorna il voto.

Disponibilità

Definizione: *Il sistema sia operante e funzionante in ogni momento.*
Sembrerebbe essere una proprietà funzionale, invece è una proprietà di sicurezza in quanto ci può essere la possibilità che ci siano attaccanti che intendono minare la disponibilità, ovvero rendere indisponibile il servizio ottenendo dei benefici impliciti. La disponibilità, quindi, non è altro che la proprietà di mantenere il sistema operante contro un insieme di richieste illegittime. Esistono misure di sicurezza a livello kernel che affrontano il problema. La misura di sicurezza più banale sarebbe limitare la richiesta, rischiando però di ottenere un servizio lento.

3.3 Proprietà di livello 3

Controllo di accesso o variante di controllo

Definizione: *Ciascun utente abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato.*
Ovvero fornire a che ciascun utente riceve uno specifico set di autorizzazioni, da qui controllo di accesso alle risorse. Per questo è anche detto proprietà di autorizzazione. Autenticazione vuol dire essere riconosciuti in maniera inoppugnabile dal sistema, cioè poter affermare che l'entità è chi dice di essere. Una cosa è riconoscere l'agente, una cosa è abbinare all'agente specifici servizi, funzioni, visibilità. Questo abbinamento si chiama controllo di accesso o autorizzazione. Parlare di autorizzazione suppone che siano definiti dei profili autorizzativi. Se sono definiti, ciascun utente riceve degli specifici profili autorizzativi cosi che sarà autorizzato ad accedere a dei profili definiti nel profilo autorizzativo.

Politiche di sicurezza

La **Security Policy** stabilisce quelle che sono le possibilità di azione all'interno di un sistema per ogni entità. Ovvero, stabilisce quali sono i profili autorizzativi da associare ad un profilo di un sistema. Rispetto alla Privacy Policy, la Security Policy descrive le operazioni che si possono fare in un sistema, regola i flussi informativi di dati e la sicurezza di un sistema. Una Security Policy che concede diritti a tutti ha seri problemi di sicurezza. La Privacy Policy invece definisce come vanno trattati i dati.

Quando aggiungiamo una regola nella policy abbiamo il problema della di coesistenza della singola regola con le altre, infatti non siamo sicuri che inserendo la nuova regola, la logica del sistema stia ancora in piedi.

Il concetto di ruoli è un concetto chiave. Il ruolo è un insieme di autorizzazioni alle quali associamo un certo numero di utenti. L'utente è la singola entità partecipante nella policy. È inoltre, importante spiegare il concetto di **modalità**. Una modalità è un regolatore per un dato insieme di operazioni. Le modalità sono obbligo, permesso, divieto, discrezione. Es. *avere il permesso di...* o *avere l'obbligo di....*
Il problema principale dei ruoli che causa inconsistenza è dovuto alla coesistenza delle singole regole, ovvero il problema di intersezione dei ruoli. Se partizionassimo l'insieme dei ruoli, ci sarebbe il problema della ridondanza. L'intersezione è una trovata implementativa per ottimizzare e ridurre la policy, e come tutte le trovare implementative rischia di avere problemi di inconsistenza.
Un approccio più semplice è quello delle priorità, che sono alla base della nostra vita, in quanto regolano tutte le nostre scelte. Questo tipo di risoluzione tramite priorità è quello che accade tipicamente nella configurazione di firewall.

MAC (Mandatory Access Controll)

Si tratta di un **modello di sicurezza**, ovvero una meta-policy, un modo per scrivere la policy. Questo modello realizza **policy mandatorie** ovvero basate sulla modalità *obbligatorio*. Queste sono policy non usate nei nostri sistemi operativi, in quanto si tratta di policy che vincolano molto l'utente, le operazioni che si possono fare e i dati trattati. Le policy mandatorie non sono modificabili né rimovibili dal sistema né bypassabili. Infatti sono generalmente utilizzati in applicazioni militari come ad esempio Bell-LaPadula e Biba. Bell-LaPadula fa valere la proprietà di confidenzialità, mentre Biba fa enforcing di integrità.
Esistono anche sistemi operativi ad alta sicurezza come SELinux, AppArmor, Tomoyo. In particolare, SELinux aggiunge un contesto di esecuzione ai processi che girano sulle macchine. Questi sono definibili, quindi realizzano un ulteriore controllo del flusso di esecuzione, che realizza per mezzo di processi, ulteriore rispetto al controllo fornito da ACL. Le ACL da sole non sono così protettive. Ad esempio, se applicativo che gira con certi permessi ha una vulnerabilità, sfruttando tali vulnerabili si ha sulla macchina vittima un processo alieno. SELinux, per mezzo dei contesti può discernere il contesto applicativo dal processo dell'attacco, quindi fornisce un ulteriore livello di controllo. Con una policy SELinx in modalità enforcing, è meno facile fare privileges escalation.

RBAC (Role Based Acces Control)

I modelli discrezionali sono sempre modelli a ruoli. DAC e RBAC sono sinonimi. Le ACL, esprimono dei permessi (permesso di lettura, scrittura, ecc), quindi la policy sottesa è una policy discrezionale con modalità di base di tipo permesso.

Meccanismi implementativi per le policy ACM:

- **Matrice**: Una matrice che indica da un lato i soggetti e da un lato gli oggetti. Matrice per ogni singola modalità. L'associazione tra soggetto e oggetto definisce il permesso.

Meccanismi implementativi per le policy ACL e CaL:

- **Access Control List**: Ogni colonna dell'ACM registrata con lo specifico oggetto
- **Capability List**: Ogni riga dell'ACM registrata con lo specifico soggetto

Possibile domanda: **Nesso tra ACL e ACM**. L'ACL si riferisce ad uno specifico utente, è quindi una linea di un ACM. Le colonne, invece, dove il perno è l'oggetto, fornisce una capatibily list relativa all'oggetto.

3.4 Tassonomia di attaccanti

Vari moventi:

- Ricchezza
- Informazioni sensibili
- Potere
- Gloria
- Divertimento
- Altro

Varie classificazioni:

- Diritti
- Risorse
- Esperienza
- Rischio accettato
- Altro

3.4.1 Modelli di attaccante

Un modello di attaccante specifica le capacità offensive di un preciso attaccante. Un modello di attaccante fa una generalizzazione, ed astrae l'attacco dal contesto. Capacità che prescindono dalla realtà. Un modello di attaccante sta agli attaccanti reali come la complessità asintotica sta alla complessità empirica di un algoritmo.

È possibile trovare un modello generale di attaccante? Ogni affermazione di sicurezza ha senso illimitatamente ad un modello. Cambiare il modello potrebbe cambiare il valore di verità di un'affermazione si sicurezza. Fare un'analisi del caso peggiore richiede un modello massimamente offensivo. Fissato il modello potremmo fare un'analisi generale come l'andamento asintotico per l'analisi di complessità.

Modello di attaccante Dolev-Yao (DY)

L'attaccante é unico e superpotente, ovvero controlla l'intera rete, ma non può violare la crittografia. Dal nome dei suoi autori (1983). Si dimostra che un insieme di attaccanti collusi equivale ad DY.
Tale modello risale ad una ricerca degli anni 70-80. Epoca dello spionaggio, nel mezzo della guerra fredda. Questo modello è una caratterizzazione astratta, senza tecnicismi di un attaccante. Il modello è basato su due spie della corona, dislocate nel mondo (uno a New York ed uno a Pechino). Vogliono comunicare in modo sicuro. Si fidano l'uno dell'altro ma hanno un problema: il mondo intero, che è contro di loro. Nel caso peggiore: tutta l'energia del mondo è concentrata a rompere la sicurezza della loro comunicazione. Ciò è equivalente ad avere un unico, super potente attaccante che equivale a tutto il mondo concentrato nel rompere quella comunicazione.

Domanda: perché consideriamo la crittografia inviolabile? Supponiamo di dover analizzare il caso di un palazzo che crolla e devo valutare se la colpa, di tale crollo, è dell'impresa o del cemento depotenziato. Il

cemento depotenziato fa le veci del poter violare la crittografia, pertanto se il modello d'attaccante potesse violare anche la crittografia, non sarebbe possibile fare un'analisi sul progetto di un protocollo. Se mi garantiscono che il cemento era buono, ho campo libero e faccio un'analisi sulla bontà della progettazione e realizzazione dell'edificio. Il nostro obiettivo non è valutare la bontà dell'algoritmo crittografico, ma la bontà del sistema di sicurezza. La crittografia è un single point of failure. Se la crittografia fosse buona, non è detto che il sistema che usa tale algoritmo crittografico sia sicuro: dipende da come è stato progettato il sistema di sicurezza. Al crittografo spetta di lavorare in modo che il crittosistema sia sicuro e rispetti alcune proprietà matematiche, ma il problema dell'esperto di sicurezza è di progettare un buon sistema di sicurezza che sfrutti al massimo l'algoritmo crittografico fatto dal miglior crittografo del mondo.

Modello di attaccante General Attacker (GA)

Chiunque può essere attaccante senza interesse a colludere con altri, al peggio con capacità di totale controllo della rete, ma senza violare la crittografia. Più aderente di DY al panorama tecnologico attuale.
Il modello precedente era specchio dell'epoca in cui è stato pensato e teorizzato. L'ipotesi di condivisione fatta da DY non vale più poiché tutti vogliono fare le scarpe a tutti, perciò non hanno bisogno di colludere. È più aderente all'epoca corrente ed è diverso da quello DY. L'utilizzo del nuovo modello di attaccante cambia le carte in tavola: i protocolli creati per essere sicuri da un attacco DY, non è detto siano sicuri dall'attacco del GA (General Attacker).

4 Crittografia

La crittografia è una scienza, matematica pura, che consiste nel trasformare un testo in altro che sia in qualche modo semanticamente equivalente, che conservi il significato. Il primo la chiamiamo **testo in chiaro**, il secondo **crittotesto**. Con l'encryption traduco il testo in chiaro in un altro testo mantenendo la semantica, in modo che nessuno leggendo il testo cifrato può capire la semantica. Serve però anche un metodo robusto per fare l'operazione inversa. Se questo metolo l'avessero tutti non andrebbe bene. Voglio implicitamente che solo il mio interlocutore possa fare l'operazione inversa.

Enigma è una macchina che traduce del testo in chiaro in un testo cifrato con una chiave che cambia. Solo i tedeschi riuscivano a cifrare e decriptare. Turing ruppe enigma, ovvero ha fatto in modo che gli alleati potessero decifrare pur senza conoscere i tecnicismi della macchina. Un crittosistema è una *coppia di algoritmi*, uno per la cifratura e uno per la decifratura.
Quello di teniamo nascosto, in genere, è uno dei due parametri, in genere la chiave (il parametro k). Algoritmi che tutti conoscono e possono eseguire, ma non tutti conosciamo tutti i parametri. Quello che teniamo nascosto è il parametro che chiamiamo *chiave*. Se solo i due interlocutori conoscono la chiave, possono parlare in maniera cifrata. Gli altri, anche se conoscono gli algoritmi, non possono decifrare il messaggio.

Bisogna però far fronte al problema dell'utilizzo distribuito del sistema. Più lunga è la chiave più tentativi di bruteforcing si deve fare, sempre nelle nostre possibilità, ma estremamente difficile. Il crittosistema sancisce la forma del crittotesto, sia la lunghezza sia i contenuti. Vedremo più avanti come ci si può scambiare la chiave segreta.

4.1 Segretezza di una chiave

Anche se una chiave é mantenuta segreta, un attaccante ha sempre un modo basilare per tentare di indovinarla (bruteforcing).

Data k, vale sempre $Pr[guess(k)] = \frac{1}{2^{|k|}}$.

Ecco perché vorremmo sempre lavorare in pratica con chiavi più lunghe possibile. Un crittosistema é sicuro quando la probabilità di indovinare una chiave non é significativamente superiore a questa.

Crittoanalisi: Dedurre un testo in chiaro pur non conoscendo la chiave, in gergo *rompere*. Nel processo di crittoanalisi è cruciale il fatto si riconoscere l'output atteso.

4.2 Crittografia simmetrica

Definizione: L'unico modo per estrarre il testo in chiaro da un crittotesto é decodificare quest'ultimo con la stessa chiave usata per costruirlo. Ovvero:

1. $D(E(m,k),k) = m$
2. $\forall k_1.\ k_1 \neq k \longrightarrow D(E(m,k),k_1) \neq m$

Alcuni esempi di crittosistemi simmetrici:

- Cifrario di Cesare, DES, 3DES
- Tipica lunghezza di una chiave 128/256 bit
- Velcità

Crittografia simmetrica in rete

Fissato un agente A (macchina, utente), esso sia munito di chiave simmetrica, indicata come k_a.
k_a è detta **chiave a lungo termine** perché il suo intervallo di validità è molto lungo, indipendente dalla specifica sessione di comunicazione. Ma ci sono alcuni limiti:

1. A non vuole rivelare k_a a B, quindi questo non può decriptare. Sia quindi k_{ab} una chiave dedicata specificatamente a questa sessione fra A e B. Questa è detta **chiave a breve termine** o **chiave di sessione**, che può essere condivisa fra i due agenti
2. Servirebbe una chiave di sessione per ogni coppia di agenti che vogliamo far comunicare fra loro
3. Come condividere k_{ab} fra A e B pur mantenendola confidenziale?

Quindi si manifesta subito il limite fondamentale della crittografia simmetrica: **come condividere il segreto iniziale su una rete insicura**.

4.3 Crittografia asimmetrica (1978)

Tecnologia nuova, (1978). Le chiavi crittografiche, diventano due, laddove nel simmetrico ce n'era solamente una. Quindi parliamo di **coppia di chiavi**. Questa coppia, sono uno l'inversa dell'altra quindi abbiamo k e k^{-1} . Ciascuna chiave **non** si può ricavare dall'altra (problema computazionalmente intrattabile), pertanto la coppia va generata monoliticamente. Se invio k in broadcast, nessuno riesce a calcolare k^{-1} . Inoltre, se ho un crittotesto costruito con una chiave, posso fare la decodifica ma non devo utilizzare la stessa chiave, ma devo utilizzare la chiave inversa. Ovvero:

1. $D(E(m,k),k^{-1}) = m$
2. $\forall k_1.\ k_1 \neq k^{-1} \longrightarrow D(E(m,k),k_1) \neq m$

Alcuni esempi di crittografia asimmetrica:

- (DSA, RSA)
- Tipica lunghezza di una chiave 1024 bit
- Generalmente più lenta della crittografia simmetrica

Esempio di usi di RSA semplificato:

- Si pubblichi $n = 33$; si prendano $k = 3$ e $k^{-1} = 7$
- Siano $E(x,e) = x^e mod n$ e $D(x,d) = x^d mod n$
- Allora, preso $m = 7$, si ha $E(m,k) = 13$ quindi, correttamente $D(E(m,k),k^{-1}) = 7$

La robustezza non cambia dal caso simmetrico. L'unica cosa che cambia è che sapendo che la decodifica va fatta con l'inversa, comunque prendo una chiave diversa dall'inversa, sicuramente non ottengo il messaggio in chiaro. Ecco perché, nel caso della crittografia simmetrica ottengo il testo solo se c'è una certa relazione tra le due chiavi utilizzate per la codifica e la decodifica. Normalmente, asintoticamente, la asimmetrica è più lenta. Standard oggigiorno RSA2048 e AES256.

Crittografia asimmetrica in rete

Fissato un agente A (macchina, utente), esso sia munito di una coppia di chiavi asimmetriche, indicata come k_b e ${k_a}^{-1}$, a lungo termine tale che:

- k_b resa nota a tutti, pertanto detta **chiave pubblica** di A
- ${k_a}^{-1}$ segreto di A, pertanto detta **chiave privata** di A

Scenario tra Alice e Bob, che vogliono comunicare su un canale insicuro. Con il caso asimmetrico, potrei fare vari tentativi. Alice codifica il proprio testo in chiaro con la chiave primaria di A (k^{-1}). Ma Alice ha due chiavi. Convenzionalmente chiamiamo quella senza il -1 chiave pubblica, e quella con il -1 chiave privata. Pubblica vuol dire che chiunque la conosce, mentre la privata la tiene per se. Stessa cosa per B. Quindi A può usare 3 chiavi su 4.

Quindi il limite della crittografia asimmetrica è la **certificazione**: come associare correttamente una chiave pubblica la suo legittimo proprietario, ovvero al legittimo proprietario della metà privata.

Cosa succede se quest'associazione non funziona? Esempio:

- A intenda spedire m a B in maniera confidenziale
- A intende quindi spedire m_{k_b}
- A pertanto prende la chiave k ma erroneamente crede che sia $h = k_b$ mentre risulta $k = k_c$
- A costruisce m_k e lo spedisce a B
- B, ricevuto m_k, lo descripta ma ottiene m' con $m \neq m'$
- C, intercettato m_k, può decriptarlo con ${k_c}^{-1}$ ottenendo m

Restano dunque due limiti da superare:

- **Crittografia simmetrica**: come condividere il segreto iniziale fra una coppia di agenti che vogliano comunicare in maniera sicura
- **Crittografia asimmetrica**: come verificare il proprietario di una chiave pubblica (certificazione)

Crittosistema sicuro

Definizione: Sia calcolato $E(m, k)$ per ogni testo m e chiave k. Sia calcolato $D(E(m, k), k_1) = n$ per ogni chiave k_1 tale che $k_1 \neq k$ se il crittosistema è simmetrico, o $k_1 \neq k^{-1}$ se il crittosistema è asimmetrico. Allora l'accesso a n **non aumenti significativamente** la probabilità di un attaccante di indovinare m o sue porzioni.

4.4 Hash crittografico

Una funzione hash crittografica è una funzione hash che prende un blocco arbitrario di dati e ritorna una stringa di dimensione fissa di bit, il *valore dell'hash crittografico*, in modo che qualsiasi cambio al dati causerà un cambio del valore hash.
Una funzione di hash crittografico ideale deve rispettare quattro proprietà:

1. Deve essere computazionalmente facile calcolare l'hash di un qualunque tipo di messaggio
2. Non sia possibile generare un messaggio da un hash dato
3. Non sia possibile modificare un messaggio senza cambiare l'hash
4. Non sia possibile trovare due messaggi differenti con lo stesso hash

Esempio: SHA-1.

Esempio di uso di hash: CTSS + hashing

CTSS può essere potenziato con una funzione hash (Cambridge University, 1967): il file delle password memorizza l'hash di ciascuna password.

Salting

Salt indica quel messaggio random aggiunto a una password per proteggerla da attacchi dizionario prima che vi si applichi l'hash. Storicamente 12 bit, oggi insufficienti.

In particolare, Unix: genera salt per ciascuna nuova password da memorizzare (mettendo il file delle password nel file `/etc/passwd` sul quale tutti possono leggere, ma solo root può scrivere, usa la password per codificare una stringa di zeri insieme a salt mediante la password encryption funcion `crypt(3)` basata su DES.

4.5 Freshness

Definizione: Importante attributo di almeno due proprietà di sicurezza — segretezza e autenticazione — il quale stabilisce che esse valgano di recente.
Esempi: uso di chiavi fresche (idealmente one-time), presenza dell'interlocutore autenticate per evitare **replay attack**.
Alcune misure:

- **Timestamp** marcatore temporale (chi vuole dare la garanzia)
- **Nonce** numero random che è usato una sola volta (chi vuole ricevere la garanzia)

5 Protocolli di sicurezza basilari

5.1 Protocolli basilari per la freshness

- **Timestamp**. Chi vuol **dare** garanzia di freshness inserisce il timestamp (pertanto potrebbe barare), associando al messaggio target in maniera affidabile (altrimenti chiunque altro potrebbe attaccare)
- **Nonce**. Chi vuole **ricevere** la garanzia di freshness pubblica la nonce e aspetta traffico che la citi, il quale risulterà posteriore all'istante di creazione della nonce, pur potendo citare materiale vecchio. Non servono orologi sincronizzati. È il primo strumento per ricevere la freshness

5.2 Sintassi dei messaggi crittografici

Atomici

- **Nomi di agenti**: A, B, C, ...
- **Chiavi crittografiche**: $k_a,\ k_b,\ ...,\ {k_a}^{-1},\ {k_b}^{-1},\ ...,\ k_{ab},\ k_{ac},\ ...$
- **Nonce**: $N_a,\ N_b,\ ...$
- **Timestamp**: $T_a,\ T_b$
- **Digest**: $h(m),\ h(n)$
- **Etichette**: *"Trasferisci 100 dal conto di..."*

Composti

- **Concatenazioni**: $m,\ m'$ dove ciascuno può essere un crittotesto
- **Crittotesto**: m_k dove il testo in chiaro può essere concatenazione

Domanda: si può autenticare un messaggio concatenato? Autenticare un crittotesto si può fare grazie alle chiavi. L'operazione più esterna è quella con la quale diamo il nome. Quindi l'autenticazione di un messaggio concatenato non si puo' fare, posso scindere le parti del messaggio, ma non su chi ha formato la coppia, perché chi la forma è la concatenazione.

5.3 Sintassi dei messaggi crittografici

Alice-Bob notation. Numero passo, mittente, freccia, ricevente, due punti, messaggio crittografico - da ripetere per ciascun passo del protocollo.

1. $A \longrightarrow B \ : \ A, N_a$
2. $B \longrightarrow A \ : \ N_{a\,k_b{}^{-1}}$

Nelle assunzioni fatte, l'etichetta del mittente diventa irrilevante a qualunque passo. Questa notazione ha dei limiti, in particolare con quella presentata per WATA (MSC). La differenza sta nel fatto che MSC aggiunge dei box sotto ciascun agente, i calcoli non di rete. Quindi quello che fa un agente quando riceve un messaggio non viene specificato. L'etichetta mittente, dal punto di vista della sicurezza, non serve a nulla. Una delle prime intenzioni dell'attaccante è far credere di star parlando con qualcuno che non è. Scrivere l'etichetta mittente, serve solo all'umano, ma non e' un'indicazione per il ricevente. Sarà il corpo del messaggio che dirà al ricevente chi è il mittente. Perché la sorgente di informazione è inaffidabile. Dobbiamo lavorare sul corpo del messaggi.
Il corpo del messaggio potrebbe trasportare anche il certificato della chiave pubblica. Non e' affidabile ugualmente.

5.4 Protocolli basilari per la segretezza

Obiettivo: segretezza del messaggio m per A e B.
Un protocollo decade quando decade un prerequisito e quando non è stata considerata qualcosa essa stessa fa decadere la sicurezza del protocollo. Va aggiornato aggiungendo l'assunzione mancante o addirittura va cambiato il protocollo.

Segretezza con crittografia simmetrica

Prerequisito: chiave k_{ab} sia condivisa fra A e B soli.

1. $A \longrightarrow B \ : \ m_{kab}$

Ogni chiave è unica, con la stessa chiave cifro e decifro. Se A manda a B un messaggio cifrato con una chiave di sessione, solo A e B possono comunicare. La condizione necessaria è che la chiave di sessione sia segreta solo per A e B. Il protocollo è elementare e semplice, ma funziona solo per le ipotesi che andiamo a fare. Le ipotesi sono tanto quante importanti quanto le basi del protocollo.

Domanda: Come si può garantire che la chiave di sessione sia segreta? Nel caso *asimmetrico*, il mittente deve codificare il messaggio per mezzo della chiave pubblica del destinatario.

Segretezza con crittografia asimmetrica

Prerequisito 1: B abbia una chiave privata valida (sicura, non scaduta)
Prerequisito 2: A possa verificare che k_b è di B

1. $A \longrightarrow B \ : \ m_{kb}$

Si noti che serve **certificazione**. Il certificato associa la chiave pubblica k_b col suo proprietario B e consente, in tal caso, ad A di essere certo che la chiave pubblica suddetta è di B poiché è certificata.

L'assunzione fondamentale, stavolta, è duplice. B possiede una chiave privata valida, ovvero che non la conosca nessun altro e che non sia scaduta. Una chiave privata scade quando scade il certificato per la chiave pubblica corrispondente. Inoltre, il prerequisito numero due, è che il mittente possa verificare che la chiave pubblica del destinatario sia quella corretta. Se A si sbagliasse e prendesse la stringa che corrisponde alla chiave pubblica di C, quando A volesse invece comunicare con B, finirebbe che quello che può andare a decodificare il traffico non sarebbe B ma C, e potrebbe essere erroneo per A.
A necessita una tabella che associ la chiave all'individuo.

5.5 Protocolli basilari per l'autenticazione

Obiettivo: Autenticazione di A con B.

Crittografia simmetrica

Prerequisito 1: Chiave k_{ab} sia condivisa fra A e B soli
Prerequisito 2: B possa verificare il prerequisito 1

1. $A \longrightarrow B$: "*Sono io*!"$_{k_{ab}}$

B deve poter verificare il primo prerequisito per poter associare l'identità dell'agente col quale sta condividendo la chiave k_{ab} di sessione. Se B utilizza una chiave sbagliata per decodificare il messaggio non otterrà "Sono io!" ma un qualcosa di non comprensibile e, pertanto non autenticherà. I prerequisiti sulla chiave ottengono il risultato desiderato.

Il prerequisito numero uno è uguale a quella della segretezza, cioè avere una chiave confidenziale per entrambi. Prerequisito due: B possa verificare il prerequisito 1. Che senso ha? B deve conoscere chi altri conosce la chiave condivisa. Per la segretezza non necessariamente abbiamo bisogno dell'autenticazione dell'identità. Dal punto di vista della segretezza l'identità dell'interlocutore non è fondamentale. Allora, per autenticare A, B ha bisogno di capire l'identità dell'agente con cui condivide la chiave di sessione. A quel punto, nel momento in cui B riceve il primo messaggio, il messaggio numero 1 è cifrato con la chiave di sessione che B sa di condividere con A, quindi B sa che dall'altra parte non può che esserci A. *Domanda*: qual è la differenza tra i messaggi dell'autenticazione e della segretezza? Nella segretezza c'è un messaggio generico. Non è il contento del messaggio che autentica A, ma è il fatto che il messaggio, qualsiasi cosa contenga, è cifrato con la chiave che solo A e B conoscono.

Crittografia asimmetrica

Prerequisito 1: A abbia una chiave privata valida
Prerequisito 2: B possa verificare che k_a è di A

1. $A \longrightarrow B$: "*Sono io*!"$_{k_a{}^{-1}}$

La cosa che cambia è la caratteristica della chiave. A per ipotesi è l'unico che conosce la sua chiave privata. La certificazione ci vuole, perché il destinatario ha bisogno di sapere a chi è associata la chiave. Non conosce la chiave privata dell'interlocutore, ma conosce la chiave pubblica. Quindi prende la chiave pubblica che usa per decodificare il messaggio. Se ottiene qualcosa, significa che dall'altra parte c'è veramente l'interlocutore previsto. Il prerequisito 2, nel caso simmetrico e asimmetrico, è analogo. Cioè verificano l'identità potenziale dell'interlocutore, garantito appunto dal prerequisito 2.

5.6 Combinare segretezza e autenticazione

Obiettivo: segretezza del messaggio m per A e B, autenticazione di A con B.

Se servono entrambe le proprietà non facciamo altro che "sommare" I due protocolli. Li sommo nel senso che sommo innanzi tutto I prerequisiti, li metto insieme.

Crittografia simmetrica

Prerequisito 1: Chiave k_{ab} sia condivisa fra A e B soli
Prerequisito 2: B possa verificare il prerequisito 1

1. $A \longrightarrow B$: $m_{k_{ab}}$

Il protocollo di fatto non cambia, qualsiasi cosa metto dentro al messaggio, verrà inviato nel crittotesto. Questo fattore mi agevola nell'autenticazione. Non importa cosa trasporto. Il messaggio funzionerà per autenticare A con B.

Crittografia asimmetrica

Prerequisito 1: B abbia una chiave privata valida
Prerequisito 2: A possa verificare che k_b è di B
Prerequisito 3: A abbia una chiave privata valida
Prerequisito 4: B possa verificare che k_a è di A

1. $A \longrightarrow B$: $\left\{m_{k_a{}^{-1}}\right\}_{k_b}$ oppure $A \longrightarrow B$: $\{m_{k_b}\}_{k_a{}^{-1}}$

Applico la cifratura a cascata, prima l'una e poi l'altra.

Tutti i protocolli basilari sono vulnerabili ad **attacco di replica**. Perché non usano timestamp o nonce. Chi riceve il traffico non ha alcun limite di freshness. Per superare questi problemi, dobbiamo inserire nonce o timestamp.

5.7 Come ottenere integrità?

Qualunque messaggio sulla rete potrebbe essere alterato, in particolare anche un messaggio protetto per segretezza e autenticazione. Un approccio potrebbe essere l'utilizzo di un qualche checksum del tipo usato nei protocolli di rete. Usare una funzione hash come checksum?

1. $A \longrightarrow B \ : \ m, h(m)$

Ma l'attaccante potrebbe anche cambiare l'hash, sostituendolo con l'hash calcolato a partire dal messaggio alterato. L'utilizzo di qualche checksum (funzione hash) ha uno scopo funzionale ma non protettivo. Così che per via dei requisiti dell'hash, se l'attaccante altera il messaggio cambia anche l'hash.

Tralasciando l'uso della crittografia simmetrica, potenziamo l'idea precedente *autenticando l'hash.* Ovvero **criptandolo con la chiave privat del mittente**.

1. $A \longrightarrow B \ : \ m, h(m)_{k_a{}^{-1}}$

Ricevuto il messaggio, B, applica la funzione hash alla prima componente, decodifica la seconda componente, confronta i due risultati e ottiene la garanzia di integrità se e solo se essi combaciano.
Questa è la **firma digitale**.

$$sign_A(m) = m, h(m)_{k_a{}^{-1}}$$

Uno dei limiti della crittografia simmetrica è che è poco scalabile, per via dell'infrastruttura di certificazione, quindi la versione asimmetrica di $m, h(m, k_{ab}))$ che scala meglio è: $m, \left\{h(m)_{k_a{}^{-1}}\right\}$.

5.8 Firma digitale

La firma digitale si basa su crittografia asimmetrica, pertanto ha bisogno di certificazione delle chiavi pubbliche. Ottiene sia **integrità del documento** che **autenticazione del firmatario**, contrariamente alla firma cartacea. Un sistema di firma consiste in due algoritmi, uno di creazione e uno di verifica. Creare una firma è più che criptare, verificarla è più che decriptare. L'obiettivo da garantire è che *nessuno deve poter firmare per conto di un altro*, mentre *chiunque deve poter verificare la firma di chiunque altro.*
Con la firma digitale, poiché usiamo hashing all'interno della firma combinato con l'encryption, otteniamo integrità. Essendo la versione che scala meglio basata su crittografia asimmetrica, che per avere senso ha bisogno di certificazione, nello stesso tempo stiamo ottenendo autenticazione.
Quindi accade che la firma digitale fornisce autenticazione dell'utente e integrità del messaggio, contrariamente alla firma grafometrica che era intesa solo per l'autenticazione.
Ovviamente, se la certificazione è vulnerabile la mia chiave non sarà più esclusivamente associata a me.

Creare una firma

Prendo il messaggio, ne faccio l'hash e lo codifico con la chiave privata del mittente.

1. cleartext $\xrightarrow{\text{hashing}}$ **digest**
2. **digest** $\xrightarrow{\text{Encrypt with sender's private key}}$ **{encrypted digest}**
3. $\Big(\text{cleartext} + \textbf{\{encrypted digest\}}\Big) = \text{digital signature}$

Verificare una firma

Alla ricezione della coppia, il destinatario decripta il digest cifrato e ottiene un altro digest, se è uguale a quello cifrato non c'è stata alterazione.

1. cleartext + **{encrypted digest}**
2. **{encrypted digest}** $\xrightarrow{\text{Decripy with sender's public key}}$ (**digest 1**)

3. cleartext $\xrightarrow{\text{hashing}}$ **digest 2**

4. **digest 1** $\stackrel{?}{=}$ **digest 2** $\begin{cases} \text{yes} \rightarrow \textbf{valid} \\ \text{no} \rightarrow \textbf{invalid} \end{cases}$

Domanda: qual è la proprietà di sicurezza che si può ottenere solo in combinazione con un'altra? **l'integrità**.

5.9 Protocolli basilari per l'integrità

Obiettivo: integrità del messaggio m nella trasmissione da A a B.

Omettiamo il caso della crittografia simmetrica.

Crittografia asimmetrica

Prerequisito 1: A abbia una chiave privata valida
Prerequisito 2: B possa verificare che k_a è di A

1. $A \longrightarrow B \ : \ sign_A(m)$

Stessi prerequisiti dell'autenticazione di A con B.

5.10 Combinare tutte le proprietà di livello 1

Obiettivo: segretezza del messaggio m per A e B, autenticazione di A con B, integrità di m nella trasmissione da A a B Omettiamo il caso della crittografia simmetrica.

Crittografia asimmetrica

Prerequisito 1: B abbia una chiave privata valida
Prerequisito 2: A possa verificare che k_b è di B
Prerequisito 3: A abbia una chiave privata valida
Prerequisito 4: B possa verificare che k_a è di A

1. $A \longrightarrow B \ : \ sign_A(m_{k_b})$

Facciamo il punto

- Tutti i protocolli basilari visti sono vulnerabili a replay attack
- Non usano alcun meccanismo di freshness
- Crittografia simmetrica: condivisione segreto iniziale
- Crittografia asimmetrica: certificazione

5.11 Protocollo Diffie-Hellmann (1976)

Obiettivo: realizzare un segreto condiviso con un sistema simmetrico su una rete ostile.
Risolve il problema fondamentale della crittografia simmetrica, anche se anticipa la crittografia asimmetrica, consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro (pubblico) senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. La chiave ottenuta mediante questo protocollo può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di crittografia simmetrica.

1. A e B concordano due parametri pubblici α e β
2. A genera X_a random quindi $Y_a = \alpha^{X_a} \ mod \ \beta$
3. B genera X_b random quindi $Y_b = \alpha^{X_b} \ mod \ \beta$
4. A e B eseguono il protocollo di scambio
    1. $A \longrightarrow B \ : \ Y_a$

2. $B \longrightarrow A \ : \ Y_b$

5. Alla recezione di 1. B calcola ${Y_a}^{X_b} \ mod \ \beta$

6. Alla recezione di 2. A calcola ${Y_b}^{X_a} \ mod \ \beta$

Si ha:

- ${Y_b}^{X_a} \ mod \ \beta = {Y_a}^{X_b} \ mod \ \beta$
- Posto $k_{ab} = {Y_b}^{X_a} \ mod \ \beta$, allora A e B condividono k_{ab}

Quindi, k_{ab} è segreta? L'attaccante può intercettare Y_a e Y_b. Per calcolare k_{ab} gli basterebbe trovare X_a o X_b, dovrebbe cioè risolvere il problema del **logaritmo discreto**, che è un problema intrattabile.

Osservazione: non c'è autenticazione degli agenti, è un problema? Quando stabiliamo il segreto condiviso, serve l'autenticazione? Se sono in fase di definizione di un segreto, è importante che abbia conferma del mio interlocutore?

Certo, se non ho certezza del mio interlocutore come posso stabilire un segreto condiviso con esso? A causa dell'assenza di autenticazione è possibile che tra i due interlocutori si piazzi un intermediario senza che i due se ne rendano conto, ed è proprio qui che si realizza l'attacco. Se invece, nella sessione mi accorgo che non c'è più il mio vecchio interlocutore dall'altra parte, il protocollo funziona.

Diffie-Hellmann attacco

L'attaccante C ha una sua coppia X_c e Y_c standard. Quando A e b eseguono il protocollo di scambio, C blocco ed altera:

1. $A \longrightarrow B \ : \ Y_a$ (bloccato da C)

1'. $C(A) \longrightarrow B \ : \ Y_c$

2. $B \longrightarrow A \ : \ Y_b$ (bloccato da C)

2'. $C(B) \longrightarrow A \ : \ Y_c$

Alla recezione di **1'**, B calcola ${Y_c}^{X_b} \ mod \ \beta$.
Alla recezione di **2'**, A calcola ${Y_c}^{X_a} \ mod \ \beta$.

A questo punto A e B calcolano lo stesso valore? **No**.

Posto:

$$k_1 = {Y_c}^{X_a} \ mod \ \beta$$
$$k_2 = {Y_c}^{X_b} \ mod \ \beta.$$

Deriva che A conosce k_1 e B conosce k_2, ma $k_1 \neq k_2$. Invece C conosce sia k_1 che k_2, che calcola come:

$$k_1 = {Y_a}^{X_c} \ mod \ \beta$$
$$k_2 = {Y_b}^{X_c} \ mod \ \beta.$$

Si nota subito il **fallimento della distribuzione della chiave di sessione**.

Il primo attacco è alla funzionalità, ovvero al fallimento del requisito funzionale che A e B si calcolassero la stessa cosa. Il problema di confidenzialità arriva subito dopo. Perché C ha intercettato sia Y_a sia Y_b. A questo punto se prende Y_a ed eleva al proprio X_c, ottiene esattamente k_1.

C possiede X_c, Y_c, Y_a e Y_b quindi

$${Y_a}^{X_c} = \alpha^{{X_a}^{X}c} = k_1$$
$${Y_b}^{X_c} = \alpha^{{X_b}^{X_c}} = k_2.$$

In realtà si è venuto a creare un tunnel tra A e C, messo in confidenzialità con la chiave k_1, e un tunnel tra B e C messo in confidenzialità tramite k_2. Senza autenticazione, pertanto, A e B sono ignari dell'attacco. A si calcola una roba che pensa di condividere con B, e viceversa, ma che invece condividono con l'attaccante. Quindi, tutto ciò che A manda a B, sarà rivelato dall'attaccante e viceversa. Il problema di segretezza viene subito dopo la devastazione funzionale che A e B non si calcolavano la stessa cosa.

Quello che il protocollo vorrebbe garantire è la segretezza, non l'autenticazione. *Quindi è un problema?* Si perché la mancanza di autenticazione in fase di definizione di un segreto impedisce a ciascuno di realizzare con chi condivide il segreto. È proprio questa problema progettuale che ne inficia anche la segretezza.

La mancanza di autenticazione, espone il protocollo ad attacchi di tipo MITM (Man In The Middle). L'attaccante C sfrutta la mancanza di autenticazione per intercettare Y_a e Y_b e mandare (ad A e B) Y_c così calcolato: $Y_c = \alpha^{X_c} \; mod \; \beta$.

Visto che il protocollo non usa alcuna tecnica di *autenticazione*, C fa un **duplice attacco di autenticazione**: passi **1'** e **2'**.
Visto che il protocollo non usa alcuna tecnica di *segretezza*, C fa un **duplice attacco di segretezza**: utilizzo indebito di Y_a e Y_b.

Irrobustire Diffie-Hellmann: risolvere il MiM

Modifica che impedisca la prima parte dell'attacco:

1. $A \longrightarrow B \; : \; \{Y_a\}_{k_a{}^{-1}}$
2. $B \longrightarrow A \; : \; \{Y_b\}_{k_b{}^{-1}}$

Modifica che impedisca la seconda parte dell'attacco:

1. $A \longrightarrow B \; : \; \{Y_a\}_{k_b}$
2. $B \longrightarrow A \; : \; \{Y_b\}_{k_a}$

5.12 RSA Key Exchange

Lo scambio RSA è un'alternativa a Diffie-Hellmann, che non fa altro che utilizzare un canale confidenziale costruito con una chiave pubblica. Il mittente invia la chiave di sessione randomicamente e la manda in una **busta digitale**

1. $A \longrightarrow B \; : \; \{k_{ab}\}_{k_b}$

Anche in questo protocollo però manca l'autenticazione e l'attaccante non può calcolare la chiave di sessione.

Affinché il protocollo funzioni, A deve innanzitutto procurarsi il **certificato** (della chiave pubblica) di B.

Busta digitale

È un modo più semplice per risolvere il problema di Diffie-Hellmann.
Si usa la crittografia asimmetrica per le impostazione fondamentali, dopo aver stabilito il segreto condiviso, procedo utilizzando la crittografia simmetrica che è anche più efficiente.
Tutto ciò funziona laddove esiste un modo inoppugnabile per associare una chiave pubblica ad una qualsiasi entità. Ciò che chiamiamo *certificato*. Il certificato è la convalida di un'associazione. Certificare non vuol dire autenticare ma fare un'associazione in maniera sicura.

5.13 Certificazione (crittografia asimmetrica)

La certificazione serve ad associare una chiave pubblica al suo legittimo proprietario, ovvero al proprietario della metà privata. L'associazione è realizzata da un **certificato digitale** (anche quello cartaceo fa un'associazione, fra foto e anagrafica). Il certificato è firmato digitalmente da una **autorità di certificazione** (CA), ad esempio *Verisign*, *AOL*, *Thwate*. Il formato standard del certificato si chiama **X.509**.

Il certificato è digitale. Il problema stesso della certificazione è l'ente certificante. Laddove ci sia un ente che certifica qualcuno o qualcosa, questo ente sarà degno di fiducia, affidabile, importante, significativo. Il documento di identità ha senso perché l'ente che certifica è riconosciuto in tutto il mondo.

Associare ente e la sua chiave pubblica

Come possiamo mettere insieme entità e la sua chiave pubblica.
Semplicemente concatenandoli. Data A l'entità, cifro $\{A, K_a\}_{K_{CA}}$ (CA certificato) che funge da firma digitale. Possiamo scrivere quindi $sign_{CA}(A, K_a)$.
Ma la firma non ci protegge da segretezza. Il protocollo ha bisogno di due proprietà di sicurezza, **integrità del certificato** e **autenticazione di chi lo costruisce**. Alla ricezione del certificato, il ricevente deve verificare se il certificato è stato manomesso, quindi necessariamente serve l'integrità. Non è strano che A può avere più certificati, non sarà A a farli, ma l'ente certificante. In realtà un certificato digitale è un po' più complesso, lo standard è **X.509v3**.

Scenario di un sito web

A è il nostro sito web (URL). Di base, un sito web non necessita di un certificato, soprattutto se siamo su HTTP. Su **HTTPS** il certificato ci vuole, ma non è detto che il certificato vada bene. La certificazione è del sito per l'utente. Il sito si deve autenticare con l'utente. Il fruitore della proprietà è l'utente, chi la esercita è l'URL.
Normalmente l'autenticazione si fa con password, ma l'autenticazione dell'URL è più complessa, perché non ha una password. Noi utenti vogliamo un'indicazione di genuinità del sito. Il protocollo che sta dietro (HTTPS) è **TLS** basato su crittografia asimmetrica, pertanto necessita di certificazione. Certificazione dell'abbinamento tra chiave pubblica ed entità (il sito). Il browser è uno strumento sia per l'utente sia per il sito. Siccome il sito vuole fare HTTPS con il browser, manda il proprio certificato. Quindi chi fa la verifica del certificato (il browser), se riesce, autentica il sito e garantisce il funzionamento di HTTPS.

Osservazione: Il browser verifica la firma digitale sul certificato X509 il quale associa l'URL alla sua chiave pubblica. Poi esegue SSL con il sito codificato con questa chiave pubblica trovata dentro il certificato e se il protocollo è andato a buon fine, significa che dall'altra parte c'è davvero chi dice di essere. HTTPS autentica il sito con il browser, non è detto che non sia malevolo, ma almeno è chi dice di essere. L'obiettivo è garantire che nel dialogo tra il browser e il sito non ci sia una terza persona in mezzo.

Ma come è gestita la gerarchia dei certificati? **RootCA** o **Primary CA** che certifica tutti i certificati CA di livello inferiore, inseriti nel browser dal costruttore.

5.13.1 Infrastruttura a chiave pubblica o PKI

Per comunicare con A, a B serve il certificato di A. Schematizzato, esso ha forma

$$\{... A ... k_a ...\}_{k_{CA}{}^{-1}}.$$

Quindi A ha bisogno del certificato della CA. A sua volta, questo è firmato da un'autorità superiore, ed ha forma

$$\{... CA ... k_{CA} ...\}_{k_{CA(n-1)}{}^{-1}}.$$

Pertanto, ciascuna autorità è certificata da una superiore, fino alla **root certification authority** (RCA) o **primary certification authority**.
Una infrastruttura a chiave pubblica o sistema di certificazione consiste nella gerarchia di autorità e nelle tecnologie che esse usano.

Si ha quindi un albero di autorità di certificazione (PKI) la cui radice è una root. Ogni cvlta che l'utente chiede un ben preciso sito il browser deve risolvere l'intera catena di fiducia fino alla radice. Siccome, in genere, una root può certificare più di un'autorità la struttura non è lineare ma ad albero. Un certificato di root ha la caratteristica sintattica particolare che lo distingua dagli altri certificati, è l'unico certificato auto firmato.
La compromissione non si propaga al di sopra del PKI, ma solo nel sotto albero.

Un certificato viene chiamato invalido per vari motivi: perché è scaduto, l'ULR è cambiato oppure perché un CA revoca un certificato. La revoca può avvenire perché il proprietario del certificato perde la chiave privata, o per contenuti illegali del sito, ecc.
Non c'è uno standard per la revoca del certificato (esiste un RFC). I vari scenari di invalidità non sono tutti trattati allo stesso modo. C'è un ulteriore problema: che la RFC non è così chiaro sul comportamento che il browser deve adottare all'arrivo di un certificato scaduto/revocato.

5.13.2 Chain of Trust

Per comunicare con A, B deve risolvere la catena di fiducia.
Sia $CA(n)$ l'autorità foglia, e sia $RCA = CA(0)$.
Il certificato di root è l'unico autofirmato.

- $\{...A...k_a...\}_{k^{-1}_{CA(n)}}$
- $\{...CA(n)...k_{CA(n)}...\}_{k^{-1}_{CA(n-1)}}$
- ...
- $\{...CA(1)...k_{CA(1)}...\}_{k^{-1}_{CA}}$
- $\{...RCA...k_{RCA}...\}_{k^{-1}_{RCA}}$

La catena di fiducia di A è la lista completa dei certificati (fino a quello di root) che permettono, insieme alla chiave pubblica della RCA, di verificare il certificato di A.

5.13.3 Segretezza vs Trust

Se l'attaccante si impossessasse della chiave privata di una certa $CA(i)$ non automaticamente si impossesserebbe di alcun altra chiave privata. Inoltre, automaticamente tutti perderebbero fiducia in qualunque chiave pubblica la cui certificazione necessiti della chiave pubblica della $CA(i)$ perché l'attaccante potrebbe creare delle false $CA(i+1)...CA8n)$ aventi chiavi pubbliche con certificati falsi.
La perdita di segretezza non si propaga; la perdita di trust si propaga verso i livelli inferiori

Certificate Revocation List (CRL)

Lista certificati revocati prima della loro scadenza, per via di chiave privata smarrita o cambio Subject Identifier. È firmata dalla CA che ha emesso i certificati ora revocati. Il Subject inoltra la richiesta e il browser deve avere le CRL recenti, ma manca uno standard. Il comportamento specifico del browser all'arrivo di un certificato scaduto è incerto e imprevedibile. Le CRL devono essere firmate digitalmente perché così come ci vuole l'autorevolezza di firmare un certificato la si deve avere anche per revocare. Sostanzialmente sono una lista di certificati. Periodicamente tutte le CA sono tenute ad emettere la lista di certificati revocati. Il problema è capire come il browser tratta le informazioni di revoca.

5.13.4 Proprietà di sicurezza necessarie per un certificato

La proprietà di sicurezza necessaria è l'**integrità** perché altrimenti è possibile cambiare l'associazione, garantito dalla firma digitale, che garantisce anche l'autenticazione di chi appone la firma. Fatta in modo che solo chi possiede le capacità di firma con una specifica identità può affliggere il nome.

Domanda: Qual è la differenza tra le autorità di certificazione? Sul piano tecnico la differenza non esiste, perché entrambi fanno le stesse operazioni con le chiavi pubbliche e private. C'è solo una differenza evanescente, sociale, fatta di fiducia (fiducia con base razionale). Oggigiorno la protezione è molto digitale e poco fisica.

5.13.5 Let's Encypt

Ente di certificazione gratuito. Ottiene certificati semplici, di tipo **DV** (Domain Validate), invece del certificato **EV** (Extend Validate), più difficile da ottenere, e **OV** (Organization Validation).

5.13.6 Tipi di certificati

- **Domain Validate** (DV): dimostrano il controllo su alcuni nomi di dominio, ad esempio rispondono a `admin@mydomain.com`. Non è un processo automatico ed economico. Nessuna entità legale legata al certificato.
- **Organization Validation** (OV): Verifica supplementare al business attuale. Aggiunto il nome dell'organizzazione.
- **Extended Validation** (EV): Controllo manuale di tutti i nomi di domino da certificare. Risorse esterne per cercare informazioni (es. governi). Entità legale chiaramente vincolata.

6 Autenticazione

Quattro gruppi di ambientazioni:

- **utente-computer**: la più comune, per accedere ad un sistema (password, impronte)
- **computer-computer**: un computer ad un altro indipendentemente dal suo utente (IP, MAC)
- **computer-utente**: fondamentale per un computer remoto (sito internet)
- **utente-utente** rara (Kerberos, WhatsApp)

Spesso richieste varie combinazioni.

6.1 Autenticazione utente-computer

Può essere:

1. **Basata su conoscenza** di segreti prestabiliti e precondivisi (password, passphrase, PIN)
2. **Basata su possesso** di dispositivi magnetici o elettronici (carte magnetiche, smart card, smart token)
3. **Basata su biometria** di generiche caratteristiche fisiche dell'utente (impronte, iride, tono di voce)

Spesso usate combinazioni di criteri (bancomat).

6.2 Autenticazione basata su conoscenza

Conoscere la giusta password comprova l'identità, semplice ed economica da implementare. Ma corre rischi di:

- **Guessing**: indovinata (attacco standard, attacco dizionario, attacco forza bruta)
- **Snooping**: sbirciata mentre viene inserita
- **Spoofing**: scoperta tramite falsa interfaccia di login (trojan)
- **Sniffing**: intercettata durante la trasmissione

Osservazione: Chiunque conosca la password di un utente per un sistema può impersonare in toto quell'utente col sistema!

6.2.1 Attacchi

- **Attacco standard**: password brevi, tipiche, relative all'utente (hobby, nomi parenti, compleanno, indirizzi)
- **Attacco dizionario**: vengono provate tutte le parole di un dizionario. Tentativi spesso arricchiti con regole che ricalcano possibili scelte dell'utente (doppia parola, parola al contrario, 0 al posto di o, 1 al posto di i, ecc)
- **Attacco forza bruta**: vengono provate esaustivamente tutte le parole costruibili in un dato vocabolario (alfanumerico, simboli speciali) di lunghezza via via crescente. Empiricamente si vede che con un ricco vocabolario, 6 è una soglia notevole.

6.2.2 Contromisure

- **Controllo sulla password**: Il sistema controlla che la password non sia banale (lunghezza, caratteri speciali) quando essa viene scelta
- **Controllo sul numero di inserimenti**: Il sistema limita i tentativi di login, pensa blocco
- **Uso di CAPTCHA**: Completely Automated Public Turing Test To Tell Computers and Humans Apart. Il sistema richiede la risoluzione di una captcha per verificare che il tentativo di login viene da un umano. Un recente algoritmo basato su Google Streen View viola il 99% delle captcha alfanumeriche

6.2.3 Norme per una buona password

Bilanciare al meglio **mnemonicità** (sia facile da ricordare cosicché non serva scriverla) e **complessità** (sia robusta verso guessing).

Questa implica di non usare una parola del dizionario, usare almeno 8 caratteri (prove empiriche su pdf password cracking) e non usare la stessa password per autenticazioni diverse, altrimenti basterebbe una sola compromissione.

Mantenere una password

La password va mantenuta in qualche modo sul sistema al quale garantisce l'autenticazione l'utente. Come?

Tipicamente memorizzando in un database ciascuna password in una qualche forma.

Storia: CTSS

Compatible Time Sharing System, MIT, 1960.

Password memorizzate in chiaro su file di sistema protetto da politica di sicurezza. Circolarità dell'idea: in generale il controllo d'accesso basato su autenticazione, qui pure viceversa; numerosi attacchi registrati.

6.3 Autenticazione basata su possesso

Identità dell'utente comprovata dal possesso del giusto oggetto, tipicamente magnetico o elettronico.
L'oggetto può memorizzare l'informazione sensibile. L'informazione su carta magnetica è **interamente leggibile**, mentre l'informazione su carta elettronica è **leggibile coerentemente con interfaccia** funzionale. Le smartcard ormai sono abbastanza potenti anche se gli smart token sono più diffusi per autenticazioni sensibili (banca online, rete istituzionale)

Discussione

Il sistema ha dei limiti:
Il processo di autenticazione riconosce l'oggetto non l'utente, come nel caso della conoscenza. Quindi è più facile smarrire un oggetto che un segreto? La soluzione è la **two-factor authentication**: combinare autenticazione su conoscenza con autenticazione su possesso.

6.3.1 Smart token

Apprezzabili capacità computazionali. Autenticazione **ad esso** basata su conoscenza (PIN). Genera uno speciale segreto detto **one-time password (OTP)**, accettato solo una o poche volte dal server. Autenticazione **al server web** ora basata su possesso e conoscenza (come bancomat). Password o sul token o sul sito equivalente. Maggiore inclonabilità rispetto ad una smartcard.

Funzionamento di uno smart token

Chiave segreta (seme) memorizzata dalla fabbrica. Prende info esterne (PIN, ora) per generare la one-time password. La OTP viene visualizzata sul display, rinnovata frequentemente (ogni 30-90 secondi). Minimizza rischi di guessing (grazie alla generazione non umana) e di sniffing (grazie alla validità limitata). I token moderni utilizzano un chip TPM che consentono di mantenere un segreto in un chip in modo sicuro. C'è una sincronizzazione algoritmica tra server e token, non diretta. Un modo tipico per condividere la password col server si basa sull'uso di un algoritmo comune (con seme comune) e orologi sincronizzati.

In realtà l'autenticazione basata su possesso sarebbe stata aggiunta senza OTP, ovvero con un token che generi un password (lunga?) fissa, esattamente come fa una carta magnetica e un smartcard alla quale non venga richiesta computazione. Non confondere conoscenza con possesso: possesso di un oggetto tipicamente ovviabile con informazione che rappresenti l'oggetto (stampanti 3D). Il fatto che un token generi una OTP è pertanto un ulteriore miglioramento rispetto al bancomat.

Domanda: Perché una OTP è meglio di una password normale? Siccome viene accettata una sola volta, non c'è il tempo di effettuare un attacco. Se invece l'attacco fosse istantaneo, se il proprietario della password

ha già usato la password, l'attacco non riesce perché il server non l'accetta più. È la caratteristiche della password, che viene accettata una sola volta, che rende l'attacco istantaneo vano.

6.4 Autenticazione basata su biometria

Possesso di caratteristiche biometriche, che sono **univoche**, comprova identità: **fisiche** (impronte digitali, forma della mano, impronta della retina e del viso) e **comportamentali** (firma, timbro della voce, grafia, keystroke dynamics). Tecnicamente meno accurata dei primi due metodi ma comunque più affidabile. Due password possono essere confrontate e risultate identiche, due campioni biometrici praticamente mai.

Per ogni tecnica di autenticazione biometrica si parla dell'invasività. Es. scanner iride.
Quella meno invasiva è l'impronta digitale ma può essere persa. Se è facile e non invasiva come le impronte digitali non è robusta. Quelle più robuste sono invasive: es DNA.

Nuova tecnologia: **finger vein** (l'impronta delle vene di ogni individuo è univoca).

Se è possibile perdere la "password" anche biometrica: il sistema di autenticazione non è robusto.
Inoltre, per l'informazione biometrica non esiste un codice universale per rappresentarla nel mondo digitale.

L'autenticazione biometrica è meno accurata perché il confronto, ad esempio, tra le impronte digitali non è possibile farlo in modo preciso, ma si fa in modo approssimato. Non si può fare in modo preciso perché non esiste il codice per rappresentarlo in modo preciso.
Il più grande problema è la r**appresentazione digitale**. Motivo per cui il sensore fa più scansioni. Ciò non accade per lettere e numeri. Quindi se la stringa della password viene flippata di un bit il sistema non autentica. La rappresentazione è impeccabile, in quanto la rappresentazione è algoritmica. Con le impronte digitali, non è possibile, perché non so acquisire le impronte in maniera univoca. Il confronto non può essere fino all'ultimo bit, ma almeno di una tolleranza. Il che rende il processo meno preciso.

Genomics privacy
Un sistema più invasivo è più sicuro? In principio si, ma ad esempio col DNA, si avrebbe nel sistema un campione di tutto il DNA: non ci sarebbe più nulla da nascondere. Privacy? Rischi di privacy.

Funzionamento

Fase iniziale di campionamento. Si deve poter rappresentare sul pc il campione biometrico per poi poterlo confrontare live in fase di autenticazione. Esecuzione di più misurazioni sulla caratteristica di interesse. Definizione di un **template** che media le misurazioni e rappresenta la caratteristica.

Fase di autenticazione. Decisa dal confronto fra caratteristica appena misurata e template. Successo se i due corrispondono **a meno di una tolleranza prestabilita**.

Discussione

L'autenticazione biometrica, è davvero la più affidabile? Si può cedere o perdere una password o un oggetto. Si può cedere o perdere una caratteristica biometrica?

A volte è considerata pericolosa e intrusiva. Si discute se gli scanner di retina siano pericolosi per la retina. La caratteristica biometrica è per sempre, mentre una password persino one-time.

6.4.1 Impronte digitali

Righe su mani e piedi formate prima della nascita. Restano inalterate per tutta la vita dell'individuo (a meno di incidenti). Formano un disegno **unico** per ogni individuo. Uno dei metodi più antichi per riconoscere l'identità. Dall'inchiostro di una volta agli scanner digitali di oggi.

Classificazione delle impronte

Tre schemi predominanti:

- **Loop** (60%), linee circolari che escono verso l'esterno
- **Arch** (5%), linee poco circolari che escono

- **Whorl** (35%), linee circolari che non escono

Riconoscimento delle impronte

Necessità di algoritmi avanzati per il riconoscimento di immagini digitali. Questi puntano all'individuazione delle **minuzie**, ovvero la fine o il punto di biforcazione di una linea.

7 Altri protocolli di sicurezza

7.1 Roger Needham – Schroder

I precedenti protocolli hanno il problema del replay. Vediamo dei protocolli reali, non implementati.
Il seguente è l'articolo più citato nel settore. Protocollo del 1978 (Roger Needham – Schroder). Il protocollo presuppone una PKI con crittografia perfetta.

- $A \longrightarrow B \ : \ \{A, N_a\}_{k_b}$
- $B \longrightarrow A \ : \ \{N_a, N_b\}_{k_a}$
- $A \longrightarrow B \ : \ \{N_b\}_{k_b}$

Obiettivi di sicurezza del protocollo

1. Autenticazione reciproca degli utenti. Etichette mittente e ricevente inaffidabili. Autenticazione garantita da segretezza delle nonce.
2. Segretezza delle nonce scambiate

Il senso di questo protocollo è che nello schema DY, in presenza di un attaccante, vogliamo che A e B possano autenticarsi a vicenda senza la possibilità di attacchi di replay. Questo protocollo è basato su crittografia asimmetrica. Nel primo messaggio A inventa una nonce che associa con la propria identità e codifica con la chiave pubblica del destinatario. È importante il fattore chiave pubblica del destinatario. Alla recezione del messaggio (1), B lo decodifica con la propria chiave privata, nessun altro può decifrare il messaggio. Quindi B vede la nonce. Al primo passaggio non si è autenticato nessuno. Al (2), B inventa la propria nonce che codifica, insieme alla nonce di A, con la chiave pubblica di A. Notiamo come, questo protocollo può essere sfruttato per ddosare B, inviando ripetuti messaggi (1). B non ha ancora autenticato A. Nell'ipotesi in cui la crittografia sia perfetta, l'attaccante non potrà leggere il contenuto del messaggio (2). Ricevuto (2), A estrae la sua nonce, questo implica che A deduce che c'è stata un'estrazione del primo messaggio, possibile solo a B. In questo modo B si autentica con A. Al passo (3) avviene l'autenticazione di A con B per lo stesso motivo del passo (2). Il protocollo ci garantisce anche la segretezza della nonce.

Questo protocollo soffre di un attacco scoperto nel (1993), ovvero c'è almeno uno scenario in cui questa autenticazione fallisce. Quando fallisce la mutua autenticazione, l'attaccante riesce ad autenticarsi come A o B.

7.1.1 Attacco di Lowe (1995)

Il protocollo è detto **NSL** (Gavin Lowe).

1. $A \longrightarrow C \ : \ \{A, N_a\}_{k_c}$

1'. $C \longrightarrow B \ : \ \{A, N_a\}_{k_b}$

2. $B \longrightarrow C \ : \ \{N_a, N_b\}_{k_a}$

2'. $C \longrightarrow A \ : \ \{N_a, N_b\}_{k_a}$

3. $A \longrightarrow C \ : \ \{N_b\}_{k_c}$

3'. $C \longrightarrow B \ : \ \{N_b\}_{k_b}$

All'inizio A inizia il protocollo con C al passo (1). C ingaggia con B una nuova connessione, dove all'interno del messaggio inserisce i valori del massaggio inviato da A a C. B alla recezione del messaggio risponde come da protocollo ad A, in quanto non sa dell'esistenza di C che si è spacciato per A. Anche qualora B inviasse il messaggio a C, C potrebbe tranquillamente rimbalzare il messaggio verso di A. Alla recezione del messaggio, A invia a C la propria nonce Nb_kc. Sia A sia B non notano alcun alterità. In tutto questo C si autentica con B come A.

Analisi dell'attacco

- 2 sessioni interlacciate
- Ipotesi che A cominci con la spia.
- Attivi, da **posizione intermedia**: la segretezza di Nb fallisce al passo (3), autenticazione di A con B fallisce col passo (3').
- Sicurezza (segretezza, autenticazione) fallita anche nell'ipotesi di crittografia perfetta.

Conseguenze dell'attacco.
Se B fosse una banca e gli altri due correntisti...
Se A fosse il docente e gli altri due studenti...

La chiave di sessione è quella sulla base facciamo le autenticazioni, in quanto la usiamo per cifrare. Prima di aggiustare l'attacco, cambiamo il modello di attaccante (General Attacker).

7.1.2 Attacco di Lowe in General Attacker

In un contesto GA, non solo c'è C in mezzo, ma anche B si comporta da attaccante. In questo caso, i problemi sono due.

1. $A \longrightarrow C \ : \ \{A, N_a\}_{k_c}$
1'. $C \longrightarrow B \ : \ \{A, N_a\}_{k_b}$
2. $B \longrightarrow C \ : \ \{N_a, N_b\}_{k_a}$
2'. $C \longrightarrow A \ : \ \{N_a, N_b\}_{k_a}$
3. $A \longrightarrow C \ : \ \{N_b\}_{k_c}$
3'. $C \longrightarrow B \ : \ \{N_b\}_{k_b}$

I passi sono gli stessi, ma si ha il problema delle vendetta. Nell'ipotesi in cui B scopra l'importanza di N_a, se anche A è una banca, B può vendicarsi su C come segue:

5'. $\{N_a, N_b, \text{"Trasferisci 20000 euro dal conto di C al conto di B"}\}_{k_a}$

7.2 Woo-Lam (meta '80)

Protocollo che usa **crittografia simmetrica** e un **TTP** (Thrusted Third Party) che possiede un database di tutte le chiavi.
La crittografia è simmetrica, e quando la crittografia è simmetrica abbiamo il problema della chiave di sessione. Il problema, non è la chiave di sessione, ma di challenge response per l'autenticazione di A con B, motivo per cui entra in gioco il TTP. *Obiettivo*: autentica di A con B.

1. $A \longrightarrow B \ : \ A$
2. $B \longrightarrow A \ : \ N_b$
3. $A \longrightarrow B \ : \ \{N_b\}_{K_a}$
4. $B \longrightarrow TTP \ : \ \{A, \ \{N_b\}_{K_a}\}_{K_b}$
5. $TTP \longrightarrow B \ : \ \{N_b\}_{K_b}$

Il messaggio 5 è cifrato solo per distinguerlo con autenticazione dal messaggio 3. Perché essendo il messaggio 5 codificato con la chiave pubblica di B, per l'ipotesi fatta sulle chiavi, il messaggio 5 non può averlo che fatto TTP. Quindi (5) non solo è distinguibile dal (3), quindi bloccato l'attacco di replay, ma garantisce che il messaggio è stato inviato da TTP, altro possessore della chiave di B. B autentica A perché A è ipoteticamente il suo interlocutore, citato insieme nel messaggio (4) che inviata al TTP ben programmata che risolve questa associazione.

Anche questo protocollo ha un attacco.
B è vittimizzato nel credere che il suo interlocutore sia A, che invece è offline, da parte dell'attaccante C. L'attacco in questione è l'attacco di sessione parallela.

7.2.1 Attacco su Woo-Lam

1. $C \longrightarrow B \ : \ A$
1'. $C \longrightarrow B \ : \ C$
2. $B \longrightarrow A \ : \ N_b$
2'. $B \longrightarrow C \ : \ N_b{}'$
3. $C \longrightarrow B \ : \ \{N_b\}_{K_c}$
3'. $C \longrightarrow B \ : \ \{N_b\}_{K_c}$
4. $B \longrightarrow TTP \ : \ \{A, \ \{N_b\}_{K_c}\}_{K_b}$
4'. $B \longrightarrow TTP \ : \ \{C, \ \{N_b\}_{K_c}\}_{K_b}$
5. $TTP \longrightarrow B \ : \ \{N_b{}''\}_{K_b}$
5'. $TTP \longrightarrow B \ : \ \{N_b\}_{K_b}$

Il problema è che B non è in grado di distingue la sessione.

I rischi di attacco aumentano.

- 1978: Needham-Schroder, 6 pagine
- Metà anni '90: **SSL**, 80 pagine
- Fine anni '90: **SET**, 1000 pagine.

Quasi vent'anni per scoprire che un protocollo di 6 pagine celava un bug...

Quando fu trovata la vulnerabilità nel protocollo, i creatori si giustificarono che il protocollo non fosse pensato per quel modello di attaccante, bensì in un contesto in cui tutti si fidano di tutti. Tutti gli attacchi hanno un senso all'interno di un contesto di attaccanti. Se cambia il modello di attaccante, l'attacco perde senso o cambia.

7.3 Potenziali soluzioni

7.3.1 Needham-Schroder asimmetrico

Dobbiamo comunicare ad A nel messaggio (2) che qualcosa è andato storto rispetto alle sue supposizioni fino a quel momento. Potremmo dirglielo autenticando B. L'obiettivo del buono è farsi riconoscere, quindi scrive la propria identità B. La soluzione è:

$$2. \ B \longrightarrow A \ : \ \{Na, Nb, B\}_{k_a}$$

In questo modo, A alla recezione, riconosce che è presente B e quindi capisce che qualcosa è andata storto.

7.3.2 Woo-Lam

$$5. \ TTP \longrightarrow B \ : \ \{A, Nb\}_{k_b}$$

7.4 Principi di disegno: Explicitness

Definizione: Se le identità del mittente e del ricevente sono significative per il messaggio, allora è prudente menzionarle esplicitamente.

Symmetric Needham-Schroder

1. $A \longrightarrow TTP \ : \ A, B, N_a$
2. $TTP \longrightarrow A \ : \ \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_b}\}_{K_a}$
3. $A \longrightarrow B \ : \ \{K_{ab}, A\}_{K_b}$
4. $B \longrightarrow A \ : \ \{N_b\}_{K_{ab}}$
5. $A \longrightarrow B \ : \ \{{N_b}^{-1}\}_{K_{ab}}$

A cosa serve la nonce? Per capire a cosa serve, bisogna andare a vedere come viene utilizzata dopo essere stata inventata. Questa nonce viene restituita nel messaggio (2), quindi ha valore [...]

TTP inventa la chiave di sessione, che viene inserita nel messaggio (2) codificato in maniera robusta. Aggiunge anche B, ovvero chi è l'interlocutore con cui utilizzare questa chiave di sessione, in maniera esplicita. Se B non fosse nel messaggio 2, mancherebbe l'esplicitazione, quindi un attaccante avrebbe potuto sostituire B con un qualsiasi altro nome. Nel messaggio (2) c'è pure un ticket, cifrato con una chiave che A non conosce. Il ticket non è altro che la copia d.ella chiave di sessione per B. Questo non può essere mandato in chiaro, ma necessita cifratura. La chiave da utilizzare è ovviamente quella di B. Decifrando il messaggio (3) B apprende la chiave di sessione. In questo modo, sia A che B hanno ricevuto la chiave di sessione in maniera protetta.

Domanda: Perché il protocollo non finisce così?
(4) e (5) è l'handshake, dove avviene la mutua autenticazione tra A e B. Il -1 è un tentativo di differenziare il messaggio (5) dal (4).

Se l'attaccante scoprisse una chiave di sessione vecchia, si potrebbe autenticare come A presso B.
Distruggere completamente un segreto dal disco, non è tanto facile. È questa la limitazione del protocollo.

7.5 Attacco di replica

Definizione: Spacciare informazione (chiavi,...) obsoleta, magari violate, come recente.

Supponiamo che C abbia violato una vecchia chiave di sessione K_{ab} che B condivise con A.

...

3. $\mathbf{C} \longrightarrow B \ : \ \{K_{ab}, A\}_{K_b}$ (rispedito identico)
4. $B \longrightarrow A \ : \ \{{N_b}''\}_{K_{ab}}$ (intercettato)
5. $\mathbf{C} \longrightarrow B \ : \ \{{N_b}' - 1\}_{K_ab}$

B come autenticherebbe A e quindi accetterebbe di usare K_{ab}.

Timestamp

Se il ticket portasse con se la data di produzione, chi lo legge può dire se questo è ancora valido o no.
Dovremmo modificare il protocollo, dando a B una qualche verifica di freshness sulla chiave di sessione.
(4) e (5) conferiscono a B garanzia che dall'altra parte ci sia di recente qualcuno che possiede K_{ab}, non meglio identificato. Non è dato sapere a B quanto tempo fa è stata creata Kab.

8 SSL

Lo scopo di SSL è mettere in sicurezza il www. www è uno dei servizi di internet, non internet. Per mettere in sicurezza il www abbiamo HTTPS che è la versione di HTTP che incapsula SSL. Così come SLL viene usato dentro HTTPS, può essere utilizzato per mettere in sicurezza la posta o i pagamenti.

Dove sta SSL? Non lo vediamo esplicitamente, ma leggiamo soltanto la "S" dopo HTTPS.

HTTPS (HTTP su TLS)

Banalmente va sulla porta 443, invece che la 80 come HTTP. Cifra tutto, anche i cookie perché tante volte registrano informazioni importanti. Dal punto di vista architetturale, SSL (TLS), sta a metà tra TCP e le applicazioni.

SSL vs TLS

SSL nasce nei primi anni 90 con Netscape, che sviluppa SSL. Nel 1994 nasce la versione 2. Nel 1995 nasce la versione 3 con un RFC (del 2011).
Nel 1999 l'IETF standardizza il protocollo. Netscape di fatto svanisce, e quindi SSL soccombe a favore di TLS 1.0. La prima versione era di fatto analoga a SSL 3.0.

8.1 Architettura SSL

SSL è progettato per uso del protocollo TCP, al fine di fornire un servizio di sicurezza end-to-end affidabile. SSL non è un unico protocollo, ma è piuttosto costruito da due livelli di protocolli.

Il protocollo SSL Record fornisce servizi di sicurezza di base a un certo numero di protocolli di più alto livello. In particolare, il protocollo HTTP che fornisce il servizio di trasferimento per le interazioni client/server web, può operare sopra SSL. SSL contiene tre protocolli di più alto livello: il **protocollo Handshake**, il **protocollo Change cipher** e il **protocollo Alert**. Questi protocolli specifici di SSL vengono utilizzati nella gestione degli scambi SSL e sono presi in esame più avanti in questo paragrafo. Due concetti importanti di SSL sono quello di *connessione SSL* e di *sessione*.

- **Connessione**. Una connessione è un trasporto (nella definizione di modello a strati OSI) che fornisce una tipologia opportuna di servizio. Per SSL, queste connessioni sono relazioni tra entità paritarie (*peer-to-peer*). Le connessioni sono transitorie. Ogni connessione è associata ad una sessione.
- **Sessione**. Una sessione SSL è un'associazione fra un client e un server. Le sessioni sono create dal protocollo Handshake e definiscono un insieme di parametri crittografici di sicurezza, che possono essere condivisi fra molteplici connessioni. Le sessioni si utilizzano per evitare una costosa negoziazione di nuovi parametri di sicurezza per ciascuno connessione.

Ci possono essere molteplici connessioni sicure fra una qualunque coppia di parti (applicazione come HTTP su clinet server). In teoria ci potrebbero essere anche molteplici sessioni simultanee fra due parti, ma questa funzionalità non è utilizzata nella pratica.
A ciascuna sessione è associato un certo numero di stati. Una volta instaurata una sessione, c'è uno stato operativo corrente sia per la lettura sia per la scrittura (per esempio, ricezione e trasmissione). Inoltre, durante il protocollo Handshake vengono creati gli stati di attesa (*pending*) per lettura e scrittura. Alla positiva conclusione del protocollo Handshake, gli stati di attesa diventano stati correnti.

Lo stato di sessione è definito dai seguenti parametri:

- **Identificatore di sessione** (*sessione identifier*): sequenza arbitraria di byte scelta dal server per identificare uno stato di sessione attivo o riattivabile.
- **Certificato dell'entità prioritaria** (*peer certificate*): certificato X509.v3 dell'entità prioritaria. Questo elemento dello stato può essere nullo.
- **Metodo di compressione** (*compression method*): algoritmo usato per la compressione dei dati prima della cifratura.
- **Specifica del cifrario** (*chipher spec*): specifica l'algoritmo di cifratura dei dati che viene utilizzato - come, ad esempio, nessun algoritmo (null), AES, ecc - e l'algoritmo hash (ad esempio MD5 o SHA-1) utilizzato per il calcolo del MAC. Definisce anche gli attributi crittografici come, ad esempio, la dimensione del codice hash (hash_size).
- **Valore segreto principale** (*master secret*): valore segreto a 48 byte condiviso fra il clinet e il server.
- **È riattivabile** (*is resumable*): flag che indica se la sessione può essere usate per instaurare nuove connessioni.

Uno stato di connessione è definito dai seguenti parametri:

- **Valore casuale di server e client** (*server and client random*): sequenze di byte scelte dal server e dal client per ciascuna connessione.
- **MAC segreto del server per la scrittura** (*server write MAC secret*): chiave segreta utilizzata nelle operazioni MAX sui dati inviati dal server.
- **MAC segreto del client per la scrittura** (*client write MAC secret*): chiave segreta utilizzata nelle operazioni MAX sui dati inviati dal client.
- **Chiave di scrittura del server** (*server write key*): chiave di cifratura convenzionale per i dati cifrati dal server e decifrati dal client.
- **Chiave di scrittura del client** (*client write key*): chiave di cifratura convenzionale per i dati cifrati dal client e decifrati dal server.
- **Vettori di inizializzazione** (*initialization vectors*): quanto si utilizza un cifrario a blocchi in modalità CBC, viene mantenuto un vettore di inizializzazione (IV) per ciascuna chiave. Tale campo è dapprima inizializzato dal protocollo di Handshake di SSL. Successivamente, l'ultimo blocco di testo cifrato di ciascun record viene mantenuto per essere utilizzato come IV per il record successivo.
- **Numeri di sequenza** (*sequence numbers*): ciascuna parte mantiene, per ogni connessione, numeri di sequenza separati per i messaggi trasmessi e ricevuti. Quando una parte invia o riceve un messaggio change chiper spec per la modifica della specifica del cifrario, si pone a zero il numero di sequenza appropriato. I numeri di sequenza non possono superare il valore di $2^64 - 1$.

8.2 Protocollo Record di SSL

Il protocolli Record di SSL fornisce due servizi per le connessioni SSL, grazie al protocollo Handshake.

- **Riservatezza**: il protocollo Handshake definisce una chiave segreta condivisa, che viene usate per la cifratura convenzionale del payload SSL.
- **Integrità dei messaggi**: Il protocollo Handshake definisce anche una chiave segreta, che viene usata per creare il codice di autenticazione del messaggio (MAC).

Il protocollo Record prende un messaggio applicativo da trasmettere, effettua la frammentazione dei dati in blocchi più maneggevoli, eventualmente esegue la compressione dei dati, applica il segmento MAC, esegue la cifratura, aggiunge un'intestazione e trasmette il messaggio risultante in un segmento TCP. la chiave segreta condivisa utilizzata per la cifratura convenzionale dei payload di SSL e quella utilizzata per creare il codice di autenticazione del messaggio (MAC) sono definite tramite il protocollo Handshake. I dati ricevuti sono decifrati, verificati, decompressi, riassemblati e infine consegnati agli utenti a livello più alto.

Il primo passo è la **frammentazione**. Ogni messaggio dei livelli sovrastanti viene frammentato in blocchi di dimensione 2^14 byte (16348 byte) o di dimensione inferiore. Successivamente e facoltativamente, si applica la compressione. La compressione deve essere senza perdita e non dovrebbe aumentare la lunghezza del contenuto di più di 1024 byte. In SSLv3 (come anche nella versione corrente di TLS), non è specificato alcun algoritmo di compressione, quindi il valore di default per l'algoritmo di compressione è *null*.

Il passo di elaborazione successivo consiste nel calcolare il MAC sui dati compressi. A tale scopo viene utilizzata una chiave segreta condivisa. Il calcolo del MAC è definito come segue:

```
hash(MAC_write_secret || pad_2 ||
      hash(MAC_write_secret || pad_1 || seg_num ||
      SSLCompressed.type ||
      SSLCompressed.length || SSLCompressed.fragment)
```

dove:

`||` è l'operatore di concatenazione

`MAC_write_secret` è la chiave segreta condivisa

`hash` è l'algoritmo hash crittografico (MD5 o SHA-1)

`pad_1` è il byte 0x36 (0011 0110) ripetuto 48 volte (348 bit) per MD5 e 40 volte per SHA-1

`pad_2` è è il byte 0x5C (0101 1100) ripetuto 48 volte (348 bit) per MD5 e 40 volte per SHA-1

`seq_num` è il numero di sequenza per il messaggio considerato

`SSLCompressed.type` è il protocollo di livello più alto utilizzato per elaborare il frammento considerato

`SSLCompressed.length` è la lunghezza del frammento compresso

`SSLCompressed.fragment` è il frammento compresso (se non si utilizza la compressione, il frammento in chiaro

Come si può notare, la differenza tra l'algoritmo appena descritto è l'algoritmo HMAC è che nell'algoritmo di SSLv3 i due pad sono concatenati mentre in HMAC sono combinati mediante OR esclusivo.

Il messaggio compresso e il MAC son quindi **cifrati** attraverso cifratura simmetrica.

Gli algoritmi di cifratura consentiti sono: AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza.

L'ultimo passo dell'elaborazione del protocollo Record di SSL consiste nell'apportare un'intestazione composta dai seguenti campi:

- **Tipo di contenuto, 8 bit** (*content type*): il protocolli di livello più alto utilizzato per elaborare il frammento accluso.
- **Versione principale, 8 bit** (*major version*): indica la versione principale di SSL in uso. Nel caso di SSLv3, il valore è 3.
- **Versione minore, 8 bit** (*minor version*): indica la versione minore di SSL in uso. Nel caso di SSLv3, il valore è 0.
- **Lunghezza compressa, 16 bit** (*compressed length*): lunghezza in byte del frammento in chiaro (o del frammento compresso se si usa la compressione). Il valore massimo è $2^14 + 2048$.

I tipi di contenuto definiti sono `change_chiper`, `alert`, `handshale` e `application_data`. I primi tre sono i protocolli specifici di SSL, descritti in seguito. Non c'è distinzione fra le diverse applicazioni che possono utilizzare SSL (per esempio, HTTP); il contenuto dei dati generati da queste applicazioni non è noto ad SSL.

8.3 Protocollo Change chiper spec

Il protocollo Change chiper spec è il più semplice fra i tre specifici protocolli di SSl, che utilizzano il protocollo Record di SSL. Consiste in un singolo messaggio, che a sua volta consiste in un solo byte con valore 1. L'unico scopo di questo messaggio è quello di forza la copiatura dello stato di attesa nello stato corrente, aggiornando la suite di cifratura da usare nella connessione corrente.

8.4 Protocollo Alert

Il protocollo Alert è utilizzato per comunicare all'unità prioritaria messaggi di allarme relativi a SSL. Come per le altre applicazioni che usano SSL, i messaggi di allarme sono compressi e cifrati, come specificato nello stato corrente.
Ogni messaggio in questo protocollo è composto da due byte. Il primo byte assume il valore warning (1) o fatal (2) per trasmettere la gravità del messaggio. Se il livello è fatal, SSL termina immediatamente la connessione. Le altre connessioni nella stessa sessione possono continuare, ma non si possono stabilire nuove connessioni nella sessione in questione. Il secondo byte contiene un codice che denota lo specifico allarme. Prima di tutto, vengono elencati gli allarmi che sono sempre fatali (le definizioni sono tratte dalla specifica SSL).

8.5 Protocollo Handshake

Il protocollo Handshake è la parte più complessa di SSL. Questo protocollo consente al client e al server di autenticasi a vicenda e di negoziare un algoritmo di cifratura e di MAC e le chiavi di cifratura da utilizzare per la protezione dei dati inviati da un record SSL.

Il protocollo Handshake è utilizzato prima di spedire qualunque dato applicativo. Consiste in una serie di messaggi scambiati fra client e server. Ciascun messaggio ha tre campi:

- **Tipo, 1 byte** (*type*): indica un tipo fra 10 possibili tipi di messaggio.
- **Lunghezza, 3 byte** (*length*): lunghezza del messaggio in byte.
- **Contenuto, $\geqslant$ 0 byte** (*content*): parametri associati con il messaggio in questione.

Tipo di messaggio	Parametri
`hello_request`	Nessun parametro
`client_hello`	Versione, random, ID di sessione, suite di cifratura, metodo di compressione
`server_hello`	Versione, random, ID di sessione, suite di cifratura, metodo di compressione
`certificate`	Catena di certificati X.509v3
`server_key_exchange`	Parametri, firma
`certificate_request`	Tipo, autorità
`server_clone`	Nessun parametro
`certificate_verify`	Firma
`client_key_exchange`	Parametri firma
`finished`	Valore hash

8.5.1 Fase 1

Stabilire le capacità di sicurezza, includendo versione del protocollo, ID di sessione, suite di cifratura, metodo di compressione e numeri casuali iniziali.

1. Client $\longrightarrow$ Server : `client_hello`
2. Server $\longrightarrow$ Client : `server_hello`

Questa fase è usata per instaurare una connessione logica e per stabilire le capacità di sicurezza che stanno ad essa associate. Il client inizia lo scambio inviando un messaggio `client_hello` con i seguenti parametri:

- **Versione** (*version*): la versione più elevata di SSL supportata dal client.
- **Random**: struttura casuale generata dal client, che consiste in un timestamp a 32 bit e in 28 byte prodotti da un generatore di numeri casuali sicuro. Tali valori servono come nonce e sono utilizzati durante lo scambio delle chiavi per la prevenzioni di attacchi di replay.
- **ID di sessione** (*session ID*): identificatore di sessione a lunghezza variabile. Un valore diverso da zero denota che il client vuole aggiornare i parametri di una connessione esistente o creare una nuova connessione nella sessione attuale. Un valore zero indica che il client vuole instaurare una nuova connessione in una nuova sessione.
- **Suite di cifratura** (*CipherSuite*): lista contenente le combinazioni di algoritmi crittografici supportati dal client, in ordine decrescente di preferenza. Ciascun elemento della lista (ovvero ciascuna suite di cifratura) definisce sia l'algoritmo per lo scambio delle chiavi sia una ChiperSpec.
- **Metodo di compressione** (*compression method*): lista dei metodi di compressioni supportati dal client.

Dopo aver inviato il messaggio di `client_hello`, il client attende il messaggio `server_hello`, contenente gli stessi parametri del messaggio `client_hello`. Al messaggio `server_hello` si applicano le seguenti convenzioni. Il campo Versione contiene il valore di versione più basso suggerito dal client e il più elevato supportato dal server. Il campo Random viene generato dal server ed è indipendente dal campo Random del client. Se il campo ID di sessione del client era diverso da zero, lo stesso valore è usato dal server, altrimenti il campo ID di sessione del server contiene il valore di una nuova sessione. Il campo Suite di cifratura contiene una specifica suite di cifratura scelta dal server fra quelle proposte dal client. Il campo Compressione contiene il metodo di compressione scelto dal server fra quelli proposti dal client.

Il primo elemento del paragrafo Suite di cifratura è il metodo di scambio di chiavi. I metodi di scambio delle chiavi supportati sono i seguenti:

- **RSA**. La chiave segreta è cifrata con la chiave pubblica RSA del ricevente. Deve essere reso disponibile un certificato a chiave pubblica per la chiave del ricevente.
- **Fixed Diffie-Hellman**. Algoritmo di scambio di chiavi Diffie-Hellman in cui il certificato del server contiene i parametri pubblici Diffie-Hellman firmati dalla CA, cioè il certificato a chiave pubblica contiene i parametri della chiave pubblica di Diffie-Hellman. Il client fornisce i propri parametri di chiave pubblica Diffie-Hellman all'interno di un certificato se è richiesta l'autenticazione del client, oppure in un messaggio di scambio di chiavi. Questo metodo porta ad avere una chiave segreta stabilita fra due parti, basata sul calcolo Diffie-Hellman utilizzando le chiavi pubbliche fissate.
- **Ephemeral Diffie-Hellman**. Tecnica utilizzata per generare chiavi segrete temporanee (*one-time*). In questo caso, vengono scambiate le chiavi pubbliche Diffie-Hellman firmate con la chiave privata RSA o DSS del mittente. Il ricevente può utilizzare la chiave pubblica corrispondente per verificarne la firma. I certificati sono utilizzati per autenticare le chiavi pubbliche. Questa sembrerebbe la più sicura delle tre opzioni Diffie-Hellman in quanto genera un chiave temporanea autenticata.
- **Anonymous Diffie-Hellman**. Utilizza l'algoritmo Diffie-Hellman di base, senza autenticazione. Ovvero, ciascuna parte invia i propri parametri pubblici dall'altra parte, senza autenticazione. Questo approccio è vulnerabile rispetto ad attacchi di tipo MITM.
- **Fortezza**. Tecnica definita per lo schema Fortezza.

La definizione del metodo per lo scambio delle chiavi è la ChiperSpec che comprende i seguenti campi:

- **ChiperAlgorithm** (*algoritmo di cifratura*): qualunque algoritmo precedentemente menzionato (RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza).
- **MACAlgorithm** (*algoritmo MAC*): MD5 o SHA-1.
- **IsExportable** (*è esportabile*): vero o falso.
- **HashSize** (*dimensione hash*): 0,16 (per MD5) o 20 (per SHA-1) byte.
- **Key Material** (*dati per le chiavi*): sequenza di byte contenente i dati usati nella generazione della chiavi di scrittura.
- **IV Size** (*dimensione di IV*): dimensione del vettore di inizializzazione per la cifratura in modalita CBC.

8.5.2 Fase 2

Il server può inviare un certificato, scambio di chiavi, e una richiesta di certificato. Il server segnala la fine della fase hello.

1. Server $\longrightarrow$ Client : `certificate`
2. Server $\longrightarrow$ Client : `server_key_exchange`
3. Server $\longrightarrow$ Client : `certificate_request`
4. Server $\longrightarrow$ Client : `server_hello_done`

8.5.3 Fase 3

Il client invia un certificato, se richiesto. Il client invia lo scambio di chiavi. Il client può inviare la verifica del certificato.

1. Client $\longrightarrow$ Server : `certificate`
2. Client $\longrightarrow$ Server : `client_key_exchange`
3. Client $\longrightarrow$ Server : `certificate_verify`

8.5.4 Fase 4

Suite change cipher e conclusione del protocollo Handshake.

1. Client $\longrightarrow$ Server : `change_cipher_spec`
2. Client $\longrightarrow$ Server : `finished`
3. Server $\longrightarrow$ Client : `change_cipher_spec`
4. Server $\longrightarrow$ Client : `finished`

8.5.5 Creazione del Master Secret

Il valore master segreto condiviso è un valore one-time di 48byte (348 bit) generato per la sessione corrente mediante scambio sicuro di chiavi. La creazione avviene in due fasi. Prima di tutto viene scambiato un valore `pre_master_secret`; poi entrambe le parti calcolano il valore `master_secret`. Per lo scambio del valore `pre_master_secret` esistono due possibilità:

- **RSA**. Il client genera un valore `pre_master_secret` di 48 byte, cifrato con la chiave pubblica RSA del server, e lo invia al server. Il server decifra il testo cifrato ricevuto utilizzando la propria chiave privata per ricostruire il valore `pre_master_secret`.
- **Diffie-Hellman**. Sia il client che il server generano una chiave pubblica Diffie-Hellman. Dopo aver scambiato tali chiavi, ciascuna parte esegue il calcolo Diffie-Hellman per creare il valore condiviso `pre_master_secret`.

Entrambe le parti calcolano il valore `master_secret` come segue:

```
master_secret = MD5(pre_master_secret || SHA('A' ||
                    pre_master_secret || ClientHello.random ||
                    ServerHello.random)) ||
                MD5(pre_master_secret || SHA('BB' ||
                    pre_master_secret || ClientHello.random ||
                    ServerHello.random)) ||
                MD5(pre_master_secret || SHA('CCC' ||
                    pre_master_secret || ClientHello.random ||
                    ServerHello.random)) ||
```

Dove ClientHello.random e ServerHello.random sono i due nonce scambiati nei messaggi hello iniziali.

8.5.6 Generazione dei parametri di crittografia (key blocks)

Le ChiperSpec richiedono un valore segreto di MAC di scrittura, una chiave di scrittura e un vettore IV di scrittura sia per il client che per il server, generati in tale ordine a partire dal valore master segreto. Questi parametri sono generati dal valore master segreto trasformandolo mediane un algoritmo hash in una sequenza di byte di lunghezza sufficiente a contenere tutti i parametri.

La generazione delle chiavi a partire dal valore master segreto utilizza lo stesso formato di generazione usato dal valore master segreto a partire dal valore segreto premaster:

```
key_block = MD5(master_secret || SHA('A' || master_secret ||
                    ClientHello.random || ServerHello.random)) ||
                MD5(master_secret || SHA('BB' || master_secret ||
                    ClientHello.random || ServerHello.random)) ||
                MD5(master_secret || SHA('CCC' || master_secret ||
                    ClientHello.random || ServerHello.random))
                    ...
```

finché viene generato un risultato di lunghezza sufficiente. Il risultato di questa struttura algoritmica è una funzione pseudocasuale. Si può considerare il valore `master_secret` come valore seme pseudocasuale per la funzione. I numeri casuali del client e del server possono essere considerati come valori salt per rendere più difficile la crittoanalisi.

8.6 TLS

TLS è un'iniziativa di standardizzazione IEFT il cui obiettivo è quello di produrre una versione di Internet standard di SSL. La versione corrente di Prosposed standard di TLS, definita nel documento RFC 2246, è molto simile a SSLv3. In questo parametro vengono evidenziate le differenze.

Numero di versione

Il formato record di TLS è lo stesso di SSL e i campi nell'intestazione hanno lo stesso significato. La sola differenza risiede nei valori del campo versione. Nella versione corrente di TLS, la versione principale corrisponde alle 3 e la versione minore a 1.

8.6.1 MAC

Ci sono due differenze tra gli schemi MAC di SSLv3 e TLS: l'algoritmo utilizzato e l'ambito dell'elaborazione MAC. TLS utilizza l'algoritmo HMAC definito nel documento RFC 2104. HMAC è definito come segue:

$$\text{HMAC}_K(M) = H[(K^+ \oplus \text{opad}) \,||\, H[(K^+ \oplus \text{ipad}) \,||\, M]]$$

dove:

H è la funzione hash contenuta (per TLS, MD5 o SHA1)

M è il messaggio di ingresso a HMAC

K^+ è la **chiave segreta** completata con un serie di zero aggiuntivi a sinistra per produrre un risultato uguale alla lunghezza del blocco utilizzato nella codifica hash (per MD5 o SHA-1, la lunghezza del blocco è 512 bit)

ipad = 00110110 (36 in esadecimale) ripetuto 64 volte (512 bit)

opda = 01011100 (5C in esadecimale) ripetuto 64 volte (512 bit)

SSLv3 usa lo stesso algoritmo ad eccezione del fatto che i byte di completamento sono concatenati con la chiave segreta piuttosto che essere combinati mediante OR esclusivo con la chiave segreta, a sua volta completata per raggiungere la necessaria lunghezza del blocco. Il livello di sicurezza dovrebbe essere lo stesso in entrambi i casi.

Per TLS, il calcolo del MAC comprende tutti i campi coperti dal corrispondente calcolo di SSLv3, con in più il campo `TLSCompressed.version`, che indica la versione del protocollo in uso.

8.7 Funzione pseudocasuale (PRF)

TLS fa uso di una funzione pseudocasuale conosciuta come **PRF** (*pseudorandom function*) per espandere i valori segreti in blocchi di dati per la generazione o validazione delle chiavi. L'obiettivo è utilizzare un valore segreto condiviso relativamente piccolo ma generare blocchi di dati più lunghi, in modo che l'approccio sia sicuro rispetto alle tipologie di attacchi preparati a funzioni hash e MAC. La funzione PRF è basta sulla seguente funzione di espansione dei dati:

```
P_hash(valore segreto, seme) = HMAC_hash(valore segreto, A(1) || seme) ||
                               HMAC_hash(valore segreto, A(2) || seme) ||
                               HMAC_hash(valore segreto, A(3) || seme) ||
                               ...
```

Dove `A()` è definita come:

`A(0)` = seme

`A(`i`)` = `HMAC_hash (valore segreto, A(`i`-1)`

La funzione di espansione dei dati usa l'algoritmo HMAC, con MD5 o SHA1 come funzioni hash sottostanti. Come si può notare, `P_hash` può essere iterata tante volta quante sono quelle necessarie a produrre la quantità di dati richiesta. Ad esempio, se si fosse utilizzata `P_SHA-1` per generare 64 byte di dati, sarebbe stata iterata quattro volte, producendo 80 byte di dati, di cui gli ultimi 16 non sarebbero stati considerati. In questo caso, anche `P_MD5` sarebbe stata iterata quattro volte, producendo esattamente 64 byte di dati. Si noti

che ogni iterazione richiede due esecuzioni di HMAC, ciascuna delle quali richiede a sua volta due esecuzioni degli algoritmi sottostanti.

Per aumentare il livello di sicurezza della funzione PRF, si usano due algoritmi hash in modo che se uno dei due algoritmi rimane sicuro, la funzione dovrebbe essere sicura. PRF è definita come:

```
PRF(valore segreto, etichetta, seme) = P_MD5(S1, etichetta || seme) ⊕
                                       P_SHA1(S2, etichetta || seme)
```

Dove:

seme è la concatenazione di entrambi i valore Random

valore segreto è il Pre Master Secret

etichetta è il "master secret"

PRF riceve in ingresso un valore segreto, un'etichetta identificativa e un valore di seme e produce un risultato di lunghezza arbitraria. Il risultato è creato dividendo il valore segreto in parti uguali (S1 e S22) ed eseguendo la funzione P_hash su ciascuna di esse, utilizzando MD5 su una metà e SHA1 sull'altra. I due risultati sono combinati mediante OR esclusivo per produrre il risultato finale; a tale scopo, P_MD5 deve essere iterata generalmente più volte di P_SHA1 per produrre una pari quantità di dati in ingresso alla funzione OR esclusivo.

8.8 Evoluzioni di TLS

- **TLS 1.0** = SSL 3.1, RFC2246, 1999
- **TLS 1.1** = SSL 3.2, RFC4346, 2006
- TLS Extensions, RFC4366, 2006 per esempio Heartbeat per:
  - Generare attività durante i tempi di attesa
  - Controllare se il peer sia attivo
  - Evitare chiusure di connessione
- **TLS 1.2** = SSL 3.3, RFC5246, 2008
  "TLS allows extension to follow the compression_methods field in an extensions block. The presence of extensions can be detected by determining wheter there are bytes following the compression_methods at the end of the ClientHello"

TLS 1.1

Nessun cambiamento significativo rispetto a TLS 1.0

TLS 1.2

Agosto 2004 viene scoperto che MD5 soffre di collisioni

Dicembre 2008 Sotirov-Stevens exploit MD5 per creare false CA

Agosto 2008 dichiarato RFC5246

- *"Substantial cleanup to the clinet's and server's ability to specify which hash and signature algorithms the accept."*

TLS evolutions

Cambi significativi rispetto a TLS 1.0.

Agosto 2008 dichiarato RFC5246

- *"Substantial cleanup to the clinet's and server's ability to specify which hash and signature algorithms the accept."*

- *"The MD5/SHA-1 combination in the digitally-signed element has been replaced with s single hash. Signed elements now include a field that explicitly specifies the hash algorithm used."*
- *"The MD5/SHA-1 combination in the pseudorandom funcion (PRF) has been replaced with cipher-suite-specified PRF."*
- *"All Cipher Suites in this document use P_SHA256."*

Finalmente l'hash è "serio" ma è il 2008!

Giorni nostri

Giugno 2015 IETF depreca SSL 3.0 in RFC7568

Giugno 2016 Google smette di usare RC4 e SSL 3.0

Giugno 2016 TLS 1.3 diventa una bozza di lavoro del IETF

Giugno 2017 I certificati di SHA-1 diventano deprecati

Settembre 2017 Google diffida di Symantec

Giugno 2015 IETF approva TLS 1.3 come standard di Internet

Agosto 2018 TLS 1.3 viene rilasciato come RFC8446

TLS 1.3: più sicurezza

Rimuove le caratteristiche obsolete e insicure come:

- SHA-1
- RC4
- DES
- 3DES
- AES-CBC
- MD5
- Arbitrary Diffie-Hellmann (CVE-2016-0701)

Più efficienza, handshake più breve. Ci sono meno round e più messaggi vengono mandati tutti in una volta.

8.9 Heartbleed (CVE-2014-0160)

Sfrutta le estensioni Heartbeat e affligge molte librerie che usano OpenSSL. L'attacco sfrutta un problema di buffer over-read ed affligge sia server che client (reverse Heartbleed). Porzioni di memoria locale possono essere lette maliziosamente day peers, con potenziale divulgazione di segreti.

9 Voto elettronico: tre importanti proprietà

- **Cast as intended** (il voto sia posto come l'elettore voleva porlo), nel caso tradizionale della carta si traduce ad avere una scheda elettorale chiara e l'elettore capace di intendere e di volere.
- **Counted as cast** (valutato per come è stato messo). Nel caso elettronico serve un sistema che garantisca che il voto sia valutato correttamente.
- **Tallied as counted** (valutazione complessiva coerente alla conta), ulteriore step di conta.

10 Sicurezza IP

La comunità di Internet ha sviluppato meccanismi di sicurezza specifici per parecchie applicazioni, compresa la posta elettronica (S/MIME, PGP) le applicazioni client-server (Kerberos), l'accesso Web e altre. Tuttavia, gli utenti hanno problemi di sicurezza che sono trasversali a livelli protocollari. Ad esempio, un'azienda può avere una rete privata TCP/IP sicura, non consentendo collegamenti a siti non fidati, cifrando i pacchetti in uscita dalla rete privata e autenticando quelli in entrata.

Un'organizzazione che realizza sicurezza a livello IP, può, quindi, garantire un utilizzo sicuro della rete sia ad applicazioni dotate di meccanismi di sicurezza sia alle numerose applicazioni che non ne dispongono. La sicurezza a livello IP comprende tre funzionali aree: **autenticazione**, **riservatezza** e **gestione delle chiavi**. Il meccanismo di autenticazione assicura, infatti, che ogni pacchetto ricevuto sia stato effettivamente trasmesso dalla parte che nell'intestazione del pacchetto è identificata come sorgente; assicura, inoltre, che il pacchetto non sia stato modificato durante la trasmissione. Il meccanismo che garantisce la riservatezza consente ai nodi in comunicazione di cifrare i messaggi in modo da evitare che vengano intercettati da terze parti. La funzionalità di gestione delle chiavi provvede ad assicurare lo scambio sicuro delle chiavi.

10.1 Sicurezza a livello di rete

Per comunicare mediante un protocollo di sicurezza, tutte le macchine devono eseguire un client per quel protocollo. Se ne può fare a meno se garantiamo sicurezza ad un livello più basso. Tutte le "vecchie" applicazioni distribuite diventerebbero automaticamente "sicure".

Precisamente, le proprietà di sicurezza possono essere aggiunte a vari livelli nello stack TCP/IP. Più in basso andiamo, più la sicurezza diventa trasparente e rigida. Viceversa, più alto andiamo, più la sicurezza necessità un'apposita gestione e quindi concede più flessibilità.

Pro

- Le applicazioni possono rimanere "ignoranti" e sostanzialmente anche gli utenti.

Contro

- Appesantisce notevolmente la comunicazione al punto da facilitare DoS.
- Può richiedere modifiche al Sistema Operativo.

10.2 Sicurezza a livello di trasporto

Pro

- In realtà va o al trasporto o alle applicazioni.
- I principali browser contengono un client TLS

Contro

- Ovviamente richiede modifiche (minime) al livello che la riceve

10.3 Sicurezza a livello applicazione

Pro

- Sicurezza personalizzabile delle applicazioni

Contro

- Disegnare sicurezza a questo livello può essere ingannevole.

10.4 IPSec

Se IPSec è alternativo ad IP, suite di protocolli (**AH**, **ESP**, **IKE**) per garantire segretezza autenticazione e integrità a livello IP. Opzionale per Ipv4, mandatorio per Ipv6. Ogni pacchetto IP è arricchito con componenti crittografiche per la sicurezza. Non tutti i nodi della rete devono essere compatibili con IPSec.

Tali funzionalità di sicurezza sono realizzate tramite intestazioni aggiuntive, chiamate intestazioni di estensione, che seguono l'intestazione IP principale. L'intestazione di estensione relativa all'autenticazione è nota con il nome di **intestazione di autenticazione** (AH, *authentication header*) mentre quella per la cifratura è conosciuta come **ESP** (*encapsulating security payload*).

- **Architettura**: riguarda i concetti generali, le definizioni e i meccanismi alla base della tecnologia IPSec.
- **ESP** (*encapsulating security payload*): riguarda il formato dei pacchetti e questioni di carattere generale connesse all'utilizzo di ESP per la cifratura dei pacchetti e, opzionalmente, per l'autenticazione.
- `Intestazione di autenticazione` (AH, *authentication header*): copra argomenti riguardanti il formato dei pacchetti e questione di carattere generale connesse all'utilizzo di AH per l'autenticazione dei pacchetti.
- **Algoritmo di cifratura**: insieme di documenti che descrivono una serie di algoritmi di cifratura sono utilizzati per ESP.
- **Algoritmo di autenticazione**: insieme di documenti che descrivono come una serie di algoritmi di autenticazione sono utilizzati per AH e per l'opzione di autenticazione di ESP.
- **Gestione delle chiavi**: documenti che descrivono gli schemi di gestione delle chiavi.
- **Dominio di interpretazione** (DOI, *domain of interpretation*): contiene una serie di informazioni necessarie ai documenti delle altre categorie per riferimenti reciproci. Tali informazioni includono gli identificatori di algoritmi di cifratura ed autenticazione che sono stati approvati e parametri operativi quali, ad esempio, il tempo di validità di una chiave.

10.4.1 Associazioni di sicurezza (SA, securiy association)

Un concetto chiave per i meccanismi IP di autenticazione e di riservatezza è l'**associazione di sicurezza** (SA, *security association*). Un'associazione consiste in una connessione logica unidirezionale tra il mittente e il ricevente, che offre servizi di sicurezza al traffico lungo tale connessione. Se è necessaria un'associazione paritetica, al fine di effettuare scambi sicuri in entrambe le direzioni, è necessario utilizzare due associazioni di sicurezza. Nell'ambito di una SA si può utilizzare uno tra i protocolli AH e ESP, ma non entrambi.

Una SA è identificata univocamente da tre parametri:

- **Indice di parametri di sicurezza** (SPI, *security parameters index*): stringa di bit assegnata alla SA in questione con significato a livello locale. SPI è contenuto sia nell'intestazione dell'AH sia in quella dell'ESP, per consentire al sistema ricevente di selezionare la SA preposta alla gestione del pacchetto ricevuto.
- **Indirizzo IP di destinazione**: allo stato attuale sono permessi solo indirizzi *unicast*; l'indirizzo di destinazione è, quindi, quello dell'entità destinazione della SA. Tale entità può essere sia il sistema di un utente finale sia un dispositivo di rete come, ad esempio, un firewall o un router.
- **Identificatore del protocollo di sicurezza**: specifica se si tratta di una SA basata su AH o su ESP.

In ciascun pacchetto IP, la SA è quindi univocamente identificata dall'indirizzo destinazione contenuto nell'intestazione IPv4 o IPv6, e da SPI contenuto nell'intestazione di estensione (AH o ESP).

10.4.2 Security association database (SAD)

In ogni implementazione IPSec, esiste una base di dati nominale (nel senso che i servizi offerti dal database devono essere presenti in ogni implementazione di IPSec, ma il modo di realizzarla costituisce una scelta implementativa) di SA (SAD, *security association database*) che definisce i parametri associati a ciascuna SA. Normalmente, una SA è definita dai seguenti parametri:

- **Informazioni relativi alla AH** (*AH information*): comprendono l'algoritmo di autenticazione, le chiavi, il tempo di validità delle chiavi e ulteriori parametri da utilizzare con Ah (parametro obbligatorio per le implementazioni AH).
- **Informazioni relativi all'ESP** (*ESP information*): comprendono l'algoritmo utilizzato per la cifratura e l'autenticazione, le chiavi, i valori di inizializzazione, il tempo di validità delle chiavi e ulteriori parametri da utilizzare con ESP (parametro obbligatorio per le implementazioni ESP).
- **Tempo di validità della SA** (*lifetime of this security association*): intervallo di tempo, o numero di byte, dopo il quale una SA deve essere sostituita con una nuova SA (e un nuovo SPI) oppure deve essere terminata. Inoltre, questo parametro indica quale delle due azioni precedentemente descritte deve verificarsi (parametro obbligatorio per tutte le implementazioni).

10.4.3 Intestazione di autenticazione

L'intestazione di autenticazione fornisce un supporto per l'integrità dei dati e l'autenticazione dei pacchetti IP. Il controllo dell'integrità dei dati assicura che non sia possibile effettuare modifiche non rilevate al contenuto di un pacchetto in transito. L'autenticazione consente a un sistema terminale, o ad un dispositivo di rete, di autenticare l'utente o l'applicazione, e di filtrare di conseguenza il traffico; previene, inoltre, attacchi volta alla contraffazione di indirizzi (address spoofing), oggi abbastanza frequenti su internet. AH fornisce protezione contro anche contro gli attacchi di replay.

L'autenticazione si basa sull'indirizzo di un codice di autenticazione di messaggio (MAC). Ciò implica che le due parti coinvolte debbano condividere una chiave segreta.

L'intestazione di autenticazione è composta dai seguenti campi:

- **Intestazione successiva, 8 bit** (*next header*): indica il tipo di intestazione che segue immediatamente l'intestazione in questione.
- **Lunghezza del payload, 8 bit** (*payload length*): ottenuta sottraendo due unità alla lunghezza, espressa in parole da 32 bit, di AH. Ad esempio, la lunghezza di default del campo relativo ai dati di autenticazione è 96 bit, ovvero tre parole da 32 bit. Con un'intestazione di lunghezza fissa pari a tre parole, nell'intestazione sono contenute in totale sei parole, e la lunghezza del campo **payload** assume quindi valore 4.
- **Riservato, 16 bit** (*reserved*): riservati per utilizzi futuri.
- **Indice dei parametri di sicurezza, 32 bit** (*SPI, securiy parameters index*): identifica una SA.
- **Numero di sequenza, 32 bit** (*sequence number*): contatore dei valori monotonicamente crescenti.
- **Dati di autenticazione, lunghezza variabile** (*authentication data*): campo di lunghezza variabile (deve essere un multiplo di parole di 32 bit) contenente il valore per la verifica di integrità (ICV, *integrity check value*), oppure il MAC relativo al pacchetto in questione.

10.4.4 Servizio anti-replay

Un attacco di replay ha luogo quando un avversario ottiene una copia di un pacchetto autenticato e lo trasmette successivamente alla legittima destinazione. La ricezione di duplicati di pacchetti IP autenticati può provocare malfunzionamenti ai servizi o altre conseguenze indesiderate. Il campo numero di sequenza contenuto nell'AH è stato concepito per contrastare questo tipo di attacco.

Quando si costituisce una nuova SA, il **mittente** inizializza a 0 un contatore utilizzato per generare i numeri di sequenza. Ogni volta che viene spedito un pacchetto su questa SA, il mittente incrementa il contatore e inserisce il corrispondente valore nel campo numero di sequenza. Questo significa che il primo valore da utilizzare è 1. Se il servizio anti-replay è abilitato, il mittente non deve consentire che il valore del numero di sequenza compia un intero ciclo raggiungendo il valore $2^{32} - 1$, ritornando quindi a 0. Se ciò fosse consentito, ci sarebbe più di un pacchetto valido con lo stesso numero di sequenza. Quando viene raggiunto il limite di $2^{32} - 1$, il mittente dovrebbe terminare la SA corrente e negoziare una nuova SA con una nuova chiave.
Dato che IP è un servizio privo di connessione e non affidabile, non garantisce che i pacchetti siano recapitati nello stesso ordine con cui sono stati spediti e nemmeno che tutti i pacchetti raggiungano effettivamente la destinazione. Per questo, il documento IPSec relativo all'autenticazione impone che il **ricevente** realizzi una finestra di dimensione W, con un valore di default per W pari a 64. Il lato destro della finestra rappresenta il

numero di sequenza più alto ricevuto fino a quel momento per un pacchetto valido. Tale numero è denotato con *N*. Per ogni pacchetto con un numero di sequenza compreso nell'intervallo da *N-W+1* a *N* ricevuto correttamente (autenticato), viene contrassegnato il corrispondente slot. Ogni volta che viene ricevuto un nuovo pacchetto, il processo continua iterativamente, come di seguito illustrato:

1. Se il pacchetto ricevuto è nuovo, e ricade all'interno della finestra, viene verificato il MAC. Se il pacchetto è autenticato, viene contrassegnato lo slot corrispondente nella finestra.

2. Se il pacchetto ricevuto è nuovo e ricade a destra della finestra, viene verificato il MAC. Se il pacchetto è autenticato, la finestra viene spostata in avanti in modo che il numero di sequenza di tale pacchetto ne costituisca il limite destro, e viene contrassegnato lo slot corrispondente.

3. Se il pacchetto ricevuto è alla sinistra della finestra, oppure l'autenticazione non va a buon fine, il pacchetto non viene preso in considerazione, tale azione rappresenta un evento registrabile.

10.4.5 Modalità trasporto vs tunnel

Cosa AH protegge mediante il MAC: nella **modalità trasporto** i dati del pacchetto e i campi non variabili dell'header IP (no IP spoofing). In IPv6 anche estensioni dell'header. Nella **modalità tunnel** l'intero pacchetto IP (dati e header IP) e i campi non variabili dell'IP esterno (no IP spoofing). In IPv6 anche estensioni dell'header.

Nel caso di **AH in modalità trasporto**, se si utilizza IPv4, AH è inserita tra l'intestazione IP originaria e il payload IP (per esempio un segmento TCP). L'autenticazione compre l'intero pacchetto, ad eccezione dei campi modificabili nell'intestazione IPv4 che sono posti a zero per il calcolo del MAC.

Nel caso IPv6, AH è vita come un payload end-to-end; questo significa che non viene esaminata ed elaborata dai router intermedi. Per tale ragione, AH è posta dopo l'intestazione base IPv6 dopo le intestazioni di estensione relative all'hop-by-hop, all'instradamento ed alla frammentazione. L'intestazione di estensione con opzioni per la destinazione può essere posta sia prima sia dopo AH, in dipendenza del fatto che si vuole attribuire. Anche in questo caso, l'autenticazione copre l'intero pacchetto, ad eccezione dei campo modificabili che sono posti a zero per il calcolo del MAC.

Nel caso di **AH in modalità tunnel** viene autenticato l'intero pacchetto IP originario e AH è collocata tra l'intestazione IP originaria e una nuova intestazione IP esterna. L'intestazione IP interna contiene gli indirizzi della sorgente e della destinazione originaria, mentre l'intestazione IP esterna può contenere indirizzi IP diversi (ad esempio indirizzi di firewal o altri gateway per la sicurezza).

Se si utilizza la modalità tunnel, AH protegge l'intero pacchetto IP interno, compresa la sua intestazione IP. L'intestazione IP esterna e, nel caso di IPv6, le intestazioni di estensione IP esterne sono protette con l'esclusione dei campi modificabili non prevedibili.

	Modalità trasporto	**Modalità tunnel**
Requisiti	Mittente e ricevente devono usare IPsec	Mittente e ricevente possono non usare IPSec
Garanzie	Autenticazione punto-punto (end-to-end)	Autenticazione intermedia (attraverso router/firewall)

10.4.6 Esempio di AH in tunnel

1. Nodo A su rete NA genera un pacchetto IP classico per nodo B su rete NB.

2. Il router/firewall di NA lo incapsula in un pacchetto IPSec modalità tunnel e lo inoltra al router/firewall di NB.

3. Questo estrae ed **autentica** il pacchetto IP originale, poi lo inoltra al nodo B su NB.

10.4.7 Encapsulating security payload (ESP)

ESP fornisce servizi volti ad assicurare la riservatezza, inclusa quella del contenuto dei messaggi e una parziale riservatezza del flusso di traffico. Opzionalmente, ESP può anche fornire servizi di autenticazione.

Formato

Il pacchetto contiene i seguenti campi:

- **Indice dei parametri di sicurezza, 32 bit** (*SPI, security parameters index*): identifica una SA.
- **Numero di sequenza, 32 bit** (*sequence number*): contatore con valori monotonicamente crescenti; fornisce funzionalità anti-replay, come nel caso di AH.
- **Dati di payload, lunghezza variabile** (*payload data*): segmento del livello di trasporto (se si utilizza modalità trasporto) oppure IP (se si utilizza modalità tunnel) protetto tramite cifratura.
- **Completamento, da 0 a 255 byte** (*padding*): campo di completamento.
- **Lunghezza del completamento** (*pad length*): indica il numero di byte di completamento del campo precedente.
- **intestazione successiva, 8 bit** (*next header*): identifica la tipologia di dati contenuta nel campo payload, mediante l'identificazione della prima intestazione nel payload in questione.
- **Dati di autenticazione, lunghezza variabile** (*authentication data*): campo di lunghezza variabile (multiplo di parole a 32 bit) contenente l'ICV. Tale valore è calcolato prendendo come base il pacchetto ESP con l'eccezione del campo dati di autenticazione.

10.4.8 Esempio di ESP in modalità tunnel

1. Nodo A su rete NA genera un pacchetto IP classico per nodo B su rete NB.
2. Il router/firewall di NA lo incapsula in un pacchetto IPSec modalità tunnel e lo inoltra al router/firewall di NB.
3. Questo estrae, **decripta** ed **eventualmente autentica** il pacchetto IP originale, poi lo inoltra al nodo B su NB.

10.4.9 Combinazioni di SA

Una singola SA può realizzare il protocollo AH o il protocollo ESP, ma non entrambi. Esistono però alcuni casi, relativi a flussi di traffico particolari, in cui può sorgere la necessità di disporre sia dei servizi offerti da AH sia quelli forniti da ESP. Inoltre, un particolare flusso di traffico può richiedere servizi IPSec tra gli host e, allo stesso tempo, servizi distinti tra gateway di sicurezza come, ad esempio, firewall.

Il documento relativo all'architettura di IPSec elenca quattro campi di combinazioni SA che devono essere fornite da tutti gli host conformi alle specifiche IPSec. Ciascuna SA può essere AH o ESP. Nel caso di SA tra host, la modalità può essere sia tunnel sia trasporto, in tutti gli altri casi, deve essere utilizzata la modalità tunnel.

1. **Sicurezza punto-punto**. La sicurezza è fornita tra i sistemi terminali che implementano IPSec. Affinché due sistemi terminali possano comunicare tramite una SA, devono condividere le chiavi segrete approvate. Alcune combinazioni:
   - AH in modalità trasporto.
   - ESP in modalità trasporto
   - ESP seguito da AH in modalità trasporto (una SA ESP all'interno di una SA AH).
   - Uno qualsiasi dei casi precedenti all'interno di una SA AH o ESP in modalità tunnel.

   Le varie combinazioni possono essere utilizzate per fornire **autenticazione**, **cifratura** ed **autenticazione prima o dopo la cifratura**.
2. **Sicurezza fra intermediari**. La sicurezza è fornita soltanto tra gateway, mentre gli host non implementano IPSec. Questo caso illustra il supporto fornito per semplici reti virtuali private (VPN). Il documento relativo all'architettura di sicurezza specifica che, in questo caso, è sufficiente una singola SA in modalità tunnel. Il tunnel potrebbe fornire il supporto per AH, ESP oppure ESP con l'opzione di autenticazione. Non sono necessari tunnel annidati, dal momento che i servizi IPSec si applicano all'intero pacchetto interno.

3. **Tipo 1 & Tipo 2**. Si basa sul caso 2 aggiungendo a questo la sicurezza end-to-end. Qui sono permesse le stesse combinazioni viste nei casi 1 e 2. Il tunnel che collega i gateway fornisce autenticazione o riservatezza, o entrambe, per tutto il traffico tra i due sistemi terminali. Nel caso in cui il tunnel che connette i gatewa sia un tunnel ESP, viene fornita anche una parziale riservatezza del traffico. I singoli host possono realizzare, tramite SA end-to-end, qualsiasi ulteriore servizio IPSec richiesto da particolari applicazioni o utenti.

4. **Tipo 1 & sicurezza punto-intermediario**. Fornisce supporto per la gestione di un host remoto che utilizza internet per raggiungere il firewall di un'organizzazione e ottenere quindi l'accesso ad una workstation o a server protetti da firewall. È richiesta solo la modalità tunnel tra l'host remoto e il firewall. Come nel Tipo 1, possono essere utilizzate una oppure due SA tra l'host remoto e quello locale.

10.4.10 Internet Key Exchange (IKE)

Il protocollo di default per la gestione automatizzata delle chiavi in IPSec è chiamato **ISAKMP/Oakley** e si compone dei seguenti elementi:

- **Protocollo Oakley** per la scelta delle chiavi. Oakley è un protocollo per lo scambio di chiavi basato sull'algoritmo Diffie-Hellman ma rispetto a quest'ultimo fornisce maggiori garanzie di sicurezza. Oakley è un protocollo generale in quanto non impone alcuno specifico formato per le chiavi.
- **Protocollo ISAKMP** (Internet Security Association and Key Managment Protocol) per le SA e la gestione delle chiavi. ISAKMP fornisce sia un'architettura di riferimento per la gestione delle chiavi in Internet sia il supporto per uno specifico protocollo - formati inclusi - per la negoziazione degli attributi di sicurezza.

ISAKMP non impone alcuno specifico algoritmo per lo scambio di chiavi, ma è composto da un insieme di tipologie di messaggi che rendono possibile l'utilizzo di una varietà di algoritmi per lo scambio di chiavi

10.5 DNSSec

DNSSec è la versione sicura del DNS, che però non ha preso molto piede. Se il DNS non è sicuro rischiamo di beccare la persona sbagliata, con il quale potremmo fare una connessione sicura.

Tre scenari:

0 Lucchetto chiuso e url corretto: **non c'è attacco**.

1 Lucchetto aperto e url corretto: **possibile attacco** (stripping di tipo 1)

2 Nessun lucchetto e url corretto: **c'è un attacco** (downgrade ad http strip di tipo 2)

3 Lucchetto chiuso e url diverso: **c'è un attacco** (strip di livello 2 e impossessamento del DNS)

HSTS è la soluzione all'attacco 1, ovvero una policy del server che impone di accettare solo richieste di tipo HTTPS.

10.5.1 Certificate Trasparency

Thanks to modern cryptography, browsers can usually detect malicious websites that are provisioned with forged or fake SSL certificates. However, current cryptographic mechanisms aren't so good at detecting malicious websites if they're provisioned with mistakenly issued certificates or certificates that have been issued by a certificate authority (CA) that's been compromised or gone rogue. In these cases, browsers see nothing wrong with the certificates because the CA appears to be in good standing, giving users the impression that the website they're visiting is authentic and their connection is secure. One of the problems is that there is currently no easy or effective way to audit or monitor SSL certificates in real time, so when these missteps happen (malicious or otherwise), the suspect certificates aren't usually detected and revoked for weeks or even months. What's more, these types of SSL missteps are occurring with increasing frequency. Over the past few years there have been numerous instances of misissued certificates being used to spoof legitimate sites, and, in some case, install malicious software or spy on unsuspecting users.

In one case, a prominent Dutch CA (DigiNotar) was compromised and the hackers were able to use the CA's system to issue fake SSL certificates. The certificates were used to impersonate numerous sites in Iran, such as Gmail and Facebook, which enabled the operators of the fake sites to spy on unsuspecting site users.

In another case, a Malaysian subordinate certificate authority (DigiCert Sdn. Bhd.), mistakenly issued 22 weak SSL certificates, which could be used to impersonate websites and sign malicious software. As a result, major browsers had to revoke their trust in all certificates issued by DigiCert Sdn. Bhd. (Note: DigiCert Sdn. Bhd. is not affiliated with the U.S.-based corporation DigiCert, Inc.)

More recently, a large U.S.-based CA (TrustWave) admitted that it issued subordinate root certificates to one of its customers so the customer could monitor traffic on their internal network. Subordinate root certificates can be used to create SSL certificates for nearly any domain on the Internet. Although Trustwave has revoked the certificate and stated that it will no longer issue subordinate root certificates to customers, it illustrates just how easy it is for CAs to make missteps and just how severe the consequences of those missteps might be.

In many cases, mistakenly issued certificates have been used by hackers for malicious attacks that have dire consequences, but the fallout after mitigation can be far ranging and harmful, too. Eventually, the Dutch CA's certificates were revoked and the CA was shut down. The revocation and closure caused a ripple effect throughout the Netherlands as people were denied access to government and private sites that were provisioned with the CA's SSL certificates.

Certificate Transparency aims to remedy these certificate-based threats by making the issuance and existence of SSL certificates open to scrutiny by domain owners, CAs, and domain users. Specifically, Certificate Transparency has three main goals:

- Make it impossible (or at least very difficult) for a CA to issue a SSL certificate for a domain without the certificate being visible to the owner of that domain.
- Provide an open auditing and monitoring system that lets any domain owner or CA determine whether certificates have been mistakenly or maliciously issued.
- Protect users (as much as possible) from being duped by certificates that were mistakenly or maliciously issued.

Certificate Transparency satisfies these goals by creating an open framework for monitoring the TLS/SSL certificate system and auditing specific TLS/SSL certificates.

11 Intrusion Detection

Un intruso è un vero e proprio **processo utente** (con certi privilegi). Raramente i virus possono essere considerati tipi particolari di intruso: un virus non è un batterio. Più facilmente un worm è un tipo di intruso, un worm è autonomo, proprio come un batterio. Un intruso **non è** né un virus né un worm ma può **installarne** uno.

Differenze tra intrusione e DoS: un **DoS** ha effetti temporanei ed è immediatamente pubblico causando un blocco delle risorse.
Un'**intrusione** ha generalmente effetti permanenti e spesso non è reso pubblico causando un uso illecito delle risorse.

11.0.1 Negazione del servizio (DoS)

Definizione: Attacco che impedisce la normale operatività del sistema, degradandola o bloccandola del tutto. Esistono tre tipologie di attacchi DoS:

- **cDoS**: mira al consumo di risorse computazionale
- **mDoS**: mira al consumo della memoria
- **bDoS**: mira al consumo della banda di trasmissione
- **DDoS**: DoS sferrato da più macchine (Distribuited Denial of Service)

Un attacco DDoS è un tentativo di consumare le risorse del bersaglio in modo che non sia in grado di fornire il servizio. Parlando in generale, la risorsa consumata è o una risorsa interna dell'host sul sistema bersaglio o la capacità di trasmissione dei dati in une rete locale alla quale il bersaglio è collegato.

Breve storia

I primi attacchi DDoS registrati risalgono al 1998. Nel febbraio 2000 ricordiamo attacchi DDoS verso Yahoo!, Amazon, CNN, eBay con per danni di circa 50 milioni di dollari. Tra il 2000 e il 2001 si registrano moltissimi attacchi DDoS a siti aziendali e governativi USA, e ad altri ISP europei (a partire da Tiscali UK). Il **21 ottobre 2002** vengono attaccati i 13 Root DNS di Internet con conseguenze minime ma vulnerabilità della rete mondiale appurata.

Effetti di un (D)DoS

Non si ottiene un profitto diretto ma indiretto, in genere con una mancanza di profitto o perdita di reputazione. Sempre più ditte basano operatività e presenza sul mercato su sistemi informativi.

Come reagire?

Per attacchi DoS è possibile chiudere tutte le connessioni provenienti dalla singola rete attaccante. Per attacchi di tipo DDoS, chiudere tutte le connessioni è difficile e sconveniente.
Una possibile soluzione potrebbe essere quella di filtrare il numero di connessioni accettate (htaccess con firewall).
Un'euristica di prevenzione consiste nel complicare computazionalmente l'accesso (**cookie transformation**).

Scenario normale:

1. $C \longrightarrow S \ : \ m1$
2. $S \longrightarrow C \ : \ m2$
3. $C \longrightarrow S \ : \ m3$

Cookie tranformation: creazione del messaggio computazionalmente complicato rimandata fino a quando è chiaro che il client sia pronto a ricevere all'IP del passo 1.

1. $C \longrightarrow S \ : \ m1$
2. $S \longrightarrow C \ : \ m2_S$
3. $C \longrightarrow S \ : \ m1, m2_S, h(m1, m2_S)$
4. $S \longrightarrow C \ : \ m2_C$

11.0.2 Intrusione

Definizione: Ottenere illecitamente privilegi superiori a quelli posseduto lecitamente.

Ovvero acquisire un accesso al sistema oppure ampliare i propri privilegi di accesso. Non esclusivamente da rete, ma anche tramite software nocivo mediate supporti rimovibili (CD, USB, ecc). Si riduce spesso a violare i meccanismi di autenticazione e/o controllo d'accesso.

Tecniche di intrusione

L'obiettivo di un intruso è quello di avere accesso ad un sistema o di aumentare l'insieme di privilegi disponibili nel sistema. In genere, ciò richiede che l'intruso entri in possesso di informazioni che avrebbero dovuto restare protette. Nella maggio parte dei casi si tratta di una password utente.

- **Violazioni di password**: tecniche standard e non standard.
- **Intercettazione di informazioni sensibili**: scovare (non solo password) se transitano in rete senza crittografia oppure violare protocolli di sicurezza.
- **Uso di combinazioni di software nocivo**: sullo stile del worm Morris.

Worm di Morris

Prima dell'attuale generazione di worm, il più conosciuto era quello rilasciato su Internet da Robert Morris nel 1998. Progettato per diffondersi su sistemi UNIX, usava svariate tecniche di propagazione. Quando una copia iniziava l'esecuzione, la sua prima attività era quella di scoprire tutte le altre macchine conosciute dall'host sul quale si trovava e che avrebbero accettato informazioni provenienti da questo. Questo worm procedeva esaminando molto elenchi e tabelle, comprese le tabelle di sistema che dichiaravano quali altri macchine erano considerate fidate da questo host, i file per la spedizione della posta elettronica, le tabelle attraverso le quali gli utenti attribuivano il permesso di accedere ad account remoti, e un programma che riportava lo stato delle connessioni di rete. Per ciascuno degli host scoperti il worm provava molte tecniche per insinuarsi.

1. Tentava di accedere e registrarsi ad un host remoto come un utente legittimo. Con questo metodo, in primo luogo cercava di "scassinare" il file locale delle password e poi utilizzava le password scoperte e i corrispondenti ID degli utenti. L'assunzione era che **molti utenti avrebbero usato la stessa password su sistemi differenti**. Per ottenere la password, questo worm eseguiva un programma di password cracking che provava:
   - ciascun nome dell'account dell'utente e semplici permutazioni di questo
   - una lista di 432 password già pronte che Morris pensava fossero probabili candidate
   - tutte le parole nel sistema di directory locale
2. Sfruttava un baco nel protocollo finger che riportava in che posto si trova l'utente remoto.
3. Sfruttava una trapdoor nell'opzione di individuazione e recupero degli errori del processo remoto che riceve e inviava la posta.

Se qualcuno di questi attacchi non aveva successo, questo worm otteneva la comunicazione con l'interprete dei comandi del sistema operativo. Inviava poi a questo interprete un breve programma di avvio, emetteva un comando che mandava in esecuzione quel programma e poi si disconneteva. Il programma di avvio chiamava poi il programma padre e scaricava il resto del programma worm. Il nuovo worm veniva poi eseguito.

11.0.3 Trattamento delle intrusioni

Definizione: Rilevare l'intrusione prima possibile (**intrusion detection**) ed espellere l'intruso non appena rilevato per minimizzare i datti che può provocare (**intrusion management**).

IDS (Intrusion Detection System)

Registra il comportamento del sistema (**auditing**) e analizza tale log tramite tecniche di pattern matching e funzionamento real time.

Il problema fondamentale per un IDS è definire un il comportamento dell'intruso rispetto a quello dell'utente legittimo. Trovare il miglior compromesso fra scambiare utente legittimo per intruso (falso positivo) e intruso per utente legittimo (falso negativo).

Record di auditing

I record di auditing costituiscono uno strumento fondamentale per la rilevazione di intrusioni. Come ingresso al sistema di rilevazione delle intrusioni deve essere mantenuta una registrazione della attività in corso. Fondamentalmente, si utilizzano due tipologie di record di audit:

- **Record di audit nativi**: praticamente tutti i sistemi operativi multiutente contengono software di accounting che raccoglie informazioni sulle attività degli utenti. Utilizzando queste informazioni, si evita l'impiego di ulteriore software di raccolta. Lo svantaggio è che i record nativi possono non contenere le informazioni necessarie oppure possono contenerle ma non in formato appropriato.
- **Record di audit specifici per la rilevazione**: si può realizzare una funzionalità di raccolta per la generazione di record di audit contenenti solo le informazioni necessarie al sistema di rilevazione delle intrusioni. Un vantaggio di questo approccio è che può essere reso indipendente dagli specifici ambienti e portabile su una vasta gamma di sistemi. Lo svantaggio è relativo al carico di elaborazione aggiunto derivante dall'avere due software di accounting in esecuzione sulla stessa macchina.

Record specifici per il rilevamento

La maggior parte delle operazini degli utenti è composta da un certo numero di azioni elemntari. La scomposiione di un'operazione utente in azioni elementari presenta tre vantaggi:

1. Dato che gli oggetti rappresentano le entità da proteggere nel sistema, l'uso di azioni elementari rende possibile la registrazione di tutti i componenti che interessano un oggetto. Pertanto, i sistema può rilevare tentativi di sabotaggio dei meccanismi di controllo dell'accesso (notando un'anomalia nel numero di condizioni di eccezione restituite) e può rilevare sabotaggi andati a buon fine notando un'anomalia nell'insieme degli oggetti accessibili al soggetto.

2. Record di audit su singoli oggetti e singole azioni semplificano il modello e la sua realizzazione.

3. Data la struttura semplice e uniforme dei record di audit specifici per la rilevazione può risultare relativamente semplice ottenere questa informazione e una sua parte, stabilendo una semplice corrispondenza fra i record nativi esistenti e i record specifici per la rilevazione.

regole di Denning

Un buon esempio di record di audit specifico per la rilevazione è stato sviluppato da Dorothy Denning. Ciascun record di audit contiene i seguenti campi:

- **Soggetto**: colui che ha iniziato l'azione. Un soggetto tipicamente è un utente connesso mediante terminale ma potrebbe essere un processo che agisce per conto di un utente o di un gruppo di utenti. Tuttavia l'attività è generata dai comandi inviati dai soggetti; questi possono essere raggruppati in differenti classi di accesso che possono anche sovrapporsi.

- **Azione**: operazione eseguita dal soggetto su o mediante un soggetti; ad esempio: connessione, lettura operazione di I/O, esecuzione.

- **Oggetto**: ricettore di azioni. Esempi di oggetti comprendono file, programmi, messaggi, record, terminali, stampanti, struttura dati create da utenti e/o programmi. Quando un soggetto agisce da ricettore di un'azione come, ad esempio, la posta elettronica, viene considerato un oggetto. Gli oggetti possono essere raggruppati per tipo. La granularità degli oggetti può variare per tipo e ambiente. Ad esempio, le azioni su basi di dati possono essere registrate a livello di intera base di dati oppure a livello di singolo record.

- **Condizione di esecuzione**: indica l'eventuale condizione di esecuzione restituita al soggetto in caso di azione parzialmente o totalmente rifiutata dal sistema.

- **Utilizzo di risorse**: una lista di elementi quantitativi in cui ogni elemento fornisce l'ammontare di utilizzo di una certa risorsa.

- **Timestamp**: valore univoco di tempo e data, che identifica l'istante in cui si è verificata l'azione.

Tecniche di rilevamento

Indipendenti dal passato (standard) ad esempio *massimo tre tentativi di inserimento di passord.*
Dipendenti dal passato: osservazione/campionamento del funzionamento/comportamento del sistema in passato, basata sull'assunzione che il futuro sarà come il passato.

Rilevamento statistico: creazione di un modello statistico del comportamento di un utente legittimo quando l'intruso si discosta sufficientemente dal modello, ed è particolarmente efficace per attaccanti esterni.

Rilevamento a regole: creazione di regole per il comportamento di un intruso, particolarmente efficace per attaccanti interni.

Date queste metriche generali, si possono eseguire diversi test per determinare se l'attività in corso rientro entro certi limiti considerati accettabili:

- **Media e deviazione standard**: calcolati su un parametro in un arco di tempo, danno l'idea del comportamento medio e della sua variabilità

- **Modello operativo**: definisce un limite di accettabilità per un parametro e segnala una intrusione quando viene superato

- **Multivariazione**
- **Processo di Markov**
- **Serie temporale**

Rilevamento a regole

Grosso database di regole, fino a $10^4 - 10^6$.

- Nessun utente dovrebbe leggere i file contenuti nelle directory personali di altri utenti
- Nessun utente dovrebbe scrivere su file di altri utenti
- Gli utenti che si connettono al sistema dopo ore spesso accedono a file utilizzati in precedenza
- Nessun utente generalmente accede direttamente ai dischi bensì usa servizi di alto livello offerti dal S.O.
- Nessun utente dovrebbe trovarsi connesso più volte allo stesso sistema
- Nessun utente esegue copie di eseguibili

Es: `ntop`, e `snort`: a regole ma basati su firme degli attacchi.

Tecniche di protezione

1. **Limitare l'automazione**: impedire il più possibile il ripetersi di tentativi. Es. dopo 3 inserimenti di password errata il sistema resetta la connessione, oppure, dopo tre inserimenti di PIN errato, il telefono blocca la SIM.
2. **Usare esche**: risorse fittizie che attirano gli attaccanti, magari distogliendoli da risorse importanti o convincendoli a rimanere di più nel sistema.
3. **Usare trappole**: massiccio monitoraggio appena l'attaccante accede all'esca.

11.0.4 IMS (Intrusion Management System)

Definizione: L'intrusione va **contenuta** (*containment*), **rimossa** (*eradication*), **riassorbita** (*recovery*) e **punita** (*punishment*).

C'è un nuovo regolamento europea sugli incidenti. Tutti quelli che forniscono servizi essenziali (sanitario, carburanti, ecc) devono avere una procedura per il rilevamento di ciò che si chiama incidente (security incident) ISO 27035. Le quattro fasi di gestione delle intrusioni, ricordano molto ciò che la direttiva NIST snocciola ed esemplifica come fasi di gestione di un tipico incidente informatico (con qualche fase in più). La gestione dell'incidente è stata la storia che c'è dietro la direttiva NIST.

12 Software nocivo (malware)

Bug vs software nocivo

*Definizione di **bug***: proprietà inattesa del software. Può essere sfruttato da un altro software non necessariamente nocivo. I bug sono tipicamente preterintenzionali.

*Definizione di **software nocivo***: software scritto con l'**esplicito** scopo di violare la sicurezza di un sistema. Non necessariamente sfrutta dei bug. Un bug preterintenzionale non rende nocivo il software ospitante

Caratteristiche del software nocivo

Un virus è un pezzo di software che può "infettare" altri programmi modificandoli; la modifica comprende la replica del programma virus, il quale può quindi continuare ad infettare altri programmi.

Un virus può eseguire qualsiasi cosa facciano gli altri programmi. La sola differenza è che si "attaccano" agli altri programmi e sono eseguiti segretamente, quando il programma ospite è in funzione. Una volta che il virus è in esecuzione, può effettuare qualsiasi operazione, come cancellare file e programmi. Durante il proprio ciclo di vita, un tipico virus passa attraverso due fasi:

- **Carico**: è la specifica violazione di sicurezza. *Sempre presente.*
- **Propagazione**: meccanismo di trasmissione fra le macchine. *Non sempre presente.*

Tassonomia dei software nocivi

Trapdoor o backdoor

Definizione: punto segreto di accesso ad un programma senza le procedure di sicurezza altrimenti previste.

Una backdoor (trapdoor) è un punto di accesso segreto ad un programma, che consente - a chi ne è a conoscenza - di entrare senza attraversare le consuete procedure di controllo dell'accesso. I programmatori hanno legittimamente utilizzato per molti anni la backdoor con il fine di rilevare e correggere gli errori per testare i programmi.

Le trapdoor diventano una minaccia quando sono utilizzate da programmatori senza scrupoli, al fine di ottenere accessi non autorizzati

Bomba logica

Definizione: frammento di codice di un programma non nocivo pronto ad "esplodere" quando si verificano certe condizioni.

Una delle prime tipologie di minaccia, precedente a virus e worm, è la bomba logica. La bomba logica è un codice contenuto in qualche programma legittimo impostato per "esplodere" a verificarsi di specifiche condizioni. Esempi di condizioni possono essere impiegate come innesco (trigger) di una bomba logica sono la presenza o l'assenza di specifici file, un particolare giorno della settimana o una data particolare, o l'esecuzione di un'applicazione da parte di uno specifico utente. Una volta innescata, una bomba logica può produrre l'alterazione o la cancellazione di dati o di interi file, causare blocchi delle macchine o altri danni.

Cavalli di Troia

Definizione: programma utile o apparentemente utile che in fase di esecuzione compia violazioni di sicurezza.

Un cavallo di Troia è un programma o una procedura di comando che svolte un compito utile, o apparentemente utile. Al suo interno è però inserito un codice nascosto che, quando richiamati, esegue alcune funzioni indesiderate o dannose. I cavalli di Troia possono essere impiegati per svolgere indirettamente funzioni che un utente non autorizzato non potrebbe svolgere direttamente, ad esempio accedere ai file di un altro utente in un sistema condiviso.

Zombie

Definizione: programma nocivo che sfrutta una macchina remota già violata per lanciare nuovi attacchi che difficilmente possono essere ricondotti all'autore dello zombie.

Si tratta di un programma che, segretamente, prende il controllo di un altro computer connesso a Internet e poi lo utilizza per lanciare attacchi che risultano difficili da far risalire al creatore dello zombie. Gli zombie vengono usati negli attacchi di negazione del servizio, tipicamente contro siti web disegnati come bersaglio. Lo zombie viene messo di nascosto su centinaia di computer appartenenti a terze parti insospettabili e poi viene utilizzato per sopraffare siti web designati come bersaglio.

Worm

Definizione: programma nocivo che infetta macchine remote, ciascuna delle quali a loro volta infetta altre macchine remote.

Si tratta di un programma che può replicarsi e inviare copie di se stesso, da un computer all'altro attraverso le connessioni di rete. Al momento del suo arrivo, può essere attivato per replicarsi e propagarsi ancora. Oltre alla propagazione, un worm di solito esegue qualche operazione indesiderata. Un virus della posta elettronica ha alcune delle caratteristiche di un worm, perché si propaga da un sistema all'altro. Tuttavia, si può ancora classificarlo come virus in quanto richiede un'azione umana per la sua diffusione. Un worm si adopera attivamente per trovare più macchine da infettare e ciascuna macchina infettata serve da trampolino di lancio automatico per attacchi ad altre macchine.

Virus

Definizione: programma nocivo che viola altri programmi non nocivi, sfruttandoli per propagarsi.

Violazione tipica mediante aggiunta di pezzi di codice indesiderati ai programmi (applicativi o per dati, settore di boot). Possono essere difficili da individuare soprattutto se sono **polimorfi** (cambiano forma) o **criptati** (nascondono le proprie tracce).

Macrovirus

Questo tipo di virus sfrutta una caratteristica peculiare di Word e di altre tipiche applicazioni di supporto al lavoro di ufficio, come Microsoft Excel: le macro. In sostanza, una macro è un programma eseguibile, contenuto in documenti elettronici o in altri tipi di file, utilizzato per automatizzare compiti di tipo ripetitivo. Il linguaggio delle macro generalmente è una variante del linguaggio di programmazione Basic. Un utente può definire all'interno di una macro una sequenza di digitazioni da tastiera e fare in modo che la macro sia invocata digitando un tasto funzionale oppure una breve combinazione speciale di tasti. Le più recenti versioni di Word forniscono maggiore protezione dai virus delle macro.

12.1 Struttura dei virus

Un virus può essere inserito in testa o in coda ad un programma eseguibile, oppure può essere allegato seguendo differenti modalità Il punto cruciale del meccanismo di funzionamento è che il programma infetti, quindi lanciato, esegue prima il codice del virus e successivamente il codice originario del programma.

Un programma infetto inizia con il codice del virus e funziona nel seguente modo. La prima riga del codice è un salto al programma principale del virus. La seconda riga è uno speciale marcatore utilizzato dal virus per stabilire se un programma vittima potenziale è già stato infettato dal virus. Quando il programma infettato viene richiamato, il controllo passa immediatamente al programma principale del virus. Dapprima il virus ricerca ed infetta file eseguibili non ancora infettati. Successivamente, può svolgere qualche azione, in genere dannosa per il sistema. Tale azione può essere svolta ogni volta che il programma viene richiamato, oppure può essere una bomba logica che si attiva solo sotto detemrinata condizioni. Infine, il virus trasferisce il controllo al programma originario. Se la fase di infezione del programma è sufficientemente rapida, è improbabile che un utente noti differenze tra l'esecuzione di un programma "sano" e quella di una "infetta".

Un virus come quello descritto può essere facilmente individuato, in quanto la veriosne infettata del programma è più lunga di quella non infettata corrispondente. Un modo per contrastare questo semplice modo di identificazione del virus è quella di eseguire la compressione di un file eseguibile in modo che le versioni infette e non, non abbiano identica lunghezza.

La struttura di un virus che comprime è la seguente:

1. Comprime il nuovo file da infettare
2. Appende il virus all'inizio
3. Decomprime il resto del file ospite
4. Lo esegue

12.2 Rimozione di software nocivo

Tre soluzioni principali:

1. **Antivirus**. È il metodo più comune ed in continua evoluzione, ma è limitato dalla necessità di aggiornamenti
2. **Sistemi immuni**. È un prototipo sviluppato da IBM a fine anni '90, che sfrutta gli antivirus. Usa un'intera struttura distribuita di prevenzione.
3. **Software sentinella**. Integrato col sistema operativo, blocca l'esecuzione di codice nocivo.

12.2.1 Antivirus

Esegue tre compiti:

1. **Individuazione/Rilevazione**: una volta avvenuta l'infezione, rileva e localizza il virus.
2. **Identificazione**: compiuta la rilevazione, identifica lo specifico virus che ha infettato il programma.
3. **Eliminazione/Rimozione**: identificato il virus, rimuove tutte le tracce dal programma infetto e ripristina lo stato originario. Elimina il virus da tutti i sistemi infetti in modo che l'infezione non si espanda ulteriormente. Può essere però problematica in caso di virus polimorfi.

A seconda di come vengano eseguiti i compiti 1 e 2, si distinguono quattro generazioni di antivirus.

Un programma di scansione di **prima generazione** necessita di una signature per effettuare l'identificazione di un virus. Il virus può contenere wildcards, ma ha sostanzialmente la stessa struttura e le stesse configurazione di bit in tutte le sue copie. Questi programmi di scansione si limitano a rilevare virus conosciuti sulla base delle loro signature. Un altro tipo di programmi tiene traccia della lunghezza dei programmi e controlla le variazioni di lunghezza. Per fare ciò utilizza una database delle signature.
Un programma di scansione di **seconda generazione** non i basa su una specifica signature. Piuttosto, usa regole euristiche per ricercare posibili infezioni virali. Una categoria di programmi di scansione di questo tipo ricerca frammenti di codice che spesso sono associati ai virus. Ad esempio, il programma di scansione può ricercare l'inizio di un ciclo di cifratura utilizzato nei virus polimorfi e scoprire la chiave di cifratura. Una volta scoperta tale chiave, il programma di scansione può decifrare il virus per l'identificazione e poi rimuovere l'infezione e ripristinare il programma originario. Un'altra tipologia di programmi di scansione di seconda generazione si fonda sulla verifica di integrità. Si appone un valore di checksum a ciascun programma. Se un virus lo infetta senza cambiare il checksum, allora la verifica di integrità rileverà il cambiamento. Per contrastare un virus sofisticato al punto di modificare anche il valore del checksum in fase di infezione di un programma, si può utilizzare un funzione hash cifrata.
I programmi di **terza generazione** sono programmi resistendi in memoria che identificano un virus in base alle sua azioni piuttosto che rispetto alla sua struttura all'interno di un programma infetto. Tali programmi hanno il vantaggio alla sua struttura all'interno di un programma infetto. Tali programmi hanno il vantaggio di non richiedere la definizione di signature ed euristiche per una vasta gamma di virus. In questo caso, è necessario solamente identificare l'insieme di azioni sintomatiche di possibile infezione in corso e quindi intervenire.
I prodotti di **quarta generazione** sono pacchetti software contenenti una vasta gamma di tecniche antivirus usate congiuntamente, fra cui componenti per la scansione di trap di attività. Inoltre, un prodotto di quarta generazione comprende funzionalità di controllo dell'accesso, per limitare le capacità del virus di penetrare nel sistema e di aggiornare file per passare l'infezione.
Con i pacchetti di quarta generazione, si ha una strategia di difesa più estesa, con misure di sicurezza di carattere più generale per i sistemi di elaborazione.

Tecniche avanzate

Quando viene eseguito un file contenente un virus polimorfo, il virus deve decifrare se stesso per attivarsi. Per scoprire tale struttura, i file eseguibili sono mandati in esecuzione attraverso un programma di scansione (GD, *generic decryption*), contenente i seguenti elementi:

- **Emulatore di CPU**: elaboratore virtuale basato su software. L'emulatore interpreta le istruzioni di un file eseguibile invece di eseguirle sul processore sottostante. L'emulatore comprende versioni software di tutti i registri e di altri componenti hardware del processore, in modo che quello reale non venga toccato dai programmi interpretati dall'emulatore.
- **Controllo di firme**: modulo che effettua la scansione del codice sotto analisi per ricercare firme di virus note.

Software sentinella

Anche un virus che superi antivirus di qualunque generazione deve alla fine fare "richieste" al sistema operativo. Un software sentinella può bloccare (o chiedere conferma all'utente) le richieste potenzialmente pericolose.

13 Firewall

Definizione: dispositivo che controlla il flusso di traffico fra reti con diverse impostazioni di sicurezza.

Controllo fra una LAN e la rete esterna e fra sottoreti interne.

I firewall costituiscono un mezzo efficace per proteggere un sistema locale o una rete di sistemi da attacchi perpetrati tramite la rete, permettendo nel contempo accesso al mondo esterno tramite reti geografiche e Internet.

La differenza tra un firewall, un IDS e un antivirus è che IDS e antivirus risolvono il problema, mentre il firewall previene il problema.

Tutto il traffico fra una rete e l'altra deve passare attraverso il firewall ad esempio impedire l'accesso ad Internet se non via firewall. Il passaggio del traffico è regolata da un'apposita **politica di sicurezza**. Il firewall deve essere sicuro esso stesso, installato su un **bastion host** (computer specializzato nell'isolare una rete locale da una connessione internet pubblica, creando uno scudo che permette di proteggere la rete locale da attacchi esterni).

13.1 Limiti di un firewall

1. Non possono proteggere da attacchi che ricevono il permesso di superare il firewall (connessioni via modem).
2. Non possono proteggere da minacce interne.
3. Proteggono minimamente dal passaggio di software nocivo, setacciare tutto il traffico sarebbe impraticabile.
4. Possono degradare le prestazioni della rete.
5. Possono essere difficili da configurare, dovuto ad un difficile compromesso fra libertà e sicurezza.
6. Non possono proteggere da attacchi non ancora documentati ai protocolli di sicurezza

13.2 Tassonomia

13.2.1 Router a filtraggio di pacchetti

Un router a filtraggio di pacchetti applica un insieme di regole a ogni pacchetto IP in ingresso e in uscita e, sulla base della valutazione di tali regole, scarta oppure inoltra il pacchetto. Solitamente il router è configurato in modo da filtrare i messaggi in entrambe le direzioni (da e verso la rete interna). Le regole di filtraggio sono basate sulle informazioni contenute in un pacchetto di rete (indirizzi IP, porte, ecc).

13.2.2 WAF (Web Application Firewall)

Funzioni evolute che gestiscono il traffico a livello di applicazione. L'utente contatta il gateway mediante applicazione TCP/IP. I gateway a livello applicazione sono in linea di massima più sicuri rispetto al filtraggio dei pacchetti. Invece di cercare di gestire a livello TCP e IP le molteplici possibili combinazioni che devono essere permesse o vietate, il gateway a livello applicazione ha necessità di esaminare solo un numero ridotto di applicazioni ammissibili. Inoltre, è semplice effettuare operazioni di log e di audit livello applicazione sul traffico in entrata.
Uno dei principali svantaggi di questo tipo di gataway è il carico addizionale di elaborazione che ogni connessione comporta.

13.2.3 Gateway a livello di circuito

Questo tipo di gateway può essere realizzato mediante un sistema *stand-alone* o può essere una funzione specializzata realizzata per alcune applicazioni da un gateway a livello di applicazione. Un gateway a livello di circuito non contiene connessioni TCP end-to-end; al contrario, stabilisce due connessioni TCP, una che connette il gateway ad un utente TCP su un host interno e l'altra che connette il gateway ad un utente TCP esterno. Successivamente, dopo la connessione, il gateway trasmetti i segmenti TCP senza esaminarne il contenuto. La funzione di sicurezza consiste nell'individuare quali connessioni saranno permesse.