NoiStudiamo

APPUNTI

Internet Security

Attacchi, proprietà di sicurezza, autenticazione, crittografia, protocolli di sicurezza, intrusion detection, malware, SSL e TLS

A CURA DI

Paolo Volpini

Novembre 2026

INDICE

Indice degli argomenti

Gli argomenti principali degli appunti e la pagina in cui iniziano.

1. **Introduzione** 4
2. **Strumenti generali di sicurezza** 6
3. **Attacchi e attaccanti** 10
4. **Proprietà di sicurezza** 14
5. **Forme di autenticazione** 30
6. **Crittografia** 34
7. **Protocollo di sicurezza** 60
8. **Intrusion Detection** 66
9. **Malware** 70
10. **SSL e TLS** 76

Internet Security - Teoria

Paolo Volpini

0 - Introduzione

Qualunque software che noi impieghiamo implica un atto di fede. Dobbiamo, infatti, fidarci del fatto che il codice sia sicuro, ma questo non è sempre il caso. Non è detto che dobbiamo fidarci.

Alcuni falsi miti ed esempi reali

Analizziamo alcuni casi reali e falsi miti sulla sicurezza informatica.

Trojan horse

Un trojan horse è un file che appare come file innocente ma che nasconde codice malevolo. Un trojan horse può anche essere realizzato con pochi comandi shell.

Ogni file Unix è dotato di un proprietario e di un gruppo. Normalmente un programma riceve i permessi dell'utente che lancia. Se il bit SUID di un eseguibile è impostato ad 1, allora un qualsiasi utente può eseguire il programma con i permessi del proprietario.

Supponiamo che una vittima riceva il trojan di nome `ls` e lo metta nella home directory (nota: a livello socio-tecnologico è accettabile. Inizialmente era molto d'uso prendere gli eseguibili e metterli nella home. La variabile PATH considerava anche la home directory in primis):

```
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
```

Il trojan:

- copia l'emulatore di terminale `/bin/sh` nella cartella temporanea con il nome di `.xxsh` in modo tale che non sia visibile. La copia di un file ne modifica i permessi, in quanto cambia l'ownership del file;

- aggiunge il SUID bit per lo user e il permesso di esecuzione per gli altri utenti a `.xxsh`. La copia del file cambia l'owner di esso;
- rimuove se stesso;
- stampa l'output di `ls` originale, passando i parametri passati inizialmente all'eseguibile.

Supponiamo di avere un utente `victim`. Un utente `attacker` può creare il file `ls` nella home directory di `victim` e renderlo eseguibile. A questo punto quando `victim` esegue `ls` e `attacker` può lanciare un interprete dei comandi con i permessi di `victim`.

Chiaramente `ls` è nella current directory di victim quando apre la shell. Oggi in PATH non viene mai considerata la home directory.

Un altro problema di PATH che è possibile sfruttare è l'ordine delle cartelle della lista. Le prime directory in PATH hanno priorità. E inizialmente PATH aveva come prima directory la home directory. Per cui questo trojan in realtà era abbastanza potente. Inoltre, se il trojan non avesse rimosso se stesso e la home directory era a priorità massima nel PATH, allora l'esecuzione continuava all'infinito, in quanto veniva chiamato nuovamente il trojan.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

1 - Strumenti generali di sicurezza

Un **attacco** potrebbe essere la violazione di una qualunque regola di sistema. Molti attacchi oggi sfruttano componenti sociali (i.e. social engineering).

Un esempio di attacco di social engineering è il **phishing**. Tale attacco mira ad estrapolare informazioni da un utente facendo sì che l'utente stesso le conceda inconsciamente, ad esempio fingendo una mail di cambio di password o che finge un tentativo di accesso illecito.

Esistono strumenti generali di sicurezza che vengono impiegati per difendersi dagli attacchi:

- crittografia, simmetrica e asimmetrica;
- policy, ovvero regole. Alcuni esempi sono policy a livello di privacy e di controllo di accesso;
- autenticazione, tramite conoscenza (password, PIN), possesso (smartcard, smart token) e biometria (impronte, iride). L'obiettivo è identificare in maniera più disambigua possibile l'utente;
- programmi di protezione quali antivirus, Intrusion Detection System e firewall;
- protocolli di sicurezza quali SSH e SSL;
- sensibilizzazione dell'utente tramite formazione.

Strumenti di antivirus commerciali come Windows Defender implementano misure di sicurezza standard per l'azienda. Il fatto che il prodotto sia commerciale non ci permette però di garantire il massimo della sicurezza a meno di testing (neanche se il codice, comunque, fosse del tutto open-source). La realtà dei fatti è che le misure di sicurezza possono sempre essere aggirate, per cui si ricorre a mitigazioni.

Limitazioni della crittografia

Sebbene la crittografia sia una scienza esatta come branca della matematica, il suo utilizzo per ottenere sicurezza in pratica non offre le stesse garanzie. Esistono limiti a livello di utilizzo di risorse, oppure limiti imposte da leggi nazionali. Magari anche la stessa implementazione software dell'algoritmo di crittografia non è sicura, ad esempio la chiave pseudorandom non è realmente pseudocasuale.

Limitazioni delle password

Una password robusta è lunga e proveniente da uno spazio di simboli molto grande. La password chiaramente può essere rubata tramite phishing, oppure memorizzata in maniera non sicura (file di testo, post-it facilmente visibili, password manager non sicuri). La password può essere attaccata in diversi modi, ad esempio con attacchi dizionario, bruteforcing e mask attacks.

Limitazioni dei firewall

Il firewall permette di definire il perimetro di sicurezza. Tuttavia, non risolve problemi derivanti da attacchi dall'interno. In particolare, ci può essere un attaccante interno, o un man-in-the-middle.

Definizione di sicurezza per punti

Definiamo allora la sicurezza tramite i seguenti punti:

- non è un prodotto, ma è un processo continuo di operazioni che garantisce confidenzialità, disponibilità e integrità dei dati;
- è l'anello più debole della catena;
- è espressa in virtù di modelli di attaccanti;
- è sempre soggetta all'analisi costi-benefici dell'attaccante;
- si realizza mediante livelli di sicurezza, in quanto la costruzione della sicurezza avviene sia a livello fisico che a livello logico.

Rischi base per la sicurezza

Esistono alcuni rischi base per la sicurezza:

- i singoli sistemi da mettere in sicurezza possono essere molto complessi (ad esempio, il browser è un sistema complesso);
- il sistema da mettere in sicurezza è composto. Non è sempre detto che la combinazione dei sistemi sia sicura;
- il codice sorgente è predisposto ad avere bug;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- le **proprietà emergenti**, cioè le proprietà che nascono dal modo in cui le componenti di un sistema interagiscono. Un esempio è il double-spending delle monete virtuali, poiché anche se la valuta virtuale impiega crittografia, firma digitale e un database per registrare la transazione, serve una proprietà che garantisca che una transazione valida rende invalide le precedenti;
- il sistema deve interagire con l'uomo, che è l'anello più debole della catena in quanto non programmabile.

Rischi digitali per la sicurezza

Esistono caratteristiche intrinseche del mondo digitale in termini di sicurezza:

- **automazione dell'offensiva**, rendendo automatica l'esecuzione dell'attacco. Gli attacchi diventano più semplici da reiterare trovate le vulnerabilità. Si sfrutta il fatto che le violazioni possono anche essere quasi intracciabili. L'automazione degli attacchi pone la privacy dei nostri dati a rischio quando vengono bersagliati i database. Esempi di attacchi automatici sono i microfurti (rubare 1 centesimo da ogni transazione del sistema);
- **assenza di distanza**, in quanto gli attacchi possono essere sferrati a distanza grazie a internet. Questo rende il nostro sistema possibilmente bersagliato da qualunque delinquente, e non bastano le leggi nazionali a proteggerci, ad esempio se l'attacco proviene da un paese in cui non ci sono accordi di estradizione. Alcuni paesi, inoltre, non accettavano meccanismi di crittografia robusti. Per molti stati non esiste il concetto di **sovranità digitale**, ossia il concetto per cui molti servizi non riescono ad esistere senza soluzioni offerte da aziende di altre nazioni. Il nesso tra giurisprudenza e tecnologia è estremamente complicato;
- **propagazione delle tecniche offensive e difensive**. Ormai è elementare cercare exploit in rete, per cui effettuare attacchi è molto più semplice oggi e potrebbe non richiedere lo stesso livello di expertise di un attaccante con skills avanzate;
- **difficoltà di reazione**. All'attacco informatico la reazione non è sempre ovvia (ad esempio, come reagire ad un pagamento non autorizzato sulla propria carta). La migliore reazione è il monitoraggio continuo delle attività, e se c'è un problema si ricerca esso nel log delle attività.

Il gioco della sicurezza

La disciplina di sicurezza è una sorta di gioco perpetuo tra attaccanti e difensori. L'attaccante:

- studia il sistema target;
- cerca vulnerabilità e punti di debolezza;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- cerca o sviluppa exploit;
- ripete questi step.

L'attività offensiva è un ciclo di tentativi. Il difensore:

- installa strumenti di difesa;
- aggiorna il sistema con security patch;
- monitora il sistema;
- ripete questi step.

L'attività difensiva è un ciclo di aggiornamento e manutenzione. Le attività offensive vengono chiamate **red activities**. Le attività difensive vengono chiamate **blue activities**. Quindi il **red team** attacca, il **blue team** difende. Nella realtà questi team sono realistici: le aziende, infatti, specializzano team su tecniche offensive e su tecniche difensive separatamente. Un penetration tester si occupa di svolgere entrambe le attività, in quanto cerca vulnerabilità, utilizzando strumenti come vulnerability scanner, e sfrutta tali vulnerabilità, proponendo poi misure di difesa per mitigare queste.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

2 - Attacchi e attaccanti

Riprendiamo il concetto di **porta** di sistema.

Porte di sistema

Una **porta** è una locazione di rete indirizzabile implementata all'interno del sistema operativo al fine di poter distinguere il traffico. Viene impiegata per indirizzare il traffico ad un servizio. Ogni daemon può essere configurato per servire su una porta differente. A livello offensivo, si effettua il **port scanning** per identificare quali porte sono aperte e quali servizi sono operativi dietro ogni porta aperta.

Un servizio che vuole essere in ascolto su una porta, o che vuole comunicare con un altro servizio tramite un indirizzo IP ed una porta, utilizza le **socket** implementate dal sistema operativo. Una socket è un'interfaccia software tra il livello applicativo e il livello di trasporto, definendo il protocollo di trasporto utilizzato, l'indirizzo IP e la porta di ascolto. Un processo che ascolta su una porta fa **binding** su questa.

Il port scanning può essere più stealth, ad esempio, se si evita il completamento dell'handshake TCP. In generale, **la stealthness non è garantita**: dipende sia da come avviene lo scanning che dal sistema di logging. `nmap`, ad esempio, ha l'opzione `-sS` per scan con SYN, ma non viene mai inviato il SYN ACK, così che la connessione non sia stabilita.

Definizione di attacco

L'attacco è lo sfruttamento di una vulnerabilità presente nel sistema bersaglio. L'esecuzione del trojan analizzato precedentemente non è realmente un attacco: l'attacco consiste nello sfruttare la copia della shell da parte dell'attaccante.
Gli attacchi informatici possono essere di natura diversa. Alcuni esempi sono i seguenti:

- accesso al sistema;
- accesso al sistema per conto di un altro;

- privilege escalation;
- impersonation;
- furto di dati sensibili;
- denial of service (DoS).

Alcuni meccanismi di autenticazione (banalmente la carta di identità) non sono molto efficaci (la carta di identità associa un'anagrafica ad un volto). In generale esiste un binomio **autorizzazione-autenticazione** che impatta ogni sistema. Modificare i permessi di un file, ad esempio, significa modificare il profilo autorizzativo dell'utenza per quel determinato file. La conoscenza del PIN, invece, è autenticazione: ci permette di autenticarci come un utente.

Consideriamo come caso studio la carta d'identità elettronica (CIE). Le carte d'identità hanno un chip contenente la digitalizzazione dell'impronta. Se la CIE venisse letta da un qualunque lettore NFC, il lettore legge i dati del possessore effettivo della carta. Non è detto che chi dà la carta però sia quest'ultimo; per tale motivo, deve arrivare una conferma in tempo reale del possessore. Ciò avviene tramite l'utilizzo dell'impronta digitale. Nella realtà ancora oggi c'è la foto nella carta, al fine di poter autenticare la coppia umano-documento. Il meccanismo di autenticazione, pertanto, non è semplicemente la carta d'identità, ma la coppia persona-carta d'identità. Nel caso della CIE ciò avviene con i dati biometrici.

In generale, la sicurezza è la **prevenzione degli attacchi** e il **corretto funzionamento del sistema**. Non bisogna solo prevenire gli attacchi, ma garantire anche che il sistema funziona. Infatti, un portatile banalmente sicuro da tutti gli attacchi è un portatile spento.

Attacco reale: furto (perdita) di dispositivi

Supponiamo di lasciare un dispositivo incustodito. L'attacco fondamentale è l'accesso al sistema.

Per far sì che l'accesso al sistema sia protetto, basta impiegare una misura di autenticazione, quale PIN, password o misure biometriche. L'accesso al sistema tuttavia potrebbe essere violato comunque, ad esempio prendendo un live usb e montando il file system al fine di scrivere una nuova password. Se il disco è cifrato, bisogna però conoscere la chiave di crittografia.

Supponiamo che l'attaccante ha accesso al sistema. L'attaccante allora vuole sfruttare le funzionalità di questo. Contromisure che è possibile applicare sono meccanismi di autenticazione sulle funzionalità. Ad esempio, il browser può richiedere una master password, o l'applicazione sul telefono richiede l'impronta digitale.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Ottenuto l'accesso alla funzionalità di sistema, l'attaccante sferra un ulteriore attacco per acquisire dati sensibili. Pertanto è sufficente crittografare i dati, ad esempio impiegando un file system crittato come LUKS.

Fallimento delle contromisure

Non è detto che le misure che applichiamo siano in grado di difenderci con successo. Ad esempio, l'accesso al sistema può essere aggirato con un boot di un sistema operativo esterno. Banalmente, anche GRUB tramite recovery mode può permettere un cambio di password montando il file system.

Non sempre, inoltre, si impostano master password nei browser. Anzi, tipicamente tutte le password sono salvate nei browser per comodità.

Le ultime versioni di Windows 11 sono automaticamente con crittografia bitlocker, ma per le installazioni Linux non è sempre detto che il disco sia cifrato.

La sicurezza può essere mista come la protezione di servizi e dati. In particolare la parte di protezione dei dati prende il nome di data protection, e la data protection garantisce la privacy, diritto di ogni cittadino.

Caso studio: privilege escalation su Ubuntu 12.04

In Ubuntu 12.04 è possibile modificare la linea di boot affinché il file system sia scrivibile, modificando `ro` (read-only) in `rw` (read-write) e inizializzare con `init=/bin/bash`. Si ottiene così una shell di root senza la necessità di password.

Chiaramente questo potrebbe essere utile per quando l'utente si scorda la password. Molto spesso il bilanciamento tra usabilità e sicurezza può esporre il sistema a vulnerabilità.

Esempio 2: pirateria digitale

Un altro esempio di attacco reale è la pirateria digitale, che in molti casi consiste in furti di proprietà intellettuale. Una possibile contromisura è il **watermarking**, una tecnica che inserisce nei file digitali dei dati identificativi univoci (spesso invisibili o semitrasparenti). Questi identificatori permettono di risalire alla fonte della ridistribuzione illegale e, se integrati con tecnologie **DRM** (**Digital Rights Management**), consentono di bloccare l'accesso non autorizzato ai contenuti, come nel caso dello streaming illegale.
Un altro caso è il furto d'identità, come il login come un altro utente. Una contromisura è introdurre

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

un meccanismo di autenticazione.
Infine, esiste anche il furto di marchio, in cui un attaccante fa un uso indebito dell'identità di un'azienda o di un brand per ingannare altri utenti. Ad esempio, il domain squatting consiste nell'utilizzo di domini che differiscono per leggeri typo dal dominio originale, come `micor-soft.com`. Un attaccante potrebbe comprare questo dominio per ospitare una pagina malevola uguale al sito originale di Microsoft, così da estrarre informazioni da altri utenti.

Esempio 3: sniffing

Lo **sniffing** consiste nell'intercettazione di dati all'interno della rete. Oggi lo sniffing è tamponato dall'utilizzo, a livello di applicazione, di protocolli mediante i quali le comunicazioni sono cifrate.

Tassonomia degli attacchi

Classifichiamo gli attacchi in:

- attacchi criminali, che hanno come obiettivo un tornaconto immediato. Si espandono in:
  - frodi;
  - attacchi distruttivi, cioè distruzione delle risorse;
  - furti di identità;
  - furto di marchi registrati;
  - furti di proprietà intellettuale;
- attacchi alla privacy, quali spionaggio, sorveglianza, estrazione di informazioni da database e analisi del traffico;
- attacchi basati su sistemi legali. L'attacco, infatti, potrebbe non essere puramente tecnico, ma può fare leva sull'ignoranza della giuria in ambito di cybersecurity, portando al fallimento nel determinare il responsabile dell'attacco;
- attacchi a scopo pubblicitario, più precisamente che bersagliano la reputazione di certe figure pubbliche. Un esempio è la negazione di servizio (DoS) di un'azienda o l'esfiltrazione di informazioni segrete da essa (e.g. *Embrace, Extend and Extinguish*, piano aziendale di Microsoft).

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

3 - Proprietà di sicurezza

Esistono diverse proprietà di sicurezza, disposte su diversi livelli. Piuttosto che stabilire se un sistema è sicuro, è necessario stabilire se un sistema gode di determinate proprietà. Al livello più basso esistono tre proprietà fondamentali: **confidenzialità**, **autenticazione** e **integrità**, note come **triade CIA** (autenticazione andrebbe sostituita con "availability", ma dettagli…). Sono proprietà fondamentali e riconosciute da standard internazionali sulla sicurezza.

La suddivisione a livelli delle proprietà non rappresenta una gerarchia di importanza, bensì una dipendenza logica. La disponibilità, ad esempio, è al livello 2 poiché non ha senso garantire disponibilità di un sistema che non è solido di per sé. L'autorizzazione è al livello 3, invece, poiché richiede autenticazione ed un sistema stabile (disponibilità) in cui implementare il controllo d'accesso.

In generale non vi è un numero finito di proprietà di sicurezza in quanto queste sono in continua evoluzione. Ci sono proprietà emergenti che accompagnano i nuovi servizi.

Confidenzialità (o segretezza)

La proprietà di **confidenzialità** garantisce che l'informazione non sia rilasciata ad entità non autorizzate a conoscerla. Così come un segreto è noto solo ad alcuni individui, i dati devono essere a disposizione solamente di chi è autorizzato a leggerli. Esistono alcune misure per garantire la confidenzialità, quali protocolli crittografici per la comunicazione di segreti o metodi di steganografia per nascondere contenuti in immagini. In questo contesto la steganografia è semplicemente nascondere il dato, ma ha certi limiti: l'attaccante potrebbe in maniera sistematica studiare i bit meno significativi per scoprire se un messaggio è nascosto.

In questo contesto si può parlare di **segreto condiviso**. Se un segreto è noto solamente ad un individuo, non sarà di grande utilizzo in un ambito distribuito. In un contesto distribuito la condivisione dei segreti permette lo scambio di dati sicuro fornendo un vantaggio tra i conoscitori di questo segreto.

Autenticazione

L'**autenticazione** è un processo che verifica che le entità siano esattamente chi dichiarano di essere. Abbiamo già visto meccanismi semplici di autenticazione, ad esempio la conoscenza del PIN o la coppia individuo-carta d'identità. Un altro esempio è l'autenticazione di un URL con il browser: ogni browser autentica l'URL verificando la validità del certificato del sito. Questa proprietà non è scontata: inizialmente, infatti, non vi era autenticazione sulla gran parte di siti web.

La realtà è che oggi si ha necessità di autenticazione. Servizi che non presentano meccanismi di autenticazione potrebbero essere utilizzati per la criminalità, ad esempio SMS senza account (questo servizio esisteva nel '99). Non esistono servizi web significativi che non richiedono l'autenticazione oggi.

Per motivi di sicurezza internazionale, i meccanismi di autenticazione sono anche fondamentali per il tracciamento di attività illegali.

Integrità

La proprietà di **integrità** garantisce che l'informazione non sia modificata da entità non autorizzate. Misure per garantire l'integrità sono il checksum e le firme digitali. Il checksum è una misura di integrità se non c'è attività malevola esplicita: è impiegato per verificare la presenza di errori durante la trasmissione. Nulla vieta che un attaccante attivo alteri il messaggio in transito per modificare il checksum. Un modo per applicare il checksum è tramite il calcolo di un digest mediante funzione hash.
Il meccanismo più sicuro per garantire autenticità è tramite **Message Authentication Code** (**MAC**). Allegato al messaggio si impiega una funzione hash che prende in input il messaggio ed una **chiave segreta** nota ad entrambe le parti comunicanti. In questo modo anche se un attaccante esterno prova a modificare sia messaggio sia checksum, in realtà non potrà mai illudere il destinatario: poiché la funzione hash è calcolata in virtù di una chiave privata, e tale chiave è nota solo a mittente e destinatario, l'attaccante non potrà mai produrre il digest corretto. Questo non solo garantisce integrità, ma anche autenticazione: l'impiego della chiave segreta permette di capire che il messaggio è stato inviato dal mittente originario.
Un'alternativa è l'impiego di una **firma digitale**. La principale differenza con il MAC è l'utilizzo di una crittografia asimmetrica: si utilizza una chiave privata dal sender per firmare il messaggio e una chiave pubblica dal receiver per verificare il mittente. Oltre a garantire integrità e autenticità, garantisce anche il **non-ripudio**: il sender non potrà negare di aver inviato il messaggio.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Privatezza

La **privatezza** (meglio nota come **privacy**) è il diritto di un'entità di rilasciare o meno i propri dati personali ad altre entità. La privatezza può essere interpretata come una regola secondo cui ogni individuo può determinare le sorti della propria **segretezza**. Le policy permettono l'identificazione degli elementi segreti, ed è scelta del singolo divulgare o meno tale segreto.
Consideriamo, ad esempio, una password. I metodi di sicurezza sulle password non discriminano le proprietà di segretezza o privatezza. Una policy aziendale potrebbe stabilire che nessun dipendente debba divulgare la password: quest'ultima, pertanto, è segreta ma non è privata, in quanto non abbiamo alcun diritto di rilasciare questa informazione.
Esiste un caso specifico di privatezza: la nostra identità. L'istanziazione di privatezza sull'identità prende il nome di **anonimato**, ma ciò non è sempre garantito: ad esempio, le carte di pagamento associano l'acquisto a chi lo effettua. Solitamente si cerca di garantire lo **pseudo-anonimato**, ossia un meccanismo che garantisce un anonimato robusto, ma comunque reversibile. Un esempio semplice è il soprannome di una persona.

Anonimato

Definiamo formalmente l'**anonimato**. L'anonimato è il diritto dell'iniziatore di una transazione di rilasciare o meno la propria identità ad altre entità. Oggi i meccanismi di gestione della sessione tramite cookie minano l'anonimato: il cookie, infatti, è in grado di associare il nostro dispositivo ad una sessione per motivi di comodità. Le modalità di incognito di browser come Chrome in realtà non garantiscono l'anonimato totale: il client non conserverà la cronologia e non invierà i cookie memorizzati, ma il server può comunque triangolare la richiesta in quanto conosce l'indirizzo di chi invia le richieste. Inoltre, per legge l'ISP è obbligato a loggare il traffico degli utenti.

L'utilizzo di VPN è un altro esempio. La VPN viene principalmente impiegata per ottenere un indirizzo IP locale alla sottorete del destinatario. Tuttavia, non si ottiene anonimato: se il traffico è loggato e il provider si trova in una nazione che prevede la conservazione dei dati, per legge se richiesto per indagini il provider dovrà fornire le informazioni del traffico. Alcune VPN sono, infatti, con no-log policy.

Si potrebbe pensare che HTTPS garantisca anonimato: il contenuto, infatti, è crittografato. Questo non è vero, in quanto l'informazione relativa alla presenza della transazione è comunque presente: HTTPS cifra solo il contenuto, ma si può sapere chi invia contenuti, chi li riceve e quando avviene la transazione.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Un modo per ottenere un buon grado di anonimato è mediante **The onion router** (meglio noto come **Tor**). Tor permette di creare una serie di anelli per nascondere il nesso tra il reale mittente e il destinatario di una richiesta: piuttosto che farla andare diretttamente al destinatario, si fa passare da più nodi, così da creare strati di IP diversi rispetto a quello del vero host mittente. Tor porta l'anonimizzazione a livello applicazione. Uno strumento simile utilizzato per ottenere un discreto grado di anonimato è **proxychain**.

Esempio: Written Authenticated Through Anonymous v2 (WATA2)

Supponiamo di voler garantire delle proprietà di sicurezza per l'esame universitario. Più precisamente, vogliamo che ogni studente sia autenticato, ma in fase di marking i compiti siano anonimi.

Come si può realizzare un protocollo per l'esame universitario scritto tale che ciascun compito sia autenticato ma anonimo? L'autenticazione del compito deve tutelare il docente per prevenire imbrogli dello studente, ma a tutela dello studente bisogna garantire l'anonimato del compito per prevenire votazione iniqua. Il protocollo **Written Authenticated Through Anonymous** (WATA) è un esempio di ciò. Seguendo questo protocollo, lo studente viene autenticato dal docente, e il compito è anonimo al docente, in modo tale da prevenire votazione iniqua ma garantire che la procedura di esame sia verificata (ossia, non ci sono impersonatori). Chiaramente, il rispetto del protocollo da entrambe le parti è fondamentale per garantire le proprietà di sicurezza.

In generale, vorremmo che fosse anonimo il compito, piuttosto che lo studente. Si adotta la seguente strategia: si adotta un foglio di compito con un bar-code e un talloncino. Sul talloncino ci sono nome, congome, matricola. Lo studente terrà il talloncino, che viene prima firmato dal docente per evitare che un altro studente prepari un finto talloncino con un bar-code da sovrapporre sul compito:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

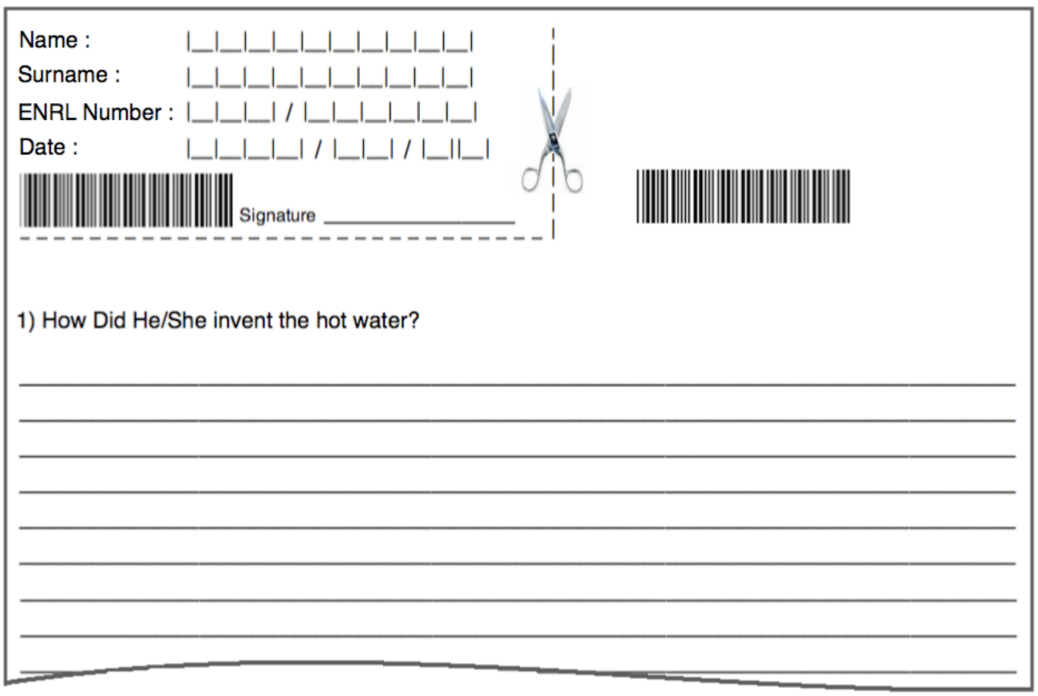


Figure 1: Foglio WATA2

Definiamo adesso il protocollo di sicurezza. Il protocollo è diviso in quattro fasi:

- preparazione;
- testing;
- marking;
- notification.

Identifichiamo tre attori:

- examiner, ossia il docente esaminatore;
- invigilator. L'invigilator può avere un ruolo funzionale comodo, così da facilitare il lavoro dell'esaminatore. Assumere che il vigilator sia del tutto partizionato può essere troppo forte. L'invigilator, in essenza, è un sorvegliante. Se un solo attore fa da examiner e da invigilator e le proprietà di sicurezza sono ancora valide, il protocollo è ancora robusto, ossia il protocollo è robusto nello scenario in cui examiner e invigilator sono collusi (ossia, il fatto che examiner e invigilator condividano informazioni). Pertanto, dal punto di vista della sicurezza le proprietà sono anche rispettate con due ruoli, ossia examiner e candidate, e l'invigilator ha solamente un ruolo funzionale comodo;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- candidate, ossia lo studente.

La fase iniziale di **preparazione** vede come protagonista il docente. Sono presenti tre database: 1. il database delle domande; 2. il database per la conservazione dei voti; 3. il database per lo storico degli esami (in generale vorremmo sempre loggare tutto).

Analizziamo formalmente:

$$quest \leftarrow_{\mathcal{R}} DB_q$$

$$id_test \leftarrow_{\mathcal{R}} \Sigma^n$$

$$DB_m \leftarrow (id_test, _)$$

$$\text{test} \leftarrow \text{print_aside}(id_test, auth_form, id_test, quest, answ_form)$$

$$\text{test}_{\text{signed}} \leftarrow \text{bio_sign}(M, \text{test})$$

Il primo step della fase di setup è la generazione del test e dell'identificativo di n simboli del foglio d'esame. Inizializziamo una tupla per il voto: il primo elemento è l'id dell'esame, il secondo è il voto. Per ora, il voto è sconosciuto e lo scopo è assegnare a tale test una valutazione.
`print_aside` genera il foglio di test. Si stampa a sinistra l'id del test e un form di autenticazione (la sezione del talloncino), e nelle parti restanti del foglio nuovamente l'id del test, le domande e il form di risposta ad esse. L'id del test e il form di autenticazione formano assieme il **token**.
Il token va irrobustito per evitare attacchi mirati a falsare l'abbinamento tra un'anagrafica scritta nel form di autenticazione e l'altra nel foglio del test. Per controbattere questi problemi si impiega la **firma a cavallo** sul token. La firma a cavallo sigilla l'associazione detta precedentemente ed è rappresentata dalla firma bio_sign. M indica chi firma (la M sta per "marker" ed è un refuso).
Il processo di preparazione viene ripetuto per ogni compito. Il numero di firme avviene per ogni singolo compito, non per singolo foglio di domanda, anche perché, una volta stabilita l'associazione tra l'id del token e fogli d'esame, c'è bisogno di un solo token per compito. Quindi se ci sono 150 compiti e 3 domande per compito, avremo 450 fogli ma sempre 150 firme.

La seconda fase è il testing. I test firmati vengono passati all'invigilator, che consegna in maniera randomica i test agli studenti. A questo punto, il candidato genera $\text{test}_{\text{auth_filled}} \leftarrow$

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

fill_auth_form$(\text{test}_{\text{signed}})$. Quando il candidato compila l'auth form e si ha la verifica di autenticazione, egli copre l'id del test col braccio per garantire l'anonimato. Successivamente, avviene una fase di verifica:

- si controlla la validità del documento;
- si controlla che lo studente è il proprietario del documento;
- si controlla se il candidato è presente nella lista di chi si è registrato per l'esame;
- si controlla se le informazioni nel documento combaciano con le informazioni presenti nell'auth form. Lo studente nasconde sempre l'id del test.

A questo punto il candidato compila il compito rispondendo alle domande, e separa il token dal foglio di risposte, consegnando entrambi i pezzi separatamente all'invigilator. La consegna del foglio di risposte anonimo è quindi randomizzata. Poi l'invigilator inoltra i fogli di risposte all'examiner:

$$\text{test}_{\text{filled}} \leftarrow \text{fill_answ_form}(\text{test}_{\text{auth_filled}})$$

$$[\text{token}, \text{a_test}] \leftarrow \text{split}(\text{test}_{\text{filled}})$$

Dopo che il testing è finito, inizia la fase di **marking**. Avendo i fogli di test, nel database di mark DB_m l'examiner inserisce i mark aggiornando le entry in virtù dell'id del test:

$$DB_m \leftarrow_{id_test} (*, mark)$$

Infine, c'è la fase di **notification**. Il candidato fornisce il proprio token all'esaminatore. Il docente fa una query al DB_m con il token e poi registra l'esito nel database della history DB_m:

$$(*, mark) \leftarrow_{\text{id_test}} DB_m$$

$$DB_h \leftarrow_{\text{id}} (*, course, mark)$$

Tale step è pubblicamente ispezionabile (ossia è noto a tutti il voto in questo step, ed è pubblica la fase di votazione), altrimenti l'esaminatore potrebbe barare fino a tale step per modificare il voto data la sua bias. Inoltre, il voto non è mai noto fino alla scansione.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Non-ripudio

La proprietà di non ripudio garantisce che l'entità non possa negare la propria partecipazione ad una transazione con uno specifico ruolo. La proprietà di non-ripudio non è garantita dall'autenticazione: il solo fatto che si è autenticati in una piattaforma non fornisce alcuna prova di terze parti che una determinata entità ha commesso un'azione. Ad esempio, un acquisto in contanti da parte di un utente non può fornire alcuna prova; invece, l'acquisto mediante carta fornisce esistenza della transazione, in quanto vi è una prova effettiva. Un altro esempio è la firma, che certifica il fatto che un individuo ha letto e compreso un documento, e garantisce anche che l'individuo **non** possa rifiutare di averlo fatto. Questo concetto è esteso al mondo digitale sotto forma di firma digitale. Lo stesso concetto è utilizzato per la PEC.

Relazioni logiche tra autenticazione, anonimato e non-ripudio

Vigono le seguenti relazioni logiche tra autenticazione, anonimato e non-ripudio:

1. l'autenticazione implica l'assenza di anonimato;
2. l'anonimato implica l'assenza di autenticazione;
    - dalla $1.$ e $2.$ si deduce che autenticazione è congruo ad assenza di anonimato;
3. autenticazione **non** implica non-ripudio;
4. non-ripudio implica autenticazione;
5. anonimato implica l'assenza di non ripudio;
    - essa deriva dalla 2 e dalla contrapposizione (cioè $\neg q \implies \neg p$) della 4;
6. non-ripudio implica l'assenza di anonimato;
    - essa deriva dalla 4 e dalla 1.

Disponibilità

Il concetto di **disponibilità** è legato alla negazione del servizio (DoS, Denial of Service). Tale proprietà garantisce che il sistema sia operante e funzionante in ogni momento. La disponibilità è garantita mediante alcune misure.

Un'idea potrebbe essere garantire la disponibilità limitando l'accesso ad utenti autenticati. Non vi è un proprio nesso tra autenticazione e disponibilità: un abbondanza di utenti autenticati può causare

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

lo stesso un sovraccarico del sistema. Inoltre, la procedura di autenticazione potrebbe essere gravosa. Pertanto non è una soluzione adeguata. In alternativa si può limitare l'accesso per impegnare computazionalmente il chiamante complicando l'accesso.

La soluzione migliore è quello di scalare il sistema orizzontalmente utilizzando load balancer. In generale non è una soluzione finale, e garantire la disponibilità è ancora una sfida.

Controllo d'accesso

Al livello 3 osserviamo il **controllo d'accesso**. Il controllo di accesso è sinonimo di **autorizzazione** e garantisce che ciascun utente abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato.

Il controllo d'accesso ha senso solo se è autenticato l'utente: se non si conosce l'identità dell'utente, allora non si può capire di quali permessi questo gode. Quindi l'autorizzazione esiste in virtù dell'autenticazione. Ma l'autenticazione in sé non basta per risolvere il problema dell'autorizzazione. L'autenticazione è pertanto uno strumento necessario per l'autorizzazione, ma non il fine. L'autorizzazione è gestita poi mediante politiche di sicurezza, che vengono applicate con un'opportuna implementazione. Quindi sono tre step: autenticazione, definizione delle politiche e implementazione di queste.

Esempi di politiche di sicurezza

1. un utente ha il permesso di leggere su qualunque file pubblico;
2. un utente ha il permesso di scrivere solo sui file pubblici di sua proprietà;
3. un utente ha il divieto di fare il dowgrade di un file;
4. un utente ha l'obbligo di cambiare la propria password quando scade;
5. un utente segreto ha il permesso di leggere e scrivere su un qualunque file non pubblico;
6. un amministratore ha il permesso di sostituire un qualunque file con una sua versione più obsoleta;
7. un utente che non cambia la sua password scaduta (negligente) ha il divieto di compiere qualunque operazione;
8. un utente che non cambia la sua password scaduta (negligente) non ha discrezione di cambiarla.

Notiamo come in generale le singole policy siano semplici, ma l'insieme di policy in sé potrebbe essere inconsistente. Ad esempio, i punti $7, 8$ sono contraddittori. La policy 7 infatti esplicita che un

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

utente che non cambia la propria password non può compiere alcuna operazione, ma la 8 specifica che l'utente non ha la possibilità di cambiarla se negligente. Quindi l'utente rimane bloccato dal sistema. Lo stesso vale per la 6 e la 3.

Per definire cos'è un'inconsistenza, è necessario capire come vengono gestite e formulate le policy.

Gestione di policy

In generale le policy vengono gestite mediante **ruoli**, per cui si parla di **Role Based Access Control** (**RBAC**). Il ruolo **prescinde** da chi lo detiene, così che il ruolo stesso determini l'insieme di autorizzazioni. Inoltre, i ruoli possono essere in relazioni di inclusione: gli amministratori, ad esempio, sono anche utenti. Le regole applicate all'utente vengono applicate anche agli amministratori, ma non il viceversa.

Definiamo pertanto gli elementi di una policy:

- **ruoli**, gruppi di policy;
- **utente**, ossia qualunque entità che ricopra un certo ruolo;
- **operazioni**, quali leggere e scrivere su file;
- **modalità**. Possiamo identificare come modalità base l'**obbligo**. Dall'obbligo possiamo derivare altre tre modalità derivate, ossia il **divieto** (il divieto di x è l'obbligo di non fare x), il **permesso** (il permesso per x è il non obbligo di non fare x), e il **discrezionale** (la discrezione per x è il non obbligo di fare x). L'unica forzatura in questo discorso è che fondiamo tutto sull'obbligo, ma nulla vieta a fissare come modalità base il permesso.

$$Vietato(x) = Obbligatorio(\neg x)$$

$$Permesso(x) = \neg Obbligatorio(\neg x)$$

$$Discrezionale(x) = \neg Obbligatorio(x)$$

Le modalità per un ruolo sono definite in maniera minimale per i ruoli più ampli, quali l'utente generico, e poi vengono definite più nel dettaglio per gli utenti con più autorizzazioni mediante un approccio incrementale. In questo modo si genera un diagramma di Eulero Venn dove il macro-insieme è il ruolo meno privilegiato e all'interno vi sono insiemi di modalità più dettagliati. Ad esempio:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- l'utente è un insieme di modalità base;
- l'amministratore è un insieme di più modalità, per cui è incluso nell'utente in quanto un amministratore gode delle stesse modalità del'utente più altre definite in virtù di tale ruolo.

Si può dire, pertanto, che i ruoli interni ereditano le modalità dei ruoli esterni. Esiste anche un'altra possibilità: gestire i ruoli in maniera partizionata. Ma questo approccio può essere molto dispersivo se ci sono policy grandi. L'approccio incrementale è il più adatto.

Inconsistenze

Una policy può avere delle **inconsistenze**. Le inconsistenze sono di due tipi: **contraddizione** o **dilemma**. La contraddizione è la seguente inconsistenza:

$$Obbligatorio(x) \land \neg Obbligatorio(x)$$

Il dilemma, invece, è la seguente inconsistenza:

$$Obbligatorio(x) \land Obbligatorio(\neg x)$$

Nelle policy precedentemente citate:

- abbiamo una **contraddizione** tra il divieto di fare il downgrade di un file per un utente e il permesso di sostituire un qualunque file con una sua versione più obsoleta per un amministratore. Poiché l'amministratore eredita anche il permesso dell'utente, ovvero l'amministratore è un utente, allora la coesistenza delle due regole implica sia il permesso sia il divieto sulla medesima operazione. Per identificare una contraddizione, pertanto, è sufficiente identificare una coppia divieto-permesso su un'operazione x;
- abbiamo un **dilemma** tra il divieto di compiere operazioni per un utente negligente e l'impossibilità di poter cambiare la password per un utente negligente. Il secondo divieto, pertanto, è un obbligo di non cambiare la password, mentre il primo indica che è obbligatorio cambiare la password in quanto non ha libertà.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Mandatory Access Control e Role-Basec Access Control

Data la complessità delle policy, si è deciso di formalizzare l'utilizzo delle policy. I **modelli di controllo di accesso** sono sistemi che permettono agli amministratori di gestire al meglio le politiche.

Il **Mandatory Access Control** è il modello più stretto per l'organizzazione delle policy. MAC è impiegato per la realizzazione delle politiche mandatorie, ovvero basate sulla modalità "Obbligatorio". Inoltre, tali policy non sono modificabili e sono fornite da un solo individuo.

In molti sistemi operativi non si utilizza MAC, in quanto le policy di tali sistemi si basano sul concetto di permesso, non di obbligo. Esistono, però, sistemi operativi di alta sicurezza come SELinux e in generale sistemi operativi **hardened**, ossia equipaggiati con un sistema mandatorio per la gestione delle operazioni.

Il sistema più naturale per la gestione delle autorizzazioni è la **Role-Based Access Control**, ossia la gestione dell'accesso basata sul ruolo. Un utente si autentica, viene riconosciuto con il suo ruolo e grazie ad esso vengono assegnati certi permessi e certe modalità sulle varie operazioni. Le politiche non sono mandatorie.

Meccanismi implementativi

Comprese le policy, è necessario capire il modo con cui è possibile fare enforcing di queste. È sufficiente prendere una matrice, sulle righe elencare i ruoli e sulle colonne gli oggetti su cui fare le operazioni. L'elemento di posto (s, o) indica i permessi di s sull'oggetto o. Ad esempio:

Utenti-Oggetti	File1	File2	File3	Program1
Alice	RW	R	-	X
Bob	R	-	RW	-
Charlie	-	R	-	RX

Tale matrice prende il nome di **Access Control Matrix** (**ACM**). Questo approccio è tipico dei sistemi operativi basati sul kernel Linux.

Chiaramente la matrice può essere letta per colonna. Ogni colona della ACM corrisponde ad un oggetto, ottenendo i permessi su specifici oggetti. Si ottiene in questo modo la **Access Control List** (**ACL**). Diversamente, se la matrice è letta per righe, si ottiene una lista degli oggetti e relativi permessi per l'utente corrispondente a quella riga: questa lista prende il nome di **Capability List**.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Tassonomia di attaccanti e threat modeling

Importante nella definizione di un modello di minaccia è l'identificazione dell'**attaccante**. L'attaccante sferra un attacco per diverse motivazioni:

- ricchezza;
- furto di informazioni sensibili;
- gloria;
- divertimento;
- attivismo.

In generale, esistono diverse motivazioni sociali per cui un attaccante è spinto ad effettuare un attacco. Tuttavia, ciò non è nel nostro interesse.

L'attaccante viene solitamente classificato in virtù di diritti, risorse a disposzione, esperienza (attaccante esperto o inesperto). Possiamo riconoscere, inoltre, diversi tipi di attaccanti:

- un hacker esterno;
- un attaccante interno, ad esempio una figura compromessa o semplicemente un attore interno che in maniera inconsapevole diventa un attaccante (accidentalmente espone dati sensibili);
- spionaggio industriale;
- servizi segreti;
- organizzazioni criminali e terroristiche;
- organi di difesa.

La definizione dell'attaccante è fondamentale per la valutazione dell'adeguatezza delle misure di difesa. Si potrebbe pensare di valutare la sicurezza di un protocollo in tre scenari:

- **white-box**, in cui l'attaccante ha **massima conoscenza** sul bersaglio;
- **black-box**, in cui l'attaccante ha **pressoché nessuna conoscenza** sul bersaglio;
- **gray-box**, in cui l'attaccante ha **qualche conoscenza** sul bersaglio.

Questo in realtà non è sempre necessario. Ad esempio, è sufficiente valutare la sicurezza nel caso white-box: se l'attaccante non riesce ad aggirare i protocolli di sicurezza con il massimo numero di conoscenze, allora certamente si può considerare che l'attaccante non potrà farlo con conoscenza gray-box o black-box. Tuttavia, se nel caso white-box l'attaccante riesce ad aggirare le misure di difesa, si dovrebbe valutare anche il caso black-box.

Siamo in grado adesso di poter introdurre un **modello di attaccante**. Il modello di attaccante specifica le capacità offensive di un preciso attaccante. Ogni affermazione di sicurezza ha senso solamente per

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

un modello: cambiare il modello di minaccia potrebbe cambiare il valore di verità sull'affermazione di sicurezza. Il modello nel caso peggiore è un modello massimamente offensivo. Fissato il modello, potremmo fare un'analisi generale.

Modello di attaccante Dolev-Yao (DY)

Il threat model Dolev-Yao definisce un **attaccante unico e superpotente in grado di controllare l'intera rete ma non in grado di violare la crittografia**. L'attaccante di tale modello di minaccia coincide con un insieme di attaccanti collusi (i.e. con lo stesso obiettivo). Questo significa che in una rete di nodi, **tutti i nodi intermedi siano malevoli e collaborino contro gli endpoint**. In questo modo, i nodi intermedi controllano l'intera rete. Con la nozione di "controllare" si intende la possibilità di poter ricevere i messaggi scambiati, utilizzarli per costruirne nuovi e leggere il traffico. Il controllo è, pertanto, attivo.

La debolezza di questo modello, tuttavia, risiede nella impossibilità di discernere se il problema riguarda la struttura del protocollo o meno. Se un protocollo si rompe contro un attaccante superpotente, i problemi possono essere di due tipi:

- la crittografia impiegata è debole;
- l'infrastruttura del protocollo è debole.

In questo modello, tuttavia, **si assume che la crittografia sia perfetta**. L'avversario, in questo caso, non ha alcuna conoscenza sulle chiavi utilizzate per crittare i messaggi, per cui l'unica cosa che può fare è mandare messaggi agli utenti del sistema e intercettare e bloccare messaggi scambiati.

Tale modello viene solitamente impiegato per i seguenti motivi:

- dimostrare che un protocollo è logicamente valido;
- ricercare in maniera automatica gli attacchi;
- astrarre i dettagli crittografici di basso livello;
- giustificare i flussi dei messaggi e del protocollo.

Questo modello nasce dalla realtà degli anni '80 dove le reti erano impiegate da pochi centri scientifici.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Modello di attaccante General Attacker (GA)

Il modello di attaccante muta in quanto cambia la realtà. Le comunicazioni mobili cominciano ad essere più comuni, nascono le reti domestiche e lo scenario d'attacco cambia: **assumere che l'intera rete sia malevola contro un unico bersaglio è restrittivo**. Potenzialmente di attaccanti massimamente offensivi sono più di uno, e per essere massimamente offensivi possono anche lavorare in maniera indipendente.

Il modello di minaccia **General Attacker** (**GA**, introdotto in *Validating Security Protocols under the General Attacker*) prevede che ogni partecipante ad un protocollo sia un **attaccante DY che non collude con altri attori**. In questo modello di minaccia ha senso continuare l'analisi di un protocollo anche **dopo** che l'attacco è stato sferrato. Inoltre, apre le porte ad altri scenari, come la possibilità di sferrare **controattacchi** (*retaliation attack*) o di **anticipare** l'attacco con un altro attacco (*anticipation attack*) nel momento in cui un attaccante è in competizione con un altro.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

4 - Forme di autenticazione

Le forme di autenticazione sono molteplici:

- **autenticazione utente-computer**. È la più comune ed è usata per accedere ad un sistema (password);
- **autenticazione computer-computer**, indipendentemente dall'utente di questi (IP, MAC);
- **computer-utente**: fondamentale per un computer remoto e in generale quando si inserisce materiale privato su un sito. Si vuole, ad esempio, verificare che il sito è fidato per l'inserimento delle informazioni private;
- **utente-utente**, abbastanza rara e tipica di protocolli quali Kerberos.

Misure di autenticazione utente-computer

L'autenticazione utente-computer può essere basata su:

- **conoscenza** di segreti prestabiliti e precondivisi (password, PIN). È la forma più facile da implementare, sebbene vi siano rischi nell'impiego di una password:
  - **guessing**, indovinare la password mediante attacchiforza bruta, dizionario o semplicemente a tentativi;
  - **snooping**, sbiricare la password mentre viene digitata;
  - **spoofing**, estrarre la password tramite falsa interfaccia di login;
  - **sniffing**, intercettare la password durante la trasmissione (eaves-dropping);
- **possesso** di dispositivi magnetici o elettronici (smart card). L'oggetto dev'essere difficile da replicare e può memorizzare informazioni più particolari:
  - uno **smart token** ha un meccanismo di autenticazione ad esso mediante PIN (alcuni hanno semplicemente un pulsante, senza autenticazione). Genera un segreto speciale noto come **One-Time Password** (**OTP**). L'autenticazione al web server è così basata su possesso e conoscenza. Senza password per lo smart token, l'autenticazione può essere basata su possesso esclusivo del token, ma è vulnerabile a snooping. Lo smart token

si sincronizza in maniera computazionale al server dalla fabbrica, ossia in base ad un algoritmo server e token sono sincronizzati (i.e. utilizzano un algoritmo di generazione pseudocasuale a seed comune e orologi sincronizzati);

- **biometria** di caratteristiche fisiche dell'utente (impronte, iride, tono di voce). Le caratteristiche fisiche sono esternamente visibili, quindi esterne al corpo, e misurabili. La biometria ha alcuni problemi:
  - copia dell'impronta digitale;
  - campionamento della voce;
  - clonazione del genoma.

  Quindi un problema è l'utilizzo di caratteristiche esterne che sono facili da misurare, ma anche facile da smarrire. L'impronta, poiché esterna, è molto più semplice da scoprire. Inoltre, è meno resistente alla coercizione (cioè costringere la vittima a usare l'impronta). Infine, il quarto problema risiede nel **campionamento** dell'informazione biometrica: poiché caratteristiche fisiche vengono memorizzate con un elaboratore, si definisce un template che media le misurazioni e rappresenta la caratteristica. Per tale motivo, **a meno di una tolleranza prestabilita** la misurazione fornita viene accettata. Quindi il riconoscimento biometrico è più impreciso; tuttavia è molto comodo.

L'autenticazione opera su una **prerogativa dell'individuo** e solitamente vengono utilizzate combinazioni di criteri (bancomat, opera con possesso della card e conoscenza del PIN).

Attacchi tipici alle password sono i seguenti:

- **attacchi standard** con password relative all'utente, facendo leva su hobby, nomi, parenti e altro;
- **attacco dizionario**. Si impiega una wordlist o un dizionario. Una wordlist è un sovrainsieme di un dizionario. Si prende un dizionario e lo si arricchisce prendendo una parola del dizionario e modificandola;
- **attacco forza bruta**. Si verificano esaustivamente le combinazioni possibili dato un certo charset e la lunghezza della password. Se C è il charset e n la lunghezza della password, si ha che le combinazioni possibili sono $|C|^n$.

Alcune contromisure sono le seguenti:

- **controlli sulla password**, ad esempio soglie sulla lunghezza minima, presenza di caratteri speciali, e così via. Di norma, si utilizzano almeno 8 caratteri, non parole del dizionario e non usare la stessa password per autenticazioni diverse;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- **controllo sul numero di inserimenti**, limitando il numero di tentativi, pena il blocco del sistema;
- **uso di CAPTCHA** (**Completely Automated Public Turing Test To Tell Computers and Humans Apart**). Il sistema richiede la risoluzione di un captcha per verificare che il tentativo di login venga da un umano. Recentemente, un algoritmo basato su Google Street View viola il 99% delle captcha alfanumeriche. Google pertanto propone reCAPTCHA, dove semplicemente prevede un click per dimostrare di essere umani.

Un problema risiede nella memorizzazione delle password. L'approccio tipico risiede nella memorizzazione di queste in un database. MIT propose nel 1960 il **Compatible Time Sharing System** (**CTSS**), il primo sistema operativo general-purpose che permetteva di eseguire più lavori batch contemporaneamente in time sharing. In CTSS, le password sono memorizzate in chiaro su un file protetto da politiche di sicurezza.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

5 - Crittografia

La crittografia è una scienza il cui obiettivo è trasformare un'informazione sintatticamente preservandone la semantica. Il messaggio binario m viene criptato in un crittogramma c, e c può essere decriptato nuovamente in m. Formalmente, il messaggio in chiaro m viene cifrato in c utilizzando un algoritmo crittografico e una **chiave** di crittografia. Un parametro chiave dev'essere utilizzato per tradurre c in m. L'algoritmo di crittografia, inoltre, è **pubblico**: la sicurezza non è garantita dalla segretezza dell'algoritmo impiegato, bensì dall'ignoranza della chiave.

La crittografia ha radici antiche, con il primo cifrario noto ai tempi dei Romani, ossia il **cifrario di Cesare**. Fino agli anni '70, inoltre, la cifratura era **simmetrica**, ossia la stessa chiave veniva impiegata sia per cifrare che per decifrare i messaggi. La variante **asimmetrica** nasce nel 1977 e prevede l'utilizzo di una chiave pubblica per cifrare e una chiave privata per decifrarli.

Crittosistema

Un **crittosistema** è una coppia di algoritmi con i dettagli dei loro input:

- $\mathcal{E}$ per criptare un messaggio e produrre un crittotesto;
- $\mathcal{D}$ per decriptare un crittotesto e produrre un messaggio.

Ogni algoritmo ha arietà due, ossia ha come parametri un **testo** e una **chiave**:

- la chiave k viene utilizzata per codificare il testo m e produrre il crittotesto c, ossia $c = \mathcal{E}(k, m)$;
- la chiave k' viene utilizzata per decifrare il crittotesto c e ottenere il messaggio in chiaro m, ossia $m = \mathcal{D}(k', c)$.

Non necessariamente $k = k'$, ma in tal caso la crittografia è **simmetrica**.

Ad esempio, Alice ha un messaggio da inviare a Bob in un canale di comunicazione non protetto. Pertanto utilizza la chiave k per cifrare il messaggio m, e invia il crittotesto a Bob. Bob utilizzerà la chiave k' per decifrare il messaggio e ottenere nuovamente il testo in chiaro m. L'avversario Eve, anche se in

ascolto sul canale di comunicazione, può solo catturare il crittotesto e dovrà cercare di capire qual è la chiave per decifrarlo, con probabilità $p = \frac{1}{2^{|k|}}$.

Un crittosistema è **sicuro** quando la probabilità di indovinare la chiave non è significativamente superiore a $\frac{1}{2^{|k|}}$; in pratica, il crittosistema è sicuro solo quando l'attaccante non ha altri modi per dedurre la chiave, se non a tentativi.

Crittografia simmetrica

Un crittosistema si dice **simmetrico** se $k' = k$, ossia se il modo di estrarre il testo in chiaro m dal crittotesto c è mediante l'utilizzo della stessa chiave k impiegata per crittografare il messaggio. Ossia:

$$\mathcal{D}(\mathcal{E}(m,k),k) = m$$

$$\forall k_1 \neq k \implies \mathcal{D}(\mathcal{E}(m,k),k_1) \neq m$$

Pertanto, Alice e Bob comunicano con la chiave k: Alice invia il crittotesto utilizzando k e Bob decifra i crittotesti sempre utilizzando k. Il problema in questo approccio, tuttavia, risiede in un punto: **comunicare k ad Alice e Bob**. Tale chiave k prende il nome di **chiave a lungo termine**, in quanto dura più di una sessione di comunicazione tra Alice e Bob. In alternativa, è possibile utilizzare una **chiave di sessione** k_{ab}. Tuttavia, questa chiave di sessione dev'essere generata sempre per ogni coppia di utenti. Infine, non si risolve il problema di comunicare k_{ab} per la corretta cifratura e decifratura dei messaggi.

Crittografia asimmetrica

Un crittosistema si dice **asimmetrico** se $k' \neq k$, ossia se il modo di estrarre il testo in chiaro da un crittotesto è decodificarlo con una chiave diversa da quella impiegata per crittografare il messaggio. Più precisamente, se k è la chiave impiegata per realizzare il crittotesto, k^{-1} è la sua **inversa** e viene utilizzata per decodificarla:

$$\mathcal{D}(\mathcal{E}(m,k),k^{-1}) = m$$

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

$$\forall k_1 \mid k_1 \neq k^{-1} \implies \mathcal{D}(\mathcal{E}(m,k),k_1) \neq m$$

Esistono vari esempi di crittosistemi asimmetrici, quali DSA e RSA. Un esempio di RSA è il seguente:

- si pubblichi $n = 33$, e si scelgano $k = 3$ e $k^{-1} = 7$;
- siano $\mathcal{E}(x,e) = x^e \mod n$ e $\mathcal{D}(x,d) = x^d \mod n$;
- preso $m = 7$, si ha che $\mathcal{E}(m,k) = 13$ e $\mathcal{D}(\mathcal{E}(m,k),k^{-1}) = 7$.

RSA si basa, pertanto, sulle proprietà di inverso modulare e esponenziale modulare.

A questo punto, se Alice vuole comunicare con Bob, Alice utilizza una chiave pubblica k_b, e Bob utilizza la sua chiave privata k_b^{-1} per decifrare il messaggio. Analogamente, se Bob vuole inviare un messaggio ad Alice, utilizzerà la chiave pubblica k_a ed Alice decifra con la sua chiave privata k_a^{-1}. Se Bob vuole verificare di star ricevendo messaggi da Alice, Alice invia un messaggio cifrato con la chiave privata: nessun altro utente potrà inviare un messaggio impersonando Alice. Per cui:

- cifrare con la chiave privata garantisce **autenticazione**;
- cifrare con la chiave pubblica garantisce **confidenzialità**.

Un problema che emerge nell'impiego della cifratura asimmetrica risiede nella **certificazione** della proprietà di una chiave pubblica. Supponiamo, per esempio, che Alice voglia inviare un messaggio segreto a Bob. Per errore, Alice non utilizza k_b, bensì k_e, chiave pubblica di Eve. A questo punto Bob utilizza la sua chiave privata k_b^{-1}, ma ottiene un testo in chiaro $m' \neq m$. Eve, tuttavia, utilizza k_e^{-1} e ottiene correttamente il testo in chiaro m. Questo attacco è sulla segretezza. Ma questo attacco può anche essere sull'autenticazione se l'abbinamento utente-chiave pubblica non è certificato: infatti, Alice può cifrare utilizzando la propria chiave privata, ma Bob potrebbe non riuscire ad autenticarla in quanto utilizza una chiave pubblica sbagliata. Il problema della cifratura asimmetrica, pertanto, può essere risolto utilizzando un **certificato**. Abbiamo bisogno di alcuni concetti aggiuntivi.

Crittosistema sicuro

Sia $\mathcal{E}(m,k)$ calcolato per ogni testo m e chiave k. Sia $\mathcal{D}(\mathcal{E}(m,k),k_1) = n$ calcolato per ogni chiave $k_1 \mid k_1 \neq k$ se il crittosistema è simmetrico, $k_1 \neq k^{-1}$ se asimmetrico. Se l'accesso a n **non aumenta significativamente** la probabilità di un attaccante di indovinare m o sue porzioni, il crittosistema si dice **sicuro**.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Hash crittografico

Una funzione di hash si dice **crittografica** se il digest ottenuto mediante l'applicazione della funzione hash sul messaggio cambia a causa di una qualsiasi modifica al messaggio originale. La funzione di hashing crittografica ha quattro principali proprietà:

- è *facile* calcolare il valore hash per un qualunque messaggio;
- è **impossibile** generare un messaggio a partire dal digest;
- è **impossibile** modificare il messaggio senza cambiare il digest;
- è **impossibile** trovare due messaggi diversi che hanno lo stesso digest.

Le funzioni hash crittografiche trovano applicazione per le firme digitali, i message authentication codes e altre forme di autenticazione. Un esempio di funzione hash è la famiglia di funzioni SHA.

Esempio d'uso: CTSS + hashing

Le funzioni hash crittografiche possono essere utilizzate per irrobustire CTSS. Non vengono memorizzate le password in chiaro, bensì i relativi digest.

Salting

Per proteggere la password da attacchi dizionario con hash chain o rainbow table, si aggiunge ad essa un messaggio random noto come **salt**. In questo modo, il digest non è calcolato mediante l'applicazione della funzione hash sulla password, ma sul risultato della concatenazione tra password e salt.

Il salting neutralizza l'efficacia delle hash chains e delle rainbow tables. Entrambi sono tabelle in cui sono memorizzate combinazioni password-digest. Aggiungere il salt rende più difficile l'utilizzo di tali tabelle, poiché l'hash non è più calcolato solamente tramite la password, bensì con il salt: si indovina una combinazione password-salt che non è valida per l'autenticazione.

Il salting pertanto aggiunge entropia all'input della funzione hash, che rende la tabella di hashing da precalcolare per invertire l'hash molto più grossa. Unix impiega questa tecnica generando il salt per ogni password da memorizzare. Usa poi la password per codificare una stringa di zeri insieme al salt, impiegando la funzione `crypt` basata su DES. Più precisamente, una volta inserita la password, si realizza una chiave di 56 bit ricavata dai 7 bit meno significativi dei primi 8 caratteri della password. Ad un salt di 12 bit viene concatenata una stringa di zeri, mentre la password viene utilizzata come chiave di encryption. Quindi si impiega una funzione di encryption che funge da hashing.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Freshness

La **freshness** è un attributo delle proprietà di segretezza e autenticazione che ne stabilisce la recente validità. Un segreto con tempo infinito è scovabile dall'attaccante, e lo scopo dei crittosistemi è rendere l'attacco impossibile entro i tempi prestabiliti.

Un esempio di freshness dell'autenticazione proviene dall'ambito web. Le piattaforme che prevedono un meccanismo di login utilizzano token di autenticazione per evitare che ad ogni richiesta HTTP sia richiesto nuovamente di autenticarsi. Il token di autenticazione ha una scadenza per confermare che l'origine della richiesta sia sempre l'utente verificato.

Per garantire la freshness si utilizzano dei **timestamp**: la expiry è un esempio di questo. Tuttavia per i timestamp c'è un problema di affidabilità legato al fatto che questo può essere modificato per far sì che il token sembri più recente. Si impiegano, inoltre, la **nonce**, ossia un numero random che viene usato solo una volta. Supponendo di non voler usare i timestamp in una rete ostile, per avere garanzie di freshness sull'autenticazione dell'interlocutore il mittente crea una nonce. L'interlocutore che riceve questa nonce si potrà autenticare citando questa.

Il marcatore temporale è inserito da chi vuole **dare** garanzia di freshness, associandolo al messaggio target in maniera affidabile. Chi invece vuole **ricevere** garanzia di freshness pubblica la nonce e aspetta che il traffico la citi. L'utilizzo di timestamp richiede orologi sincronizzati, mentre la nonce no.

Da crittografia a sicurezza

Le primitive crittografiche vengono utilizzati per la realizzazione di protocolli di sicurezza. Ci interessa garantire le proprietà di **segretezza**, **autenticazione** e **integrità**.

Useremo la seguente sintassi:

- gli agenti sono identificati con lettere maiuscole, quali $A, B, C...$;
- le chiavi crittografiche sono indicate con k_a, k_b.... Le chiavi note ad entrambe le parti sono indicati con doppio pedice, come $k_{ab}, ...$, mentre le chiavi inverse con l'esponente k_a^{-1};
- la nonce è indicata con $N_a, N_b, ...$;
- il timestamp è indicato con $T_a, T_b, ...$;
- il digest è indicato con $h(m), ...$;
- le concatenazioni sono indicate con m, m'. In questo caso, m è concatenato con m';
- i crittotesti sono indicati con m_k. Indica che m è crittografato con chiave k.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Ci poniamo adesso il problema di autenticazione di un messaggio concatenato. Ossia, A invia la concatenazione m, n a B. Il problema adesso è di vedere se B può autenticare m, n. In realtà, la concatenazione è un'operazione che può essere svolta da chiunque: l'unica cosa che si può fare è verificare l'autenticità delle singole parti nel momento in cui viene utilizzata, ad esempio, una chiave privata. Per cui $m_{k_a^{-1}}, n_{k_a^{-1}}$ può far verificare solamente m ed n, ma non la loro concatenazione.

Sintassi di un protocollo di sicurezza

Ogni protocollo di sicurezza prevede la seguente sintassi:

$$1.\ A \to B : A, N_a$$

Si indica pertanto il numero del passo, il mittente, una freccia, il ricevente, due punti e il messaggio crittografico.

In un protocollo di sicurezza, è il messaggio che darà le garanzie di sicurezza al ricevente; queste non arrivano dal mittente. Il mittente **non** è affidabile, come dire che l'indirizzo IP del mittente ha senso solo dal punto di vista funzionale.

Protocolli basilari per la segretezza

L'obiettivo è garantire la segretezza del messaggio m per A e B.

Con crittografia simmetrica è sufficiente avere una chiave segreta k_{ab} nota ad A e B. Il protocollo vede uno step:

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

L'assunto che dà senso al protocollo è la conoscenza della chiave segreta da parte di entrambe le parti.

Nel caso di crittografia asimmetrica, sono necessari due prerequisiti:

1. B deve avere una chiave privata valida, cioè sicura e non scaduta;
2. A deve verificare che k_b è chiave pubblica di B.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Il protocollo ha uno step:

$$1.\ A \to B : m_{k_b}$$

Protocolli basilari per l'autenticazione

L'obiettivo è far sì che A si possa autenticare a B.

Nel caso di crittografia asimmetrica:

1. A deve poter avere una chiave privata valida;
2. B deve poter verificare che k_a è di A.

Il protocollo ha uno step:

$$1.\ A \to B : \text{"Sono io"}_{k_a^{-1}}$$

Il contenuto è irrilevante per garantire autenticazione; è sufficiente utilizzare un qualunque testo intelligibile. L'autenticazione è garantita dall'impiego della chiave privata di A.

Nel caso di crittografia simmetrica:

1. la chiave k_{ab} dev'essere condivisa solamente tra A e B;
2. B deve poter verificare il prerequisito 1.

Il protocollo ha uno step:

$$1.\ A \to B : \text{"Sono io"}_{k_{ab}}$$

Combinare autenticazione e segretezza

A questo punto possiamo combinare i risultati ottenuti per garantire sia segretezza del messaggio m sia autenticazione di A con B.

Nel caso di crittografia simmetrica:

1. la chiave k_{ab} dev'essere condivisa tra A e B solamente;
2. B deve poter verificare il prerequisito 1.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Il passo è:

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

Nel caso di crittografia asimmetrica:

1. B deve avere una chiave privata valida;
2. A deve poter verificare che k_b è di B;
3. A deve avere una chiave privata valida;
4. B deve poter verificare che k_a è di A.

Il passo è:

$$1.\ A \to B : \{m_{k_a^{-1}}\}_{k_b}$$

oppure:

$$1.\ A \to B : \{m_{k_b}\}_{k_a^{-1}}$$

Integrità

Qualunque messaggio sulla rete potrebbe essere alterato. Per garantire integrità, l'approccio consiste nell'utilizzo di un hash come checksum:

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

Il problema, in questo caso, risiede nella possibilità da parte dell'attaccante di ricalcolare l'hash, mandando $m', h(m')$. L'hash va potenziato con una chiave.

Una possibilità risiede nell'utilizzo di un hash autenticato, ossia l'hash viene criptato con una chiave privata del mittente:

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

Ricevuto il messaggio, B:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- applica la funzione hash ad m;
- decodifica la seconda componente;
- confronta i due risultati;
- ottiene garanzia di integrità se e solo se essi combaciano.

Questo è il concetto dietro la **firma digitale**. La firma digitale, infatti:

$$\text{sign}_A(m) = m, \{h(m)\}_{k_a^{-1}}$$

Nel caso simmetrico, si fa utilizzo della chiave di sessione condivisa tra due agenti per implementare il **Message Authentication Code** (**MAC**). In particolare, MAC(m, A, B) è realizzato con una delle due alternative:

- $m, h(k_{ab}, m)$. In questo caso, la chiave simmetrica è concatenata al messaggio, e il tutto viene hashato;
- $m, \{h(m)\}_{k_{ab}}$. In questo caso, si calcola l'hash del messaggio e lo si protegge con una chiave simmetrica. Questo simula la firma digitale, ma utilizzando una chiave simmetrica. A differenza della firma digitale, tuttavia, essa **non garantisce non ripudiabilità**, in quanto la chiave è nota sia ad A che a B: in questo caso, B potrebbe generare un MAC e fingere che ciò sia stato fatto da A. In generale il MAC garantisce integrità e autenticazione reciproca, ma non la non-ripudiabilità.

Firma digitale

La firma digitale, introdotta precedentemente, è basata su crittografia asimmetrica. Ha bisogno di certificazioni delle chiavi pubbliche e ottiene sia integrità del documento sia autenticazione del firmatario, contrariamente ad una firma cartacea. Un sistema di firma è composto da due algoritmi: uno di creazione e uno di verifica. La creazione di una firma è più che criptare, e la verifica della firma è più che decriptare. Nessuno deve poter firmare per conto di un altro, mentre chiunque deve poter verificare la firma di chiunque altro.

La firma digitale garantisce:

- **integrità**, per il meccanismo visto precedentemente;
- **autenticazione**, in quanto il mittente che invia la firma utilizza la chiave privata nota solo a quest'ultimo;
- **non-ripudiabilità**, ossia il mittente **non** può negare di aver inviato il messaggio in quanto è il proprietario della chiave privata che genera la firma, e tale chiave è nota solo a quest'ultimo.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Il metodo di una creazione della firma è il seguente:

1. si prende il testo in chiaro e si produce il digest;
2. si cifra il digest con la chiave privata;
3. **la firma è la coppia messaggio in chiaro e il digest crittografato**, cioè la firma è:

$$\text{sign(m)} = m, \{h(m)\}_{k^{-1}}$$

La firma può essere verificata con i seguenti passi:

1. si decifra il digest con la chiave pubblica del mittente e si ottiene un digest;
2. si prende il testo in chiaro e si calcola il digest;
3. si confrontano il digest ottenuto mediante lo step $2.$ e il digest decifrato nello step $1.$:
    - se i digest coincidono, la firma è valida;
    - se i digest non coincidono, la firma non è valida.

La verifica della firma può essere svolta da chiunque.

Garantire segretezza, autenticazione e integrità

Per garantire segretezza, autenticazione e integrità, basta combinare i passi precedenti. Per la crittografia asimmetrica:

1. B deve avere una chiave privata valida;
2. A deve poter verificare che k_b è di B;
3. A deve avere una chiave privata valida;
4. B deve poter verificare che k_a è di A.

A questo punto:

$$1.\ A \to B : \text{sign}_A(\{m\}_{k_b})$$

Vediamo come sono garantite le tre proprietà:

- la segretezza è garantita dal fatto che il messaggio è cifrato con la chiave pubblica di B. Solamente B potrà decifrare il messaggio;
- A si autentica a B mediante la sua firma digitale;
- l'integrità del messaggio è garantita mediante la firma digitale di A.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Problema dei protocolli basilari

Poiché non è implementato alcun meccanismo di freshness, tutti i protocolli visti sono vulnerabili a **replay attack**, attacchi in cui il messaggio viene replicato. Infatti, per un attaccante è sufficiente utilizzare un messaggio che lo autentica recuperato precedentemente.

Scambio di chiavi

Abbiamo visto le seguenti problematiche:

1. la comunicazione mediante crittosistema simmetrico necessita la condivisione del segreto condiviso;
2. la comunicazione mediante crittosistema asimmetrico necessita la certificazione delle chiavi pubbliche.

Presentiamo, pertanto, due protocolli per lo scambio di chiavi simmetriche e il meccanismo di certificazione delle chiavi pubbliche.

Protocollo Diffie-Hellmann

A e B concordano due parametri pubblici α, β (il termine "pubblico" va inteso nell'accezione comune e non nel senso crittografico di crittografia asimmetrica). Successivamente, A genera X_a randomicamente, e si calcola $Y_a = \alpha^{X_a} \mod \beta$. B genera X_b randomicamente, e si calcola $Y_b = \alpha^{X_b} \mod \beta$. Si segue il protocollo di scambio:

$$1.\ A \to B : Y_a$$

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

Alla ricezione di $1.$ B calcola $Y_a^{X_b} \mod \beta$, e alla ricezione di $2.$ A calcola $Y_b^{X_a} \mod \beta$. Entrambi otterranno lo stesso numero. Infatti A ottiene:

$$(\alpha^{X_b} \mod \beta)^{X_a} \equiv \alpha^{X_b^{X_a}} \quad (\mod \beta)$$

E B ottiene:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

$$(\alpha^{X_a} \mod \beta)^{X_b} \equiv \alpha^{X_a^{X_b}} \quad (\mod \beta)$$

Pertanto i valori ottenuti coincidono e vengono utilizzati come segreti. Questo perché l'attaccante dovrebbe calcolare il logaritmo discreto per estrarre X_a e X_b, ma il logaritmo discreto è **intrattabile**, ossia computazionalmente complesso. Ad ogni modo, anche intercettando Y_a e Y_b, l'attaccante non risolve nulla.

Attacco MitM su protocollo Diffie-Hellmann

Osserviamo che tale protocollo non fornisce alcuna misura di autenticazione. È possibile sferrare un attacco **Man in the Middle** (**MitM**). Più precisamente, l'attaccante C ha una sua coppia X_c e Y_c, e quando A, B eseguono lo scambio, C si intromette:

$$1.\ A \to B : Y_a \qquad \text{bloccato da } C$$

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

$$2.\ B \to A : Y_b \qquad \text{bloccato da } C$$

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

L'attaccante si impossessa di Y_a e Y_b e somministra la propria Y_c. Alla ricezione di $1'$, B calcola $Y_c^{X_b} \mod \beta$. Alla ricezione di $2'$, A calcola $Y_c^{X_a} \mod \beta$. Pertanto, B possiede $Y_c^{X_b} = \alpha^{X_c^{X_b}}$, mentre A possiede $Y_c^{X_a} = \alpha^{X_c^{X_a}}$. Per cui, posto che:

- A conosce la chiave $k_1 = Y_c^{X_a} \mod \beta$,
- B conosce la chiave $k_2 = Y_c^{X_b} \mod \beta$,
- $k_1 \neq k_2$,

e che C conosce le reali k_1 e k_2, che calcola come:

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

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

segue che la distribuzione della chiave di sessione fallisce, in quanto i segreti non coincidono. L'attacco pertanto è sulla autenticazione per i passi $1', 2'$ e sulla segretezza per i calcoli di k_1, k_2.

Per irrobustire Diffie-Hellmann, è possibile adottare una delle due modifiche:

1. i due interlocutori utilizzano la propria chiave privata per mandare Y_a e Y_b. L'attaccante può decifrare con la chiave pubblica, ma per dare il proprio Y_c dovrebbe conoscere le chiavi private, che non sono note. In questo modo, si impedisce la prima parte dell'attacco. Questo garantisce **autenticazione**. Ciò è riassunto con gli step:

$$1.\ A \to B : \{Y_a\}_{k_a^{-1}}$$

$$2.\ B \to A : \{Y_b\}_{k_b^{-1}}$$

2. i due interlocutori utilizzano la chiave pubblica per inviare Y_a e Y_b. In questo modo, anche se l'attaccante invia Y_c, non può scoprire Y_a e Y_b. Questo garantisce **segretezza**. Ciò è riassunto con questi step:

$$1.\ A \to B : \{Y_a\}_{k_b}$$

$$2.\ B \to A : \{Y_b\}_{k_a}$$

Si ricorre, pertanto, alla crittografia asimmetrica.

RSA Key Exchange

In alternativa a Diffie-Hellmann è possibile utilizzare un altro algoritmo basato su crittografia asimmetrica. Il mittente inventa una chiave di sessione e la manda in una busta digitale:

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

Anch'esso manca di autenticazione, ma l'attaccante non può calcolare la chiave di sessione. Affinché il protocollo funzioni, A deve procurarsi il certificato della chiave pubblica di B.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Certificazione (crittografia asimmetrica)

La certificazione è fondamentale per associare una chiave pubblica al suo legittimo proprietario. L'associazione è realizzata mediante un **certificato digitale** firmato da una **autorità di certificazione** (CA), elemento fondamentale di un certificato. Il formato standard del certificato prende il nome di **X.509**.

Un certificato combina tre elementi fondamentali:

- l'individuo da certificare A;
- la sua chiave pubblica k_a;
- l'autorità di certificazione CA.

I primi due elementi sono legati mediante **concatenazione**. Serve una primitiva per firmare tale concatenazione, che è fornita chiaramente dal certificato dell'autorità di certificazione:

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

Nel formato **X.509**, l'individuo da certificare è indicato con il "Subject Name" e l'autorità di certificazione è indicata con "Issuer Name". Nel dettaglio il formato prevede:

1. Versione;
2. numero seriale del certificato;
3. l'identificatore dell'algoritmo di firma;
4. issuer name;
5. periodo di validità;
6. subject name;
7. la chiave pubblica del subject;
8. issuer unique identifier (dalla versione 2);
9. subject unique identifier (dalla versione 2);
10. extensions (dalla versione 3);
11. firma della CA.

Il certificato è verificabile mediante la chiave pubblica della CA. Gli interlocutori, pertanto, possono verificare il certificato, e quindi estrarre la chiave pubblica del subject. Ma adesso il problema viene spostato verso la CA: in linea teorica, non si capisce qual è la chiave pubblica della CA. Ossia, bisogna determinare chi certifica il possesso della chiave pubblica della CA. Quindi anche la chiave pubblica della CA dev'essere certificata; chi certifica tale chiave è un ente di grado più alto, e il problema si

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

ripete per tale ente. Questo problema dà vita ad una gerarchia ad albero nota come **Public Key Infrastructure** (**PKI**). L'elemento alla radice dell'albero prende il nome di **root certification authority** (**RCA**), o **primary certification authority**. Al livello di root, vale la **auto-certificazione**:

$$\{\text{RCA}, k_{\text{RCA}}\}_{k_{\text{RCA}}^{-1}}$$

Se un nodo dell'albero viene compromesso, la chiave privata utilizzata dal CA corrispondente a quel nodo non è più segreta; i certificati firmati da questa CA non saranno più validi. Quindi anche tutto l'albero radicato a tale nodo è inaffidabile, dove con "inaffidabile" si intende che non è più possibile porre fiducia all'abbinamento identità-chiave pubblica che si voleva apprendere. Non si perde, chiaramente, segretezza, in quanto le chiavi da certificare sono già pubbliche.

La lista completa dei certificati che permettono di verificare il certificato di A prende il nome di **catena di fiducia** di A. Per ottimizzare il processo di verifica, si memorizza per il periodo di validità il certificato per una determinata chiave pubblica.

Esempio: HTTPS

HTTPS è la versione sicura di HTTP. Le chiavi simmetriche per la comunicazione sicura vengono scambiate mediante un algoritmo di crittografia asimmetrica. Questo richiede che la pagina web abbia un certificato valido, per cui è necessaria una PKI. I certificati di root risiedono direttamente nel browser, al fine di evitare injection di root. Inoltre, è una scelta commerciale del produttore del browser decidere quali certificati di root inserire: infatti, se un browser rimuove un certificato di root di una RCA, tutti i siti web certificati da tale RCA non potranno essere visitati in maniera sicura dal browser. In generale, quando il browser non riesce a risolvere la chain of trust fino ad una radice che conosce, lo notifica all'utente, che può decidere di accettare il certificato o meno. Accettare il certificato temporaneamente non è mai la scelta sicura. Conviene o rifiutare o accettare permanentemente.
Nulla vieta ad un attaccante di poter comprare un nome di dominio, comprare il certificato e far sì che il browser riconosca l'URL. Facendo domain squatting, il nome del dominio può essere molto simile a quello di un sito già noto.

Infine, il fatto che un sito sia certificato non implica che il sito è sicuro: indica solamente che il sito web che appare corrisponde all'identità dell'URL.

Chiaramente il certificato ha un prezzo. Per tale motivo, esistono due tipi di certificati:

- **Domain Validate** (**DV**), a basso prezzo e che verifica che il compratore detiene il dominio per cui chiede il certificato. Il certificato non fornisce **nessuna** informazione aggiuntiva sull'identità del possessore del dominio;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- **Extended Validation** (**EV**), a prezzo più alto e che include più step di verifica. Nel dettaglio, il certificato EV garantisce che il dominio appartiene ad un **ente legale verificato**. In passato il certificato EV mostrava il nome dell'azienda in una barra verde. Oggi i browser non lo mostrano più, rendendo il certificato meno visibile.

È importante sottolineare come il certificato DV garantisca sicurezza del canale, **ma non sull'identità del sito**, per cui l'interlocutore potrebbe non essere sicuro. Non vi è alcuna garanzia che un sito con certificato DV sia affidabile.

Segretezza e Trust

Come visto, se l'attaccante si impossessasse della chiave privata di una certa CA, tutti perderebbero fiducia in qualunque chiave pubblica la cui certificazione necessiti della chiave pubblica di tale CA. Chiaramente, l'attaccante non si impossessa di alcun altra chiave privata automaticamente.

La perdita di segretezza, a differenza della perdita di trust, non si propaga ai livelli inferiori.

Certificate Revocation List (CRL)

I certificati possono essere revocati prima della loro scadenza. I certificati recovati in anticipo vengono inseriti in una lista nota come **certificate revocation list** (**CRL**). La lista è firmata dalla CA che ha emesso i certificati ora revocati.

Tipicamente le CRL hanno una lifetime di validità di 24 ore. Durante il periodo di validità della CRL, questa viene consultata da un'applicazione PKI per verificare il certificato della CRL.

La best practice sarebbe quella di consultare la CRL prima di accettare il certificato. Se l'accettazione di un certificato fallisce in assenza di una valida CRL, nessuna operazione dipendente sull'accettazione del certificato potrebbe accadere. Questo è tipico di sistemi come Kerberos, dove il fallimento del recupero del token di autenticazione impedisce l'accesso al sistema. Ciò rende difficile l'utilizzo di CRL in una PKI, per il fatto che l'accesso alle CRL nega il vantaggio di PKI sul protocollo di crittografia simmetrica, cioè il fatto che i certificati possono essere self-authenticating: infatti, bisogna scaricare periodicamente la CRL ed elaborarla per consultare lo stato dei certificati.

Un'alternativa all'utilizzo di CRL è il protocollo di validazione del certificato noto come **Online Certificate Status Protocol** (**OCSP**). OCSP è un protocollo impiegato per ottenere lo status di revoca di un certificato. Si bsaa sull'utilizzo di richieste OCSP ad un OCSP responder. Il responder invia una

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

risposta firmata, dichiarando se il certificato è “buono” oppure revocato. OCSP, tuttavia, è vulnerabile a replay attacks, per cui permette l’utilizzo di una nonce. Tuttavia, molti provider non utilizzano una nonce per motivi di carico di rete, e utilizzano risposte pre-firmate valide per un periodo limitato. Questo ha un vantaggio rispetto a CRL: invece di scaricare la CRL periodicamente, si fa Firefox è un esempio di browser che non utilizza CRL e adotta OCSP.

Garantire freshness

Per garantire freshness si utilizza una **nonce**. La nonce ha bisogno di un round per essere fruita, cioè è utilizzabile solamente una volta. Tale meccanismo è un tipo di **challenge**.

Supponiamo di voler garantire freshness per A, mittente. A pubblica una nonce N_a e la fa sapere a B in un cifrario asimmetrico:

$$1.\ A \to B : N_a$$

A questo punto B invia nuovamente la nonce pubblicata, cifrandola con la propria chiave privata:

$$2.\ B \to A : \{N_a\}_{k_b^{-1}}$$

In questo modo, B è autenticato con A.

La codifica della nonce non può essere più vecchia dell’istante in cui ha inventato la nonce, in quanto A conosce il momento in cui è stata creata: semplicemente, non la conosceva nessuno. Questo, però, non significa che il messaggio non possa essere replicato, ma è l’esistenza della nonce che definisce un marcatore temporale: chi si autentica con A per mezzo di riproduzione del messaggio, non potrebbe replicare nulla di più vecchio dell’istante di creazione della nonce. Il limite temporale è costruito verso il passato.

Se A è scaltra abbastanza nel creare nuove N_a ad ogni sessione, A dovrà vedere nonce inventate di recente. Questo è uno scenario in cui mal grado non esistano orologi sincronizzati, il lavoro offline di A garantisce freshness, ossia che le repliche non siano anteriori all’istanze in cui A inventò la nonce N_a. L’impiego della chiave privata, inoltre, è fondamentale per individuare B: altrimenti, N_a potrebbe essere catturata in MitM.

Un’altra versione possibile:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

$$1.\ A \to B : \{N_a\}_{k_b}$$

$$2.\ B \to A : N_a$$

In questo modo, la nonce è decifrabile solamente da B, permettendo a B di autenticarsi con A: la decifrazione, infatti individua l'unica entità in grado di farlo, ossia il detentore della chiave pubblica. Inoltre, poiché A rivede la nonce nascosta, non solo individua B, ma questi ha inviato la nonce solo dopo che A l'ha creata. Questa è una misura alternativa ma equivalente alla prima.

Mutualità

Gli schemi proposti non sonno **mutuali**. Solo uno dei due interlocutori si autentica: per ora, B si autentica con A, ma A non si sta autenticando con B. In realtà, è sufficiente ripetere gli step in modo tale che A si autentichi a B:

$$1.\ A \to B : \{N_a\}_{k_b}$$

$$2.\ B \to A : N_a$$

$$3.\ B \to A : \{N_b\}_{k_a}$$

$$4.\ A \to B : N_b$$

I passi $2, 3$ possono essere semplificati mediante concatenazione:

$$2.\ B \to A : N_a, \{N_b\}_{k_a}$$

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Protocollo di Needham-Schroder

Risolto il problema dell'autenticazione con freshness, bisogna capire come gli interlocutori autenticati possano sfruttare tale autenticazione. Bisogna, quindi, trovare un modo per garantire un modo di continuare la comunicazione, tenendo conto del fatto che sono autenticati. Ossia, **bisogna far continuare la comunicazione riconoscendosi per tutta la sessione**.

Tipicamente, una volta autenticati gli interlocutori vorrebbero costruire un canale cifrato per ottenere una sessione sicura. Per realizzare il tunnel sicuro, gli interlocutori necessitano di una chiave di sessione: tale chiave garantisce segretezza dopo l'autenticazione e ciò garantisce che qualunque cosa sia stata cifrata da quella chiave proviene dall'interlocutore autenticato.

In realtà è sufficiente modificare leggermente il protocollo descritto precedentemente:

$$1.\ A \to B : \{A, N_a\}_{k_b}$$

$$2.\ B \to A : \{N_a, N_b\}_{k_a}$$

$$3.\ A \to B : \{N_b\}_{k_b}$$

Questo funziona sempre a scopi di autenticazione con freshness; viene semplicemente aggiunta confidenzialità. A questo punto, si utilizza come chiave di sessione simmetrica:

$$k_{ab} \equiv N_a, N_b$$

Oltre alla confidenzialità, al passo 1 viene anche specificato come messaggio A, cioè il nome del mittente. In realtà, ciò non rappresenta alcuna misura di sicurezza, ma è solamente una facilitazione: B, d'altronde, non potrebbe sapere con chi sta comunicando, e quindi non potrebbe sapere quale chiave pubblica utilizzare. La garanzia che è A il vero interlocutore la dà solamente lo step 3. A, invece, può sapere che sta comunicando con B per il fatto che riceve N_a: solamente chi detiene k_b^{-1}, cioe B, poteva decifrare N_a.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Attacco di Lowe

Nel 1995 Lowe pubblica un attacco MitM contro il protocollo di Needham-Schroder. Un terzo attore C può essere coinvolto nella comunicazione. Supponiamo che A voglia comunicare con C:

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

A questo punto, C malevolo invia un messaggio a B:

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

Per cui B risponde:

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

C non può decifrare il messaggio, per cui lo inoltra ad A:

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

A manda l'ultimo step a C:

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

E a questo punto C conosce N_b, e inoltra a B il messaggio:

$$3'.\ C \to A : \{N_b\}_{k_b}$$

Per A e B, non ci sono state irregolarità nell'esecuzione dei passi del protocollo: B pensa che ci sia A, il protocollo va a termine e pertanto B autentica A. Non solo è l'autenticazione a fallire, ma anche la confidenzialità: l'attaccante, infatti, è a conoscenza di N_a e N_b, che costituiscono la chiave di sessione. **L'inganno**, tuttavia, **è su** B: C impersona A agli occhi di B, per cui l'attacco è orientato a B piuttosto che ad A. A, infatti, ha idea di comunicare con C. In generale, è B a non distinguere la sessione.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Per comprendere la potenza dell'attacco, supponiamo che B sia una banca. C potrebbe inviare un messaggio cifrato con la chiave di sessione per richiedere un trasferimento di denaro da A a C. Oppure, se A è un docente e B uno studente, C potrebbe dire a B che una lezione è annullata. La vittima, chiaramente, dipende dal contesto: nel primo caso è A ad essere la vittima, ma nel secondo è B ad esserlo.

Attacco di vendetta con modello General Attacker

Needham giustificò il protocollo affermando che è chiaro che se C fosse modellato come un outsider non avrebbe senso che A comunichi con C: il modello dell'attaccante DY non si adatta in quanto A e B sono agenti fidati, ossia è pensato per collaboratori remoti. Quindi anche C dovrebbe essere un collaboratore remoto e l'attacco non è realistico.

Lo stesso attacco può essere studiato con il modello General Attacker. In particolare, N_a dovrebbe essere noto solo tra A e C. Ipotizziamo, pertanto, che B **possa essere malevolo**. A non sa che N_a è finita anche nelle grinfie di B: B può sfruttare questo vantaggio contro A. A pensa di condividere questo segreto con C. Se B ha N_a, può mandare messaggi ad A impersonando C.

Questo apre le porte ad uno **scenario di vendetta**: se A fosse una banca, B potrebbe vendicarsi su C e dire di trasferire soldi sul proprio conto impersonando C.

Contromisure

La contromisura più efficace è la modifica allo step 2. Invece di inviare solamente la coppia N_a, N_b crittografata con chiave pubblica, l'interlocutore invia anche la propria identità:

$$2.\ B \to A : \{N_a, N_b, B\}_{k_a}$$

In questo modo, C non può modificare il messaggio, in quanto dovrebbe decifrarlo, ma questo è impossibile in quanto non detiene k_a^{-1}. Sebbene sia vero che il messaggio possa essere creato da chiunque, C non potrebbe mai creare un messaggio dicendo che è B: A, infatti, sta comunicando con C. Se invece specificasse di essere C, non ci sarebbe l'attacco in quanto sta semplicemente instaurando una comunicazione con A.

Altre proposte potrebbero essere:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- modificare il primo step, in modo tale che sia $1.\ A \rightarrow B : \{\{A, N_a\}_{k_a^{-1}}\}_{k_b}$. Questa soluzione non è adeguata, in quanto A invia a C il messaggio cifrato con k_c, che decifra e lo cifra nuovamente con k_b, per cui si ripete la sequenza descritta;
- modificare il primo e il secondo step, così da avere $1.\ A \rightarrow B : \{\{A, N_a\}_{k_a^{-1}}\}_{k_b}$ e $2.\ B \rightarrow A : \{\{N_a, N_b\}_{k_b^{-1}}\}_{k_a}$. In questo caso, l'attaccante non può decifrare il messaggio, e può semplicemente inoltrarlo ad A. Ma A nota che le nonce sono cifrate con k_b^{-1}, e poiché non coincide con la chiave privata dell'interlocutore annulla la comunicazione. La soluzione è funzionante, ma è un grado ulteriore di crittazione non necessario, e B estrae N_a in una sessione differente;
- modificare il secondo step, in modo tale che sia $2.\ B \rightarrow A : \{\{N_a, N_b\}_{k_b^{-1}}\}_{k_a}$. È una modifica della soluzione precedente, in quanto viene rimosso il grado di encryption al primo step. Tuttavia, è sempre presente un ulteriore grado al secondo step non necessario;
- modificare il primo step in modo tale che sia $1.\ A \rightarrow B : \{A, B, N_a\}_{k_b}$. In realtà la soluzione non è adeguata: se A comunica con C, C semplicemente decifra con k_c^{-1} e riscrive il messaggio, portando al successo dell'attacco.

Protocollo di Woo-Lam

Il protocollo di Woo-Lam impiega **cifratura simmetrica**. Il protocollo è impiegato per l'autenticazione **unidirezionale**: A vuole autenticarsi con B.

$$1.\ A \rightarrow B : A$$

Il primo step è **presentarsi**, e funge da "indizio" da fornire all'entità presso cui autenticarsi. Chiaramente sappiamo che qualsiasi attaccante può dire che si chiama A.

$$2.\ B \rightarrow A : N_b$$

Nello step 2 B genera una nonce per lo schema **challenge-response**. Dopo che A riceve il challenge, A utilizza la propria chiave segreta per cifrare la nonce:

$$3.\ A \rightarrow B : \{N_b\}_{k_a}$$

Tipicamente, in protocolli a schemi simmetrici si assume che troveremo un **TTP** (**Trusted Third Party**) che possiede un database di tutte le chiavi; il TTP, pertanto, fa le veci di un authentication server. A questo punto, B non può utilizzare la chiave k_a in quanto è nota solo ad A. Poiché B non può

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

verificare che A ha ricevuto la nonce, in quanto il messaggio è non decifrabile, B può contattare il TTP per ottenere conferme:

$$4.\, B \rightarrow TTP : \{A, \{N_b\}_{k_a}\}_{k_b}$$

B non solo inoltra il messaggio ricevuto allo step 3, ma anche l'identità del mittente originale, il quale costituisce una **guess**: B infatti deduce che il mittente sia A, ma non lo sa fino a quando non riceve conferme dal TTP. La coppia mittente-messaggio viene sigillata con la cifratura di B: senza essa, l'attaccante può cambiare A e ingannare il TTP. Questo perché l'identità A esplicitata viene impiegata per comunicare al TTP di cercare la chiave a lungo termine di A per decifrare il messaggio che B inoltra. Se TTP usasse la chiave di qualcun altro per decifrare la seconda componente, non si otterrebbe N_b ma N_b'. Quindi B troverebbe una nonce diversa da quella originariamente inviata e l'attacco è di Denial of Service. Si potrebbe desumere che il livello di encryption esterno sia discutibile: l'attaccante potrebbe infatti invertire un bit per far sì che B rifiuti l'apertura di sessione.

A questo punto, il TTP comunica a B quanto trovato:

$$5.\, TTP \rightarrow B : \{N_b\}_{k_b}$$

Il TTP impiega l'encryption per due motivi:

1. il TTP garantisce la propria identità con l'invio del messaggio cifrato con la chiave nota solo a lui e a B;
2. il messaggio sia diverso dallo step 2, evitando relay.

In questo modo, la fiducia di B nei confronti del TTP è massima.

Dal messaggio quattro acquisisce senso l'esplicitazione, mentre prima si parla di challenge cifrato fino al passo 3.

Attacco

Se A è offline, un attaccante può indurre B a pensare che sta con A:

$$1.\, C \rightarrow B : A$$

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

$$1.' \; C \rightarrow B : C$$

Su una sessione, C, attaccante, dice di essere A, mentre sull'altra segue il protocollo identificandosi.

$$2. \; B \rightarrow A : N_b$$

$$2.' \; B \rightarrow C : N_b'$$

B genera due nonce diverse N_b, N_b'. Se B manda entrambe le sfide, significa che a fine protocollo se B vedesse N_b associata ad A, autenticherebbe A; se B vedesse N_b', autenticherebbe C. A questo punto:

$$3. \; C \rightarrow B : \{N_b\}_{k_c}$$

$$3'. \; C \rightarrow B : \{N_b\}_{k_c}$$

$$4. \; B \rightarrow TTP : \{A, \{N_b\}_{k_c}\}_{k_b}$$

$$4.' \; B \rightarrow TTP : \{C, \{N_b\}_{k_c}\}_{k_b}$$

Ma così, B non sa distinguere le sessioni: non può, infatti, conoscere il testo in chiaro in quanto K_c è privata e nota solo a C e al TTP. Per cui:

$$5. \; TTP \rightarrow B : \{N_b''\}_{k_b}$$

$$5.' \; TTP \rightarrow B : \{N_b\}_{k_b}$$

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Con $5'$ concorda l'identità con la nonce cifrata. Pertanto, B non riesce più a distinguere le sessioni, e deduce che il suo interlocutore sia A in quanto riceve N_b. Il TTP, chiaramente, funge da oracolo, e non può comunicare a B altro se non la nonce cifrata.

Contromisure

Possibili contromisure sono le seguenti:

$$3.\ A \to B : \{A, N_b\}_{k_a}$$

$$5.\ TTP \to B : \{A, N_b\}_{k_b}$$

$$4.\ B \to TTP : \{A, \{A, N_b\}_{k_a}\}_{k_b}$$

$$2.\ B \to A : N_b, B$$

Il problema del protocollo risiede che, sebbene è vero che fino al messaggio 3 il meccanismo challenge-response non necessita esplicitazione, dal messaggio 4 l'esplicitazione è necessaria per confermare l'identità. Il messaggio 4 è sufficientemente esplitico, ma sul 5 non c'è esplicitazione. L'attacco si basa sul fatto che B, ricevendo N_b, assume che l'interlocutore sia A. Per cui, la soluzione corretta è la modifica sul messaggio 5, dove il TTP esplicita che A è colui che sta comunicando.

Le altre modifiche non sono sufficienti:

- la modifica al punto 4 non risolve nulla, in quanto 4 è già esplicito abbastanza;
- la modifica al punto 2 non cambia nulla, solamente la formazione del messaggio.
- la modifica al punto 3 non risolve nulla: C può inviare A, N_b su entrambe le sessioni, ingannando B.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Principi di design: esplicitazione

Quanto visto sottolinea l'importanza di un principio di design di protocolli di comunicazione: l'**esplicitazione**. In generale, **se le identità del mittente e del ricevente sono significative per il messaggio, è prudente menzionarle esplicitamente**.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

6 - Protocollo di sicurezza

I protocolli di sicurezza garantiscono specifiche proprietà di sicurezza mediante l'utilizzo di tecniche specifiche. Si assume che vi sia un protocollo di trasporto già esistente.

Abbiamo visto l'impiego della crittografia. Ma in realtà esistono altre alternative.

Steganografia

La steganografia è un'alternativa alla crittografia che consiste nel nascondere le informazioni. La segretezza è ottenuta mediante l'occultamento delle informazioni, quindi per **non distinguibilità**. Il segreto, in questo caso, è il pattern utilizzato per nascondere le informazioni.

Chaffing e winnowing

Chaffing e winnowing significano mischiare e separare. Rappresenta una forma di steganografia. Un esempio di impiego è il seguente:

1. il mittente S e il ricevente R concordano una chiave k mediante Diffie-Hellmann;
2. S spedisce a R la coppia $m, MAC(m, k)$;
3. S spedisce a R molte coppie false x, y;
4. R seleziona la coppia giusta verificandone il MAC. Solo sulla coppia giusta il controllo di integrità non fallisce. L'attaccante non saprà distinguere le varie coppie false, in quanto non ha la chiave k, sebbene queste siano inviate in chiaro.

Sicurezza a livello di rete: IPSec

Per comunicare mediante un protocollo di sicurezza, tutte le macchine devono eseguire un client per quel protocollo. Chiaramente, i protocolli di sicurezza a livello di rete non sarebbero necessari se possiamo garantire la sicurezza ad un livello più basso. Progettare la sicurezza in un qualsiasi livello ha vantaggi e svantaggi:

- implementare la sicurezza a livello di rete permette di poter ridurre le necessità di sicurezza a livello di applicazione. Tuttavia, può appesantire notevolmente la comunicazione e richiederebbe modifiche al sistema operativo;
- implementare la sicurezza a livello di applicazione permette di poter ottenere soluzioni maggiormente personalizzabili, ma può essere ingannevole per la gestione degli stati.

Introduciamo una suite di protocolli per mettere in sicurezza il protocollo IP. Tale suite è nota come **IPSec**, e contiene protocolli per garantire segretezza, autenticazione e integrità a livello IP. La suite è opzionale per IPv4 e mandatoria per IPv6. Ogni pacchetto IP è arricchito con componenti crittografiche per la sicurezza. Non tutti i nodi della rete devono essere compatibili con IPSec.

IPSec supporta due modalità di funzionamento:

- una **modalità di trasporto** per la connessione end-to-end. Il payload è l'unico ad essere cifrato, è computazionalmente leggero e ogni host che vuole comunicare deve avere il software necessario per implementare IPSec. Viene aggiunto solamente l'header IPSec;
- una **modalità tunnel** per la connessione gateway-to-gateway. L'intero pacchetto IP originale viene cifrato, ma solo i gateway devono avere il software IPSec. Utilizza un doppio incapsulamento, ponendo come payload della comunicazione tra indirizzi gateway quanto si ottiene cifrando l'unione di indirizzi mittente e destinatario degli end-point con il payload effettivo. Adottando ESP, gli indirizzi degli end-point non sono rilevabili.

La suite prevede un protocollo per lo scambio delle chiavi, il protocollo **IKE** (**Internet Key Exchange**). IKE consta di due protocolli:

- **Oakley**, basato su Diffie-Hellmann e che fornisce la chiave iniziale. È implementato a livello di applicazione;
- **ISAKMP** (**Internet Security Association and Key Management Protocol**), utilizzato per la creazione di chiavi di sessione mediante la chiave iniziale. È implementato a livello di trasporto.

Security Association

Una **Security Association** (**SA**) è una relazione unidirezionale tra mittente e destinatario. È alla base del funzionamento di IPSec e vengono stabilite dal protocollo IKE, sebbene sia possibile anche impostarle manualmente. Più precisamente, dopo lo scambio della chiave privata, vengono scambiate le informazioni della Security Association tra gli interlocutori.

Le SA individuano una comunicazione unidirezionale, per cui due SA garantiscono una comunicazione in un canale full-duplex. La SA sancisce il protocollo di sicurezza richiesto per quel traffico, nonché le tecniche di cifratura utilizzate. In particolare, sono importanti i seguenti campi:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

1. **SPI** (**Security Parameter Index**) per indicizzare. È un intero a 32 bit;
2. **Destination IP**, solo unicast;
3. **identificatore del protocollo IPSec da impiegare**, uno tra AH e ESP.

Ogni nodo della rete deve mantenere un Database per le SA, noto come **SAD** (**Security Association Database**). Il SAD mantiene tutte le SA attive in quel nodo. Ogni entry del SAD contiene tutti gli elementi di una SA e in più:

1. **AH info**, cioè le informazioni aggiuntive sull'algoritmo di autenticazione;
2. **ESP info**, cioè le informazioni aggiuntive sull'algoritmo di codifica;
3. **lifetime**, cioè la durata della SA.

IKE viene impiegato per stabilire le SA. Nella prima versione (IKEv1) il protocollo prevede due fasi:

1. la fase 1 prevede la creazione di un canale sicuro tra due peer IKE noto come **ISAKMP Tunnel**. In questa fase, i peer utilizzano Diffie-Hellmann (Oakley) per stabilire la chiave di sessione e stabiliscono gli algoritmi di encryption da utilizzare. Le specifiche sugli algoritmi prendono il nome di **IKE SA**;
2. la fase 2 imposta le SA per IPSec, definendo i parametri delle SA per entrambi i peer.

Dalle informazioni della SA è possibile stabilire le **modalità** con cui il traffico viene **protetto**. Per identificare il traffico da proteggere, si utilizza una **Security Policy** (**SP**). La SP è una regola che stabilisce il **tipo** di traffico da **instradare**. Ogni SP è memorizzata in un database, noto come SPD (**Seucrity Policy Database**). Una SP contiene:

1. IP sorgente e destinazione;
2. protocollo e relativa porta da instradare nel tunnel;
3. l'identificativo della SA da utilizzare per processare i dati.

Stabiliti SA e SP, è possibile cominciare la comunicazione, che sfrutta i protocolli AH e ESP.

Authentication Header

L'**Authentication Header** (**AH**) è un frammento aggiunto a ciascun pacchetto IP per il supporto dell'autenticazione e dell'integrità dei pacchetti. Mittente e ricevente concordano in primis una chiave IKE e lo utilizzano con l'intero pacchetto, escludendo però i campi variabili e non prevedibili, per il calcolo del MAC. Se un nodo che fa routing altera l'IP sorgente, la verifica del MAC fallisce. AH, tuttavia, non garantisce in alcun modo la confidenzialità del messaggio.

AH ha i seguenti campi:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- **next header**, di 8 bit;
- **payload length**, di 8 bit e in termini di 32 bit;
- 16 bit **riservati**;
- **SPI**, parte del SA per selezionare il SA in un SAD;
- **numero di sequenza dell'AH** all'interno del SA;
- **MAC del pacchetto**.

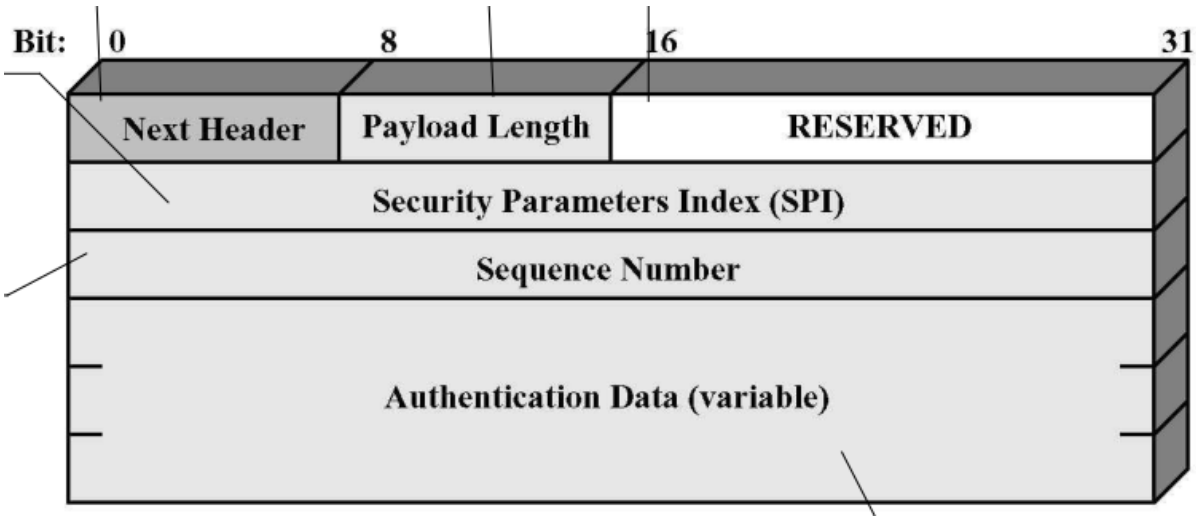


Figure 1: Schema AH

Un mittente sceglie la SA dal SAD. Ogni SA è identificata dal SPI, e viaggia con l'AH del pacchetto. Visto tale AH, ogni nodo attraversato ne preleva l'SPI per selezionare dal proprio SAD la SA relativa al pacchetto. I pacchetti, anche se autenticati, non possono essere replicati grazie al numero di sequenza dell'AH. Il ricevente accetta una finestra di numeri di sequenza, ma il meccanismo a finestra non può essere aggirato: anche se un pacchetto viene replicato e il suo numero di sequenza è modificato per fittare una finestra, il controllo di MAC fallisce. Qualunque alterazione, pertanto, viene riscontrata. Ai consueti controlli del meccanismo a finestra viene aggiunto il MAC.

AH supporta entrambe le modalità trasporto e tunnel:

- nel trasporto mittente e ricevente devono usare IPSec. L'autenticazione avviene punto-punto (end-to-end);
- nel tunnel mittente e ricevente possono non usare IPSec, ma l'autenticazione avviene mediante **intermediari**, attraverso router o firewall. Un esempio di tunnel è la comunicazione tra due nodi A e B che non usano IPSec, ma che comunicano mediante router di confine che usano IPSec. In modalità tunnel, inoltre, viene aggiunto un nuovo header IP. Quando A comunica a

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

B, chiaramente A inserisce l'ip di B. Il router di confine di A aggiunge un nuovo header ip, contenente l'ip del router di confine della rete di B.

Encapsulating Security Payload (ESP)

ESP è un frammento aggiunto ai pacchetti IP per fornire autenticazione (opzionale), confidenzialità e controllo di integrità alla comunicazione (se si inserisce MAC). L'header IP, a differenza di AH, non viene coperto dai controlli. Prevede i campi:

- **SPI** di 32 bit;
- **numero di sequenza** a 32 bit;
- **dati del payload**, crittografati;
- **padding** da 0 a 255 byte. È usato quando la cifratura richiede una lunghezza minima superiore ai dati del payload;
- **lunghezza del padding**, a 8 bit;
- **next header**, a 8 bit;
- dati di autenticazione, non sempre presenti.

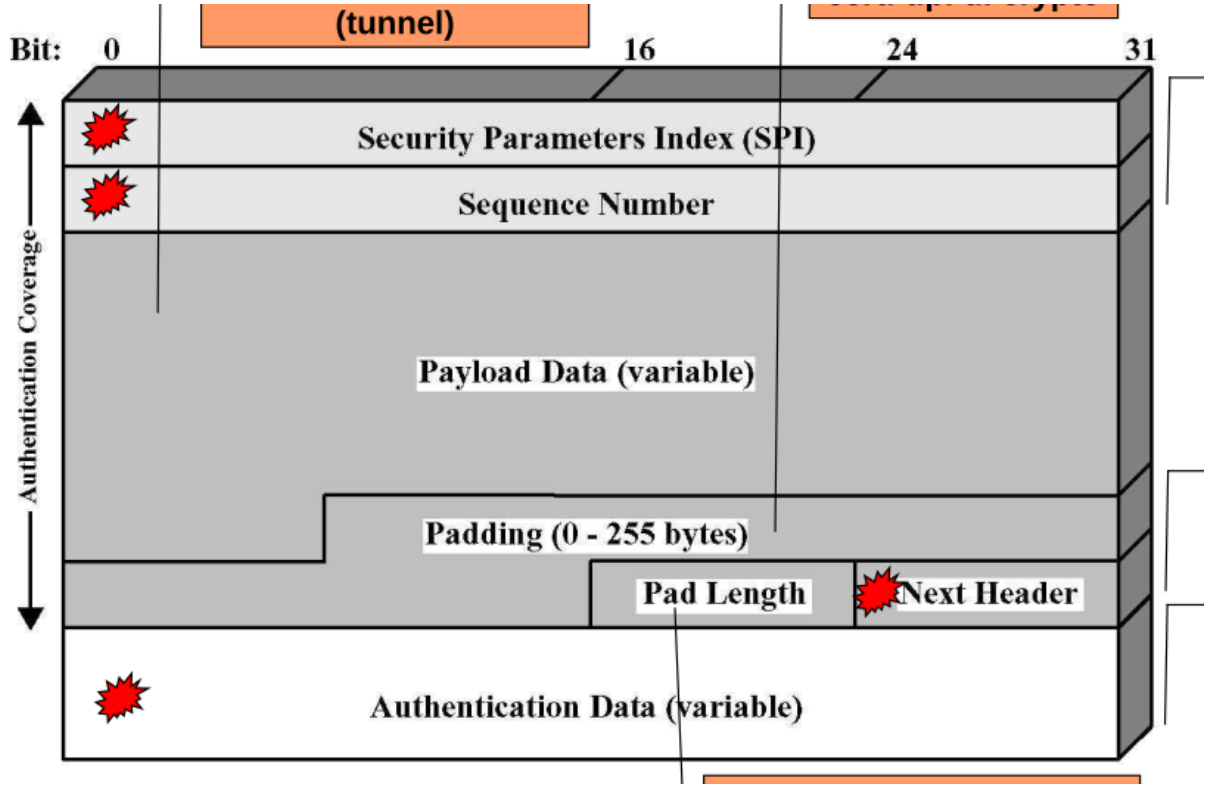


Figure 2: Schema ESP

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

ESP **avvolge totalmente** i dati sottostanti, per cui fino all'header TCP i dati sono crittografati. Tuttavia, non è coperto l'header IP. Inoltre, poiché ESP include possibilmente anche i dati di autenticazione, IPSec potrebbe essere semplificato mediante la rimozione di AH.

Combinazioni SA

Non ha senso combinare più di due SA nella modalità di trasporto. Può servire, tuttavia, combinare più SA per la modalità tunnel, definendo diversi tunnel. Ogni nodo compatibile IPSec deve, pertanto, supportare 4 tipi di combinazioni SA:

1. **sicurezza punto-punto**. Si utilizza AH e/o ESP in trasporto sugli host comunicanti;
2. **sicurezza fra intermediari**. Si utilizza AH e/o ESP in tunnel, quindi sui nodi intermedi della comunicazione;
3. combinare $1.$ e $2.$ per avere un **trasporto sicuro** end-to-end e un **tunnel sicuro** tra i nodi intermedi;
4. usando AH in tunnel sul punto-intermedio e/o ESP in tunnel sul punto intermedio. Ossia, si utilizza il tipo $1.$ per la comunicazione end-to-end e il tunnnel per la comunicazione tra un host e un nodo intermedio.

IPSec può essere ulteriormente semplificando preferendo solamente la modalità tunnel e garantendo che il protocollo sia utilizzato su ogni nodo.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

7 - Intrusion Detection

Un virus tipicamente non è un processo in esecuzione. Se un processo è in esecuzione, si parla di **intrusione**. Un'intrusione è più pericolosa di un software che compromette un eseguibile (virus), propagandosi ad altri file per corromperli; il virus appartiene alla classe dei malware. Per questo sin dagli anni '80 si usa la parola virus. Un'intrusione è un processo alieno.

Un intruso, pertanto, è un vero e proprio processo, e raramente virus sono tipi particolari di intrusi. Un intruso è solitamente impiegato per installare virus. L'intrusione, inoltre, è distinta dal Denial of Service:

- il DoS ha effetti temporanei, facilmente rilevabili e blocca le risorse. Non prevede autenticazione;
- l'intrusione ha un effetto generalmente permanente, è spesso nascosto e usa illecitamente le risorse, piuttosto che bloccarle.

DoS

La **negazione di servizio** (**Denial of Service**, **DoS**) è un attacco che impedisce la normale operatività del sistema, degrandadola o bloccandola del tutto. Può essere di tre tipi:

1. **cDoS**, consumo di risorse computazionale;
2. **mDoS**, consumo della memoria;
3. **bDoS**, consumo della banda di trasmissione.

Quando l'attacco è distribuito, si parla di **DDoS** (**Distributed Denial of Service**). In questo caso, le richieste vengono inviate da diversi **zombie** che attaccano la vittima. La macchina attaccata, o l'insieme di macchine attaccate, prende il nome di **C2 server** (**Command and Control server**). In locale i DoS erano facili quando il router avevano il servizio di broadcast.

Tradizionalmente la tecnologia usata per far fronte ai DoS è il **load balancing**. Il bilanciatore di carico è la prima macchina che distribuisce le richieste tra diverse macchine. Un'euristica di prevenzione utilizzata dal load balancer è la complicazione computazionale dell'accesso (**cookie transformation**):

$$1.\ C \rightarrow S : m_1$$

$$2.\ S \rightarrow C : m_{2s}$$

$$3.\ C \rightarrow S : m_1, m_{2s}, h(m_1, m_{2s})$$

$$4.\ S \rightarrow C : m_{2c}$$

Normalmente l'interazione tra client e server è uno scambio continuo di messaggi. In questo caso, il server S invia un messaggio più semplice sotto forma di cookie, m_{2s}, e richiede al client di inviare lo storico dei messaggi e il digest di questi. Se il client è usato per effettuare DoS, inoltra le stesse richieste in loop al server, non elaborando la richiesta di quest'ultimo. In questo modo, il server è in grado di discrimare client autentici da client zombie.

Intrusione

Un'intrusione è l'ottenimento illecito dei privilegi superiori a quelli posseduti lecitamente. La manifestazione tipica è che su una macchina ci sia un processo alieno. L'esistenza di un processo alieno è un caso particolare della definizione data.

L'intrusione prevede solitamente l'ottenimento dell'accesso al sistema oppure l'ampliamento dei propri privilegi d'accesso. Spesso l'intrusione si riduce a violare i meccanismi di autenticazione e/o controllo d'accesso, e non necessariamente avviene mediante rete (ad esempio, inserendo pen-drive con malware).

Esistono diverse tecniche per l'intrusione:

- **violazioni di password**;
- **intercettazione di informazioni sensibili**;
- **uso di combinazioni di SW nocivo**.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Individuare le intrusioni: Intrusion Detection System

L'IDS (Intrusion Detection System) è un sistema per la rilevazione di intrusioni. Il sistema registra il comportamento del sistema e analizza tale log usando tecniche di pattern matching. L'IDS può anche essere distribuito e lavorare con più processi o più macchine.

Il problema fondamentale per la realizzazione di un IDS è **definire il comportamento di un intruso rispetto a quello di un utente legittimo**. Vengono solitamente impiegate tecniche di machine learning combinate con l'analisi del comportamento medio nel tempo. Bisogna sempre trovare il miglior compromesso tra falsi positivi (utente legittimo scambiato per intruso) e falsi negativi (intruso scambiato per utente legittimo).

Il monitoraggio del comportamento del sistema prende il nome di **auditing**. Esistono diversi record di auditing:

- **record nativi** del sistema operativo. Tutti i sistemi operativi prevedono la raccolta di informazioni generali sulle attività degli utenti. Non sempre tali record danno informazioni utili;
- **record specifici per il rilevamento**. Viene utilizzato software aggiuntivo per raccogliere informazioni specifiche per il rilevamento, fornendo informazioni mirate ma appesantendo il sistema. Le operazioni classiche vengono tradotte in sequenze di azioni elementari, in modo tale che sia più semplice controllare i tentativi di aggiramento dei controlli di accesso. Un esempio di record specifico è **record di Denning**. Ogni record contiene sei campi:
  - **soggetto**, che è l'utente o processo che esegue azioni per mezzo di appositi comandi;
  - **azione**, operazione svolta dal soggetto;
  - **oggetto** su cui viene svolta l'azione;
  - **eccezione** se prodotta in risposta all'azione;
  - **uso risorse** a livello di righe stampate, settori letti, tempo CPU e unità di IO utilizzate;
  - **timestamp**.

Esistono diverse tecniche di rilevamento:

- **indipendenti dal passato**, ad esempio tre fallimenti di inserimento di password;
- **dipendenti dal passato**, osservando il funzionamento del sistema in passato e assumendo che il futuro sarà come il passato.

In particolare, riconosciamo:

- **tecniche di rilevamento statistico**. Tali tecniche si basano sulla creazione di un modello statistico del comportamento dell'utente legittimo. L'intruso si discosta sufficientemente dal modello. È particolarmente efficace per attaccanti esterni. Esistono diversi modelli:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

  - **media e devazione standard**. Si calcolano media e deviazione standard su un parametro, dando l'idea del comportamento medio e della sua variabilità. Alcuni esempi sono media e deviazione standard sulla frequenza di login durante un giono, la quantità di output in una località, l'utilizzo di risorse di sessione e la frequenza di utilizzo;
  - **modello operativo**. Si definisce un limite di accettabilità per un parametro e segnala una intrusione quando viene superato. Un esempio è il numero di limiti di tentativi per l'accesso mediante password o il numero di fallimenti per operazioni di lettura e scrittura (che individua un intruso che cerca di accedere a file senza autorizzazione);
  - **multivariazione**;
  - **processo di Markov**;
  - **serie temporali**;

- **tecniche di rilevamento a regole**. Si creano regole per il comportamento di un intruso. È efficace per attaccanti interni. Ad esempio, una regola potrebbe essere "nessun utente dovrebbe leggere i file contenuti nelle directory personali di altri utenti", e il tentativo di violare la regola costituisce un comportamento di un intruso.

Generali tecniche di protezione utilizzate sono le seguenti:

- **limitazione delle automazioni** per impedire il ripetersi di tentativi;
- **usare esche**, cioè risorse fittizie che attirano gli attaccanti;
- **usare trappole** che attivano monitoraggio appena l'attaccante accede all'esca.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

8 - Malware

Bug e software nocivo

Il **bug** è una proprietà inattesa del software che può essere sfruttato da altro software non necessariamente nocivo.

Un **software nocivo** (**malware**) è software scritto con l'esplicito scopo di violare la sicurezza di un sistema. Non sfrutta necessariamente dei bug.

Esistono diversi tipi di malware:

- software nocivo che necessita di un programma ospitante, quali backdoor, logic bomb, trojan horse e virus;
- software nocivo indipendente quali zombie e worm.

Backdoor

Una **backdoor** è un punto di accesso segreto ad un programma senza le procedure di sicurezza altrimenti previste.

Normalmente le backdoor sono installate per permettere agevoli manutenzioni da remoto, ad esempio mediante password predefinite. Possono essere installate anche da alcuni malware come trojan.

Logic bomb

Una logic bomb è un frammento di codice di un programma non nocivo pronto a "esplodere" quando si verificano certe condizioni. Per essere considerata tale, la bomba logica dev'essere ignota all'utilizzatore del programma.

Un particolare tipo di bomba logica è la **bomba a tempo**, che si attiva dopo un periodo di tempo predefinito.

Trojan horse

Un trojan horse è un programma che appare utile che in fase di esecuzione compie violazioni di sicurezza. Solitamente quando viene eseguito viene scaricato un payload, che solitamente è una backdoor o un qualsiasi software che trasforma il bersaglio in un server Command and Control, attendendo istruzioni da remoto.

Zombie

Uno zombie è un programma nocivo che sfrutta una macchina remota già violata o la macchina stessa. È molto diffuso oggi su internet ed esempi tipici sono legati agli attacchi DoS.

Worm

Un worm è un programma nocivo che infetta le macchine remote, ciascuna delle quali a loro volta infetta altre macchine. È più dannoso di uno zombie perché si trasmette da macchina in macchina senza intervento umano.

Un worm noto è il **Worm Morris** che attacca rete di macchine Unix. Secondo la storia ufficiale, Morris aveva un bug per cui alcune versioni non si terminavano ma si replicavano anche sulla stessa macchina. Le macchine poi soffrivano un serio calo prestazionale.

Worm Morris sfruttava alternativamente uno tra tre noti bug di Unix per violare la macchina remota:

- un attacco known-ciphertext sul file di password;
- BOF mediante programma finger;
- backdoor nel programma sendmail.

Dopo aver violato la macchina remota, scaricava 99 linee di codice C che si auto compilavano ed eseguivano per scaricare l'intero worm.

Nel primo caso, si sfruttava il fatto che il file di password ha tre colonne criptate il cui contenuto era leggibile da tutti. Morris, pertanto, provava a codificare possibili candidati: se aveva successo, faceva login sulla macchina remota e scaricava le righe.

In alternativa, il worm lancia `finger` e `fingerd` risponde dalla macchina remota. Un buffer overflow su buffer di input di fingerd permette a fingerd di eseguire lo shellcode caricato nel buffer che scarica 99 linee di C.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Infine, il worm sfruttava sendmail, un servizio background per attendere mail. Lo si può far passare in debugging mode per eseguire comandi: Morris sfrutta la trapdoor per scaricarvi le 99 linee sul sendmail remoto.

Virus

Il virus è un programma nocivo che viola altri programmi non nocivi, sfruttandoli per propagarsi. La violazione è raggiunta mediante l'aggiunta di pezzi di codice indesiderati ai programmi e possono essere difficili da individuare. Oggi non sono molto diffusi in quanto il mercato si è spostato su ransomware e sulla scrittura di exploit per vulnerabilità del black market o di vulnerabilità note.

Alcuni virus particolari sono i **macrovirus**, cioè virus scritti come macro di un'altra applicazione, come le macro della suite di Microsoft Office. Le macro possono essere lanciate all'apertura del documento, infettando senza dare sintomi. Oggi ci sono livelli di sicurezza per impedire l'esecuzione automatica delle macro.

Struttura del virus

La struttura del virus è abbastanza semplice. Solitamente un virus prende un file e verifica se c'è un identificativo del virus all'inizio del file: in tal caso, il file è stato già infettato. Verificato se il file è infettato, si verifica se il trigger viene attivato: in tal caso si attiva il codice malevolo per danneggiare il file. La struttura è sempre:

- **identificazione della presenza dell'infezione**;
- **infezione**;
- **danneggiamento**;
- **restituzione del controllo** al programma originale.

Nel dettaglio, ogni virus ha sempre una **routine di ricerca**, per cercare i file adatti ad essere infettati dal virus, ed una **routine di infezione**, che ha il compito di copiare il codice nel programma scelto, così che il codice del virus sia eseguito al lancio del programma infetto. Molti virus sono anche progettati per eseguire codice estraneo alle finalità di replicazione, per cui contengono anche una **routine di attivazione**, che contiene i criteri in base al quale il virus effettua l'attacco, e un **payload**, la sequenza di istruzioni dannosa che esegue violazioni di sistema.

La firma del virus viene solitamente posizionata all'inizio o alla fine. Talvolta si pone sia all'inizio sia alla fine, o si inframezza nel file. Il file infetto si può rilevare perché di lunghezza diversa; alcuni virus comprimono il file infetto alla lunghezza originale. Una strategia utilizzata è la seguente:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- comprimere il nuovo file da infettare;
- appendere il virus all'inizio;
- decomprimere il resto del file ospite;
- eseguire il file non compresso.

Esempio: Brain

Brain era un virus che attaccava sistemi Microsoft rinominando i volumi del disco in BRAIN. Più precisamente, il boot sector di un floppy disk veniva rimpiazzato con una copia del virus: il vero boot sector viene spostato in un altro settore e marcato come inutilizzabile.

Caricato in memoria alta, Brian esegue una chiamata di sistema per resettare il limite di memoria alta al di sotto dell'area che lo contiene. La posizione 19 del vettore degli interrupt contiene l'indirizzo della procedura per gestire le letture da disco. Il virus imposta l'indirizzo della procedura al proprio indirizzo, gestendo tutte le letture dei settori in memoria.

Gestendo le letture, controlla se il quinto e il sesto dei byte letti contengono la sua firma, il valore esadecimale 1234:

- se la firma non è presente, il settore letto era normale e Brain lasciava semplicemente passare la lettura;
- se la firma è presente, la lettura veniva intercettata, in quanto l'utente prova a leggere il boot sector infetto da Brain. Per non farsi scoprire, Brain reindirizzava la lettura al settore originale marcato come inutilizzabile, inducendo il sistema operativo a pensare che il disco fosse pulito.

L'infezione di un nuovo floppy disk avveniva quando si tentava l'accesso ad esso: Brain copiava sé stesso nel boot sector del nuovo disco, spostando quello originale.

Di per sé il virus non andava a distruggere i dati, ma solo a causare rallentamenti dovuti al reindirizzamento delle letture. Infatti, il virus è nato principalmente per proteggere il software medico per il monitoraggio cardiaco realizzato dai fratelli pakistani creatori del virus. Il codice del virus Brain, infatti, era inserito nel boot sector del floppy disk originale del software medico. Quando un utente cercava di duplicare quel floppy disk per piratarlo, anche il virus si copiava automaticamente sul floppy: se il disco veniva inserito in un altro computer, il virus si caricava nella memoria e infettava altri floppy disk puliti nel calcolatore.

Una versione più aggressiva può prevedere il danneggiamento casuale di 6 settori, con un meccanismo di propagazione che viene iterato fino a quando ci sono settori non ancora infetti sul disco. Lo spazio disponibile sul disco diminuirà progressivamente.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Rimuovere software nocivo: Antivirus

L'**antivirus** è un software che esegue tre compiti:

1. **individuazione**, cioè trovare il file col virus;
2. **identificazione**, cioè stabilire di che virus si tratta;
3. **eliminazione**, cioè rimuoverlo dal sistema. Può essere problematico in caso di virus **polimorfi**.

In virtù dei primi due step, esistono diversi tipi di antivirus:

1. gli antivirus di **prima generazione** facevano un confronto delle firme con un database e scansionando i file applicativi alla ricerca di variazioni nella loro lunghezza rispetto alle lunghezze standard prese dal database;
2. gli antivirus di **seconda generazione** utilizzano euristiche. Ricercano frammenti di codice statisticamente associati a virus e ricercano violazioni di integrità, appendendo un checksum ad ogni file applicativo;
3. gli antivirus di **terza generazione** ricercano azioni illecite rimanendo sempre residenti in memoria. Identifica azioni illecite, bloccandole non appena siano rilevabili. La persistenza in memoria dell'antivirus comporta un rallentamento delle prestazioni;
4. gli antivirus di **quarta generazione** usano insiemi di tecniche già viste nelle prime tre generazioni, ri-regolando il controllo d'accesso per limitare la propagazione di un virus trovato. Ad esempio, trovato un qualunque virus, vengono tolti i permessi di scrittura a qualunque utente.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

9 - SSL e TLS

SSL è un protocollo di sicurezza utilizzato per la distribuzione di una chiave di sessione. SSL e TLS non sono nati per HTTP, ma sono stati portati dentro HTTP. HTTPS è infatti HTTP over TLS: HTTPS è la versione di SSL portata su HTTP.

I termini SSL e TLS sono, rispettivamente, **Secure Socket Layer** e **Transport Layer Security**.

HTTPS

RFC 2818 specifica che HTTPS, HTTP over TLS:

- usa la porta 443;
- cifra contenuti, form, cookie e gli header;
- prevede che il server invia al browser l'header **HSTS** (**HTTP Strict Transport Security**). HSTS è un header che costringe i browser a comunicare con un sito usando **esclusivamente** HTTPS. Quando il browser riceve l'header HSTS, memorizza la direttiva: nel momento in cui il client provi a comunicare con HTTP, il browser trasforma la richiesta usando HTTPS. HSTS blocca lo stripping del primo tipo.

Gli header HTTPS non sono sempre esistiti, e nelle prime versioni di SSL non c'è.

Il primo tipo di stripping SSL è uno stripping banale, che prevede un passaggio da HTTPS a HTTP. Ad esempio, l'utente vuole andare su `https://securesite.com`, ma un MitM fa un downgrade della risposta e invia a `http://securesite.com`. La tendenza di alcuni browser moderni è nascondere il protocollo utilizzato, sotto l'assunzione che oggi HTTPS è de facto uno standard. Questo stripping viene evitato mediante HSTS.

Un secondo tipo di stripping SSL prevede l'attacco mediante DNS, facendo sì che invece di essere redirezionato su `https://securesite.com` l'utente venga mandato su `http://secureesite.com`, con una "e" in più. In questo caso, HSTS non protegge dallo stripping.

SSL

SSL fornisce uno strato sicuro posizionandosi tra il livello di trasporto e il livello di applicazione. Esistono applicazioni portate su SSL per ogni ambito, quali POP3S, HTTPS e FTPS.

SSL nasce come prodotto commerciale di Netscape, che fornisce SSL versione 2 nella versione 1.1 del loro browser Netscape Navigator 1.1; SSL poi passa alla versione 3.0 nel 1995. Nel 1999 ci si pone il problema di generalizzare e standardizzare SSL, per cui nasce il working group TLS dell'IETF. Il protocollo SSL viene implementato in maniera standardizzata in TLS 1.0: SSL infatti non è oggi utilizzato, in quanto si utilizza TLS.

Suite del protocollo SSL

SSL è una suite di quattro protocolli.
Alla base vi è il **record protocol**. Costituisce la parte di SSL che esibisce il layer di sicurezza.
Successivamente vi è l'**handshake protocol**, un protocollo di sicurezza responsabile della distribuzione della chiave di sessione. Una prima esecuzione dell'handshake protocol stabilisce la chiave di sessione, grazie alla quale il record protocol può stabilire lo strato sicuro di comunicazione.
Il **change cipher spec protocol** è il protocollo che segnala esplicitamente la transizione dallo stato di comunicazione non crittografato a quello crittografato, indicando che i messaggi successivi saranno protetti utilizzando i parametri di cifratura e le chiavi di sessione stabiliti durante la fase di handshake.
Infine, l'**alert protocol** segnala errori e chiusure di connessione tra i peers. Gli alert sono cifrati e hanno due livelli: warning e fatal. Un fatal alert chiude sempre la connessione.

SSL Record Protocol (SSL RP)

SSL record protocol fornisce tre servizi per le connessioni SSL:

- **confidenzialità**, definendo la chiave segreta condivisa utilizzata per l'encryption dei payload SSL;
- **integrità e autenticazione del messaggio** per la formazione del MAC.

Il record protocol prende un messaggio applicativo che dev'essere trasmesso, lo frammenta in record gestibili, opzionalmente comprime i dati, applica un MAC, cifra e aggiunge SSL Record Header.
La frammentazione del messaggio prevede la divisione in blocchi di dimensione $\leq 2^{14}$ byte.
La compressione dev'essere **lossless** e non deve incrementare la lunghezza del contenuto di più di 1024 byte. SSL non prevede specifiche di algoritmi di compressione, per cui di default è impostata a

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

`null`, cioè nessuna compressione. La compressione avviene solo per motivi di efficienza.
Il MAC è calcolato sui dati compressi. Una chiave segreta condivisa è utilizzata per il MAC. Il calcolo prevede i seguenti passi:

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

dove:

- `||` indica la concatenazione;
- `MAC_write_secret` è la chiave segreta condivisa;
- `hash` è una funzione crittografica di hash. È MD5 o SHA-1;
- `pad_1` è il byte `0x36` ripetuto 48 volte per MD5 e 40 volte per SHA-1;
- `pad_2` è il byte `0x5C` ripetuto 48 volte per MD5 e 40 volte per SHA-1;
- `seq_num` è il numero di sequenza del messaggio;
- `SSLCompressed.type` è il tipo di messaggio specificato in SSL Record Header;
- `SSLCompressed.length` è la lunghezza del frammento compresso;
- `SSLCompressed.fragment` è il frammento.

Messaggio e MAC vengono crittografati utilizzando la crittografia simmetrica. L'encryption non deve aumentare la lunghezza del contenuto di più di 1024 byte, per cui la massima lunghezza ammessa è $2^{14} + 2048$.

Infine, viene aggiunto **SSL Record Header**, prevede quattro campi:

- **content type**, a 8 bit. Può essere di tipo Handshake, Alert, Cipher Change Spec o Application;
- **major version** (8 bit);
- **minor version** (8 bit);
- **compressed length**, la lunghezza in byte del frammento. Il valore massimo è $2^{14} + 2048$ (16 bit).

Lo schema di processamento è il seguente:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

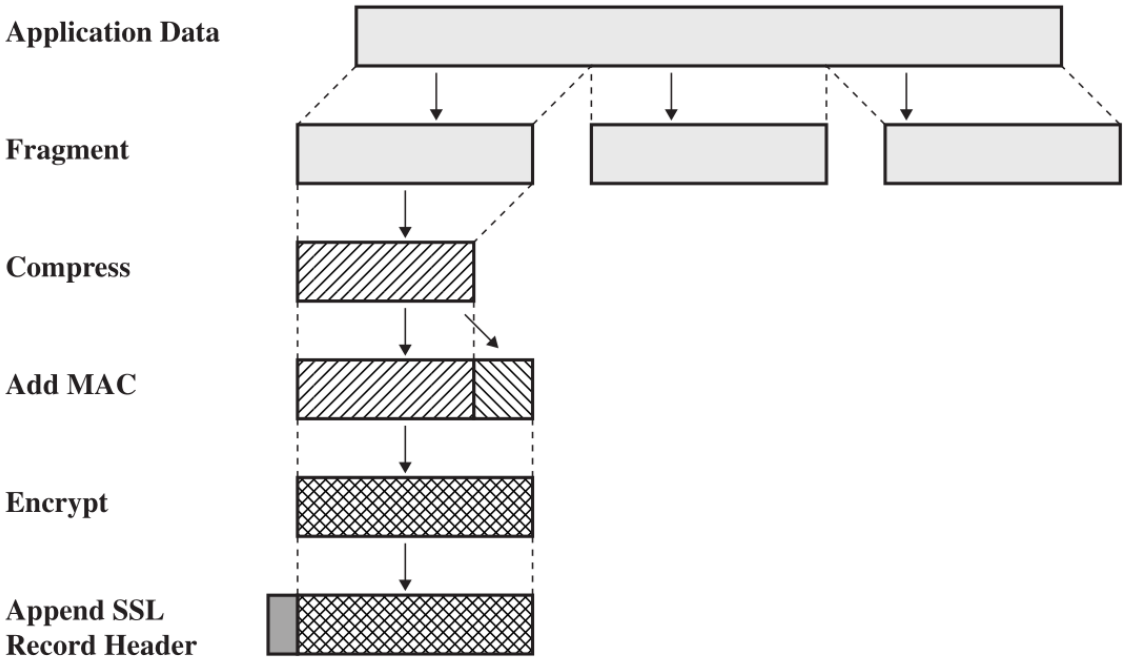


Figure 1: Schema SSL RP

SSL Change Cipher Spec Protocol (SSL CCSP)

Change Cipher Spec è un componente critico la cui funzione è segnalare la transizione dallo stato iniziale non crittografato a uno stato crittografato usando i parametri crittografici negoziati. Il protocollo di handhsake prevede la negoziazione degli algoritmi crittografici, lo scambio di chiavi e l'autenticazione delle parti comunicanti; a questo punto, CCS segna che i parametri entrano in vigore. Per tale motivo, solitamente CCS è visto come parte dell'handshake protocol.

Dopo che il server e il client hanno scambiato le chiavi e che il server, opzionalmente, ha verificato il cliente mediante certificato, il client invia un messaggio "Change Cipher Spec" per indicare che i messaggi successivi verranno crittografati usando le chiavi di sessione negoziate; il server risponde con il proprio messaggio "Change Cipher Spec" per segnalare che i suoi successivi messaggi saranno scambiati con chiavi di sessione.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

SSL Alert Protocol (SSL AP)

SSL Alert Protocol è il protcoollo utilizzato per segnalare cambi di stato dei peer durante la connessione. Gli alert avvengono su SSL Record Protocol, per cui sono protetti. Ogni messaggio è costituito da due byte: livello e codice. Il livello può essere fatal e warning. In caso di livello fatal, la sessione SSL è immediatamente terminata. Esempi di fatal sono `unexpected_message`, `bad_record_mac`, `decompression_failure`:

- `unexpected_message` indica che il messaggio ricevuto è inatteso, dovuto ad un errata implementazione di uno dei partecipanti nella comunicazione SSL;
- `bad_record_mac` indica un errore nell'interlocutore che riceve un record con un MAC errato;
- `decompression_failure` indica un errore nella fase di decompressione del frammento.

SSL Handshake Protocol (SSL HP)

SSL Hanshake Protocol è il protocollo di sicurezza per garantire confidenzialità, autenticazione e integrità. Durante l'handshake:

- vengono stabilite le suite di crittografia utilizzate;
- viene autenticata l'identità del server tramite il certificato;
- vengono generate chiavi di sessione per usare la crittografia simmetrica dopo il completamento dell'handshake.

Si stabilisce un **segreto master** per derivare il **MAC write secret** e la **chiave usata per l'encryption**.

I messaggi dell'handshake protocol prevedono il seguente formato:

1. **type**, 1 byte. Esistono 10 tipi di messaggi, e sono `hello_request`, `client_hello`, `server_hello`, `certificate`, `server_key_exchange`, `certificate_request`, `server_done`, `certificate_verify`, `client_key_exchange`, `finished`;
2. **length**, 3 byte;
3. **content**, a lunghezza variabile.

Ogni messaggio dell'handshake è trasportato all'interno di un Record Protocol, e poiché i parametri di sicurezza non sono ancora noti ogni messaggio è trasportato in cleartext. Distinguiamo diverse fasi:

1. nella fase uno il client vuole stabilire la connessione con il server. Il client invia un messaggio `client_hello` al server. Il server risponde con un `server_hello`. Si tratta di un accordo a due vie:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- il client invia versione supportata, una nonce random per evitare replay attack, id di sessione (0 per nuova sessione, diverso per ripristinare una sessione per ragioni di efficienza in quanto si evita l'handshake), cipher suite (Cipher Spec e algoritmo di key exchange) e metodi di compressione supportati;
- il server risponde con il più basso numero di versione supportato tra client e server (sempre per motivi di efficienza), nonce random per evitare replay attack, id di sessione (ne genera uno nuovo se riceve 0, altrimenti invia lo stesso), cipher suite e metodo di compressione scelti.

2. nella fase due il server invia il certificato, manda la propria metà per lo scambio di chiavi (e.g. Diffie-Hellmann) e richiede il certificato per l'autenticazione del cliente. Questa fase autentica il server e dà inizio lo scambio delle chiavi. I messaggi inviati dal server sono, in ordine:

    1. `certificate` per l'invio del certificato;
    2. `server_key_exchange` per la propria parte dell'algoritmo di scambio di chiavi;
    3. `certificate_request` per richiedere il certificato del client;
    4. `server_hello_done`;

3. la terza fase prevede l'invio del certificato da parte del client e 48 byte di **pre-master secret**. Firma poi il traffico inviato fino ad ora, chiedendo al server la verifica della firma. I messaggi scambiati sono:

    1. `certificate`, per l'invio del certificato client. Un alert è generato se il certificato è vuoto;
    2. `client_key_exchange`, cioè il pre-master secret;
    3. `certificate_verify`, cioè la firma del traffico;

4. la quarta fase prevede l'invio da parte del client e del server del messaggio di CCS. Successivamente, client e server inviano il digest di tutti i messaggi scambiati, utilizzando le nuove cipher spec negoziate durante l'handshake. Client e server mandano, pertanto, `change_cipher_spec` e `finished`, dove `finished` include il digest di tutto il traffico.

Nella Cipher Suite vengono specificati:

1. **Cipher Spec**, definendo l'algoritmo di encryption, l'algoritmo di MAC e il tipo di cifratura;
2. **Key Exchange Algorithm**, che può essere uno tra:

    1. **RSA key-exchange**;
    2. **Diffie-Hellmann anonimo**, ossia la versione tradizionale che soffre di MITM;
    3. **Diffie-Hellmann effimero**, con parametri pubblici Y_a, Y_b che vengono cambiati ad ogni sessione e firmati digitalmente dai peer. Questo permette la **Perfect Forward Secrecy**

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

(**PFS**), in quanto i parametri X_a e X_b generati da client e server vengono cancellati a fine sessione;

4. **Diffie-Hellmann fissato**, con parametri pubblici Y_a, Y_b fissati derivati dai certificati. Più precisamente, i parametri pubblici α, β, Y_b sono inclusi nel certificato del server, ed il parametro X_b è **fisso**. Pertanto, **non vi è PFS**: se l'attaccante riesce ad ottenere X_b, può decifrare ogni messaggio, dato che per ogni connessione è sufficiente calcolare la chiave $Y_a{}^{X_b} \mod \beta$.

Il master secret viene creato con la concatenazione di tre applicazioni di MD5, creando un segreto di 48 byte. La generazione prevede la formula:

```
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))
```

Il pre-master secret è usato come seed, mentre i valori random utilizzati per la freshness vengono usati come salt. Per cambiare ogni volta gli hash, vengono aggiunti A, BB e CCC.

Per il MAC si costruisce il `key_block` per la realizzazione della `write_key` e della `write_MAC_secret`:

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

Il processo continua fino a quando non vengono realizzati le chiavi richieste.

Osserviamo come il procedimenti descritto preveda l'utilizzo del solo pre-master secret inviato dal client. Questo ha a che vedere con il fatto che **il protocollo sottointeso è RSA**: il server invia la chiave pubblica con il suo certificato, così che il client possa inviare la chiave di sessione al server. Nel caso in cui venga impiegato Diffie-Hellman, il server invia uno dei due valori Y usati per la derivazione del segreto condiviso mediante potenze dell'algebra modulare.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Ridurre la latenza di rete in SSL

La latenza di rete è un problema, per cui solitamente si salva lo **stato di sessione** e lo si riprende durante una nuova connessione.

Lo stato di sessione è definito da:

- **id di sessione**, scelto in maniera arbitraria dal server;
- **certificato del peer** in formato X.509;
- **metodo di compressione**;
- **cipher spec** a livello di algoritmi MAC e encryption dei dati;
- **master secret** di 48 byte;
- **flag** di ripristino.

Lo stato di sessione viene salvato alla fine della fase quattro. Per il ripristino, il cliente invia l'id di sessione da ripristinare. In questo modo, la fase 2 e 3, **più computazionalmente pesanti**, vengono saltate e solo la fase 1 e 4 vengono eseguite. Nuovi key block sono calcolati dal master secret salvato nello stato di sessione e dalle nuove nonce casuali generate.

TLS

TLS è stato introdotto nel 1999 con RFC 2246. Differisce da SSL per:

- algoritmo di MAC utilizzato;
- funzioni di generazione di valori pseudocasuali;
- ulteriori alert e Cipher Suites.

RFC 2104 prevede che l'HMAC sia calcolato con:

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

dove:

- H è la funzione hash scelta;
- M è il messaggio, che secondo RFC 5246 è la concatenazione già vista in SSL, ossia `seq_num || SSLCompressed.type || SSLCompressed.length || SSLCompressed.fragment`;

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

- K^+ è la chiave segreta con padding di zero alla sinistra così che la sua lunghezza sia pari alla lunghezza di blocco del codice hash;
- ipad è il numero `0x36` ripetuto 64 volte;
- opad è il numero `0x5C` ripetuto 64 volte.

Per la costruzione dei keyblock si utilizza uno **Pseudo-hash** o P-hash. Una funzione P-hash è una funzione costruita mediante una catena ricorsiva di HMAC. Partendo da un seed iniziale, si producono gli output come:

$$A(1) = \text{HMAC}(secret, seed)$$

$$A(2) = \text{HMAC}(secret, A(1))$$

$$A(3) = \text{HMAC}(secret, A(2))$$

...

Ogni singolo output viene concatenato ad un seed, e viene ricalcolato l'HMAC sul segreto $secret$ e sulla concatenazione. L'output finale è l'unione di blocchi del tipo:

$$\text{HMAC}(secret, A(i)||seed)$$

Quindi il primo blocco è $\text{HMAC}(secret, A(1)||seed)$, il secondo $\text{HMAC}(secret, A(2)||seed)$ e così via. Si procede fino a quando l'output finale non raggiunge la lunghezza desiderata. Lo schema di generazione può essere riassunto con la figura sottostante:

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

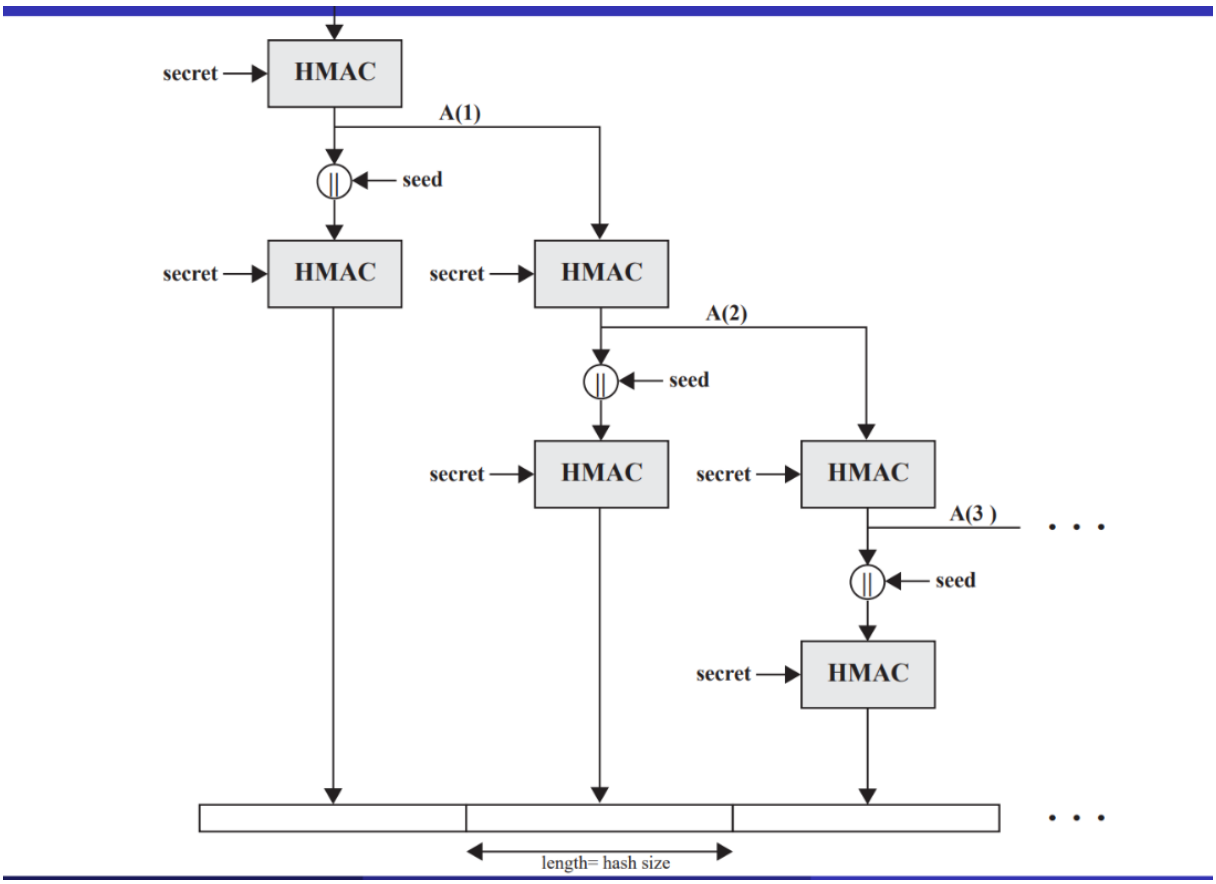


Figure 2: P-hash

Le P-hash sono impiegate per la **funzione pseudocasuale** usata per la generazione del master secret e del key block:

$$PRF(secret, label, seed) = \text{P_MD5}(S1, label||seed) \oplus \text{P_SHA-1}(S2, label||seed)$$

Si utilizzano per gli HMAC dei P-hash l'hashing MD5 e l'hashing SHA-1. Il segreto viene diviso in $S1, S2$. Nella formula soprastante:

- $seed$ è la concatenazione dei due valori casuali scambiati tra client e server;
- $secret$ è il pre-master secret per ottenere il master secret, oppure il master secret per ottenere il keyblock;
- $label$ è "master secret" per avere la master secret o "key block" per il keyblock.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it

Evoluzione di TLS

TLS 1.1 è molto simile a TLS 1.0. Nel 2008 esce TLS 1.2, che permette l'utilizzo di estensioni da aggiungere dopo i metodi di compressioni in fase uno (RFC 5246); tra questi vi è HSTS. Inoltre, con TLS 1.2:

- viene rivisitato l'impiego di MD5 a causa della debolezza dell'algoritmo;
- la combinazione di MD5 e SHA-1 nel calcolo dell'HMAC viene sostituita con un singolo hash;
- gli elementi firmati includono un campo che specifica l'algoritmo di hash impiegato;
- la combinazione di MD5 e SHA-1 nella funzione pseudo-random viene sostituita con funzioni pseudo-random cipher-suite-specifiche.

Materiale fornito da noistudiamo.it | Contatti: info@noistudiamo.it