Questo è un documento scritto con il fine di ripassare tutto il programma del corso di Sicurezza Informatica
Le pagine indicate nell'indice si riferiscono alle rispettive pagine da cui ho dato un occhiata su tkanon.ninja

Indice

Introduzione

Sicurezza informatica: tutto ciò che ha a che fare con il controllo di azioni deliberate che provocano danni (security, non safety). Infatti **security** si occupa di attacchi "intenzionali", dei disastri si occupa la **safety**.

Gestione del rischio cyber:

- Affrontare i problemi di cybsec si concretizza in un'analisi del rischio
- Semplificando **RISCHIO = PROBABILITÀ ⊗ IMPATTO**
- Un buon sicurista deve analizzare correttamente probabilità e impatto di tutti gli eventi possibili e fornire mitigazioni (che devono essere convenienti)

Proprietà della sicurezza:

- **Confidentiality**: riservatezza
- **Integrity**: che comprende Authority
- **Availability**: disponibilità

Minacce possibili:

- **Disclosure**: riservatezza
- **Disruption**: disponibilità
- **Deception**: Integrità
- **Usurpation**: uso non autorizzato del sistema attaccato

Due blocchi necessari alla messa in sicurezza:

- Politiche di sicurezza: chi può fare cose, dichiarazione di intenti
  Casi di fallimento delle politiche di sicurezza:
    - **Porosità**: esiste un vettore di attacco non considerato

    - **Vulnerabilità**: errore nella politica o nel meccanismo di sicurezza (errori umani compresi)
- Meccanismo di sicurezza: strumenti o procedure per far rispettare le politiche di sicurezza
  Strategie di meccanismi di sicurezza (non sono applicabili sempre tutte):
    - **Prevenzione**
    - **Rilevazione**
    - **Reazione**
    - **Ripristino**

Cybsec killcahin:

- Recon
- Weponisation
- Delivery
- Exploitation
- Installation
- Command & control
- Exfiltration

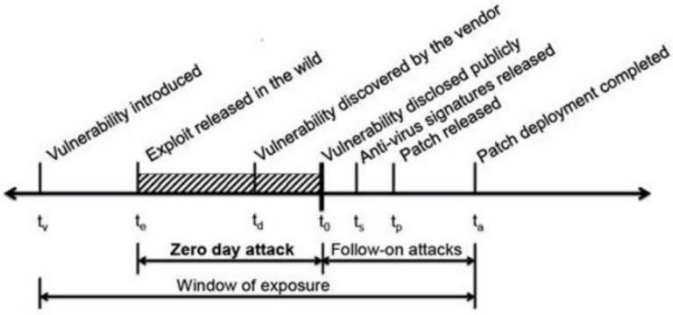
Offensive security

Analisi fatta dal ruolo dell'attaccante. Non è facile calcolare a tavolino l'impatto di un attacco (può fare più danni del previsto).

Vulnerabilità zero-day: vulnerabilità sconosciuta a chi è interessato a sfruttarla.
Window of exposure: periodo temporale tra il momento dell'introduzione della vulnerabilità e il momento di rilascio di una patch da parte del fornitore.
Zero-day attack: attacco effettuato durante la finestra di opportunità.
Follow-on attacks: periodo di intensificazione degli attacchi a causa della pubblica esposizione della vulnerabilità.

Tre livelli di approfondimento:

- **Vulnerability Assesment**
    - Rileva vulnerabilità già note
    - Non va oltre: non sfrutta vulnerabilità che potrebbero essere state scoperte dalla vulnerabilità di partenza
- **Penetration Testing**
    - È importante definire il perimetro e le informazioni che potrebbero essere scoperte
    - Definire da quale set di permessi iniziare l'attacco
    - Scegliere il metodo di test considerando i limiti dettati dall'oggetto da testare
- **Red Team Operations**: con "red team" si fa riferimento ad un team che ha lo scopo di trovare le vulnerabilità per comunicarle all'azienda



Preparazione al Penetration Testing:

- **Reconnaissance**: raccolta di tutte le informazioni utili per l'attacco, estensione del perimetro di test e preparazione degli strumenti

- **Enumeration**: delimitazione del perimetro di test e verifica puntuale delle risorse e delle loro proprietà

La prima fase è **OSINT** (Open Source INTelligence): raccolta di informazioni sul bersaglio tramite fonti pubbliche.
Sapendo che ogni host ha un indirizzo IP associato, lo si può cercare informazioni su di esso attraverso IANA.
È utile consultare anche i record DNS per scovare indirizzi IP registrati, l'esistenza e la collocazione di specifici server applicativi, l'esistenza di sottoreti non direttamente raggiungibili... Per fare ciò si possono usare comandi:

- Lookup di base: `lookup` quali host, `dig` e `nslookup`.
- Strumenti di ricerca che includono guessing e forza bruta: `dnsenum`, `dnsmap`, `dnsreecon`, `fierce`...

Una volta che si sanno gli indirizzi da analizzare (o ci si trova nella rete del bersaglio):

- Si può effettuare un `ping` un indirizzo alla volta, con il rischio che possa essere bloccato da router/firewall se non ci si trova nella rete locale o con il rischio che venga ignorato dall'host. Sarà dunque necessario fare sniffing passivo tramite `wireshark`, `tcpdump`...
- Si possono sennò fare scansioni massive attraverso `masscan`
- Una volta trovati gli host attivi, vanno cercate le porte aperte usando tool quali `nmap` che offre la scansione contemporanea di range di indirizzi e porte usando un set predefinito di porte "più popolari" e usando diverse tipologie di scansione

`lsof`: lists of open files Esempio: `lsof -i -n | egrep 'COMMAND|LISTEN|UDP'`
`netstat`: mostra lo stato delle socket Unix e di rete
`ss`: rimpiazzo recente di `netstat`

Attacchi al processo di autenticazione e autorizzazione

Regola delle tre A per il controllo degli accessi:

- **Identificazione**: implicita e deve essere staccata dall'autenticazione (Esempio: dati biometrici, social security number...)
- **Autenticazione**: attribuzione certa dell'identità di un soggetto che utilizza le risorse. Si basa su tre fattori:
  - Qualcosa che l'utente conosce (pin, domanda segreta, password...)
  - Qualcosa che l'utente possiede (carta magnetica, hard token, telefono...)
  - Qualcosa che l'utente è (biometria, posizione geografica...)
- **Autorizzazione**: verifica dei diritti di un soggetto di compiere una determinata azione
- **Auditing**: la sicurezza è un processo, quindi l'auditing serve per capire quanto le politiche siano fondamentali/funzionali

Due tipi di autenticazione:

- **Autenticazione passiva**: è presente un *Prover* che possiede un segreto e lo manda al *Verifier* che è l'unico a sapere il segreto. È importante usare canali di comunicazione sicuri per evitare problemi di intercettazione o replay attack. Esiste anche il problema del furto del segreto se non mantenuto bene dal *Verifier*; dunque, il *Verifier* non sarà in possesso del segreto in chiaro ma dell'hash calcolato su di esso, cosicché un attaccante non può sapere la password raccogliendo informazioni dal *Verifier* ma il *Verifier* può continuare a discriminare una password giusta da una sbagliata. Ora il segreto è resistente a rainbow tables (tabelle con pezzi di hash già risolti), dictionary attacks.
  Essendo hash una funzione deterministica serve evitare che gli attaccanti possano precalcolare l'hash: viene introdotto un **salt**, ovvero una variazione random nella password che cambia l'hash della password. Il salt cambia ogni variazione della password da parte del *Provider* e il *Verifier* è a conoscenza del salt.
- **Autenticazione attiva**: il *Provider* convince il *Verifier* di possedere un segreto senza che questo sia intercettabile o ricavabile (S-Key, OTP, challenge-response…)

Una volta entrati in un sistema si cerca di elevare i propri privilegi di sistema sfruttando vulnerabilità locali. In Linux ogni utente appartiene ad almeno un gruppo e ogni file, descritto da un i-node, ha un set di 12 bit di autorizzazioni memorizzato nell'i-node stesso. I bit in questione sono:

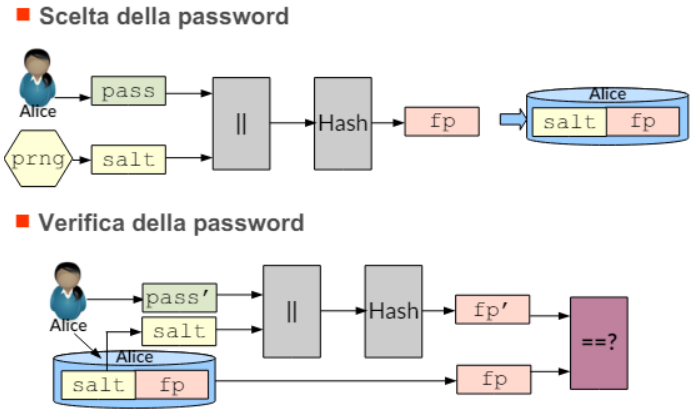
- **R** (Read): lettura di un file o dell'elenco dei file di una directory
- **W** (Write): scrittura di un file o aggiunta/cancellazione/rinomina di file in una directory
- **X** (Execute): esecuzione del file come programma o lookup dell'i-node nella directory
- **SUID** (Set User ID): per i file se è settato fa sì che al lancio il SO generi un processo che esegue il file con identità dell'utente proprietario invece che con l'identità di colui che lo lancia. Per le directory non viene usato
- **SGID** (Set Group ID): per i file si comporta come il SUID ma esegue il file con permessi del gruppo del proprietario del file. Per le directory se l'utente appartiene anche al gruppo proprietario della directory, allora ogni file che l'utente crea lì assumeranno come gruppo quello proprietario della directory
- **STICKY**: per i file suggerisce al SO di tenere in cache una copia del programma (obsoleto). Per le directory impone che i file al suo interno siano cancellabili solo dai proprietari, sovrascrivendo il permesso di *Write* della cartella

I file con SUID o SGID abilitati vanno sorvegliati visto che conferiscono privilegi alti a chiunque li esegua.

Nell'i-node sono presenti anche le Access Control Lists (ACL), delle liste di utenti e gruppi ai quali sono assegnate delle maschere di permessi.

A root sono assegnati permessi speciali dette Capabilities: sono 41 e riguardano molteplici aspetti di controllo delle risorse di calcolo e dei processi e dell'accesso alla rete

`adduser`: crea un utente
`addgroup`: crea un gruppo
`passwd`: cambia la password di un utente
`find`: usato per trovare file con determinati permessi Esempio: `find / -type f -perm +6000`
`setfacl`: imposta le ACL
`getfacl`: mostra le ACL



Applicazioni web

A livello web le vulnerabilità possono essere:

- Lato client: errori di definizione del perimetro di interazione con i server, manipolazione del DOM per ingannare l'utente…
- Protocollo: sicurezza del canale, uso di cookie e serializzazione…
- Lato server: errori di gestione richieste http, di controllo dell'accesso, di interpretazione dei dati…

OWASP (Open Web Application Security Project) è un progetto nato sull'web per la gestione delle vulnerabilità, divise in:

- **A1 - Broken access control**: dati e azioni sensibili esposte basandosi su *security through obscurity* (Esempio: un server invia una risorsa al client usando un link, il client cambiando un dettaglio del link ha accesso a risorse di altri utenti). Questa vulnerabilità si chiama **IDOR** (Insecure Direct Object Reference), in cui un oggetto viene erogato solamente perché si sa come si chiama. Un caso particolare di IDOR è **File Disclosure**, a metà strada tra l'injection e l'access control, in cui l'oggetto è un elemento del file system (Esempio: path traversal)
- **A2 - Cryptographic failures**: errori nella gestione di un dato sensibile che va protetto *at rest* o *in transit* (Esempio: una vittima mette i dati della sua carta di credito in un sito sicuro, i dati sono scritti nel server ma protetti male e accessibili da tutti i membri dell'IT, rendendo così i dati disponibili ad un insider)
- **A3 - Injection**: l'attaccante usa i campi di testo per ridirigere l'esecuzione. Funziona se dietro al web server si ha un db relazionale che si basa sull'input dell'utente. Per mitigare questa vulnerabilità è necessario controllare l'input. Tre possibili esecuzioni:
    - **Reflected XSS**: usato in relazione con il phishing. Un malintenzionato invia ad una vittima un link che punta ad un server legittimo, però il link incorpora un payload malevolo (Esempio: il link indirizza la vittima ad una pagina web con una form già fabbricata dal malintenzionato che gli permetterà di accedere a dati sensibili della vittima)
    - **Stored XSS**: simile al reflected XSS ma il payload malevolo è inviato dal malintenzionato direttamente al server che lo immagazzina nei suoi db, pronto per essere estratto da una vittima che potenzialmente invia dati sensibili all'attaccante. Per scoprire le vulnerabilità vengono fatte una delle due possibili analisi:
        - White box: analisi statica del codice che si dispone
        - Black box: blind injection dove l'output del server invia in ritorno solo messaggi di in/successo e non dati
    - **DOM XSS**: Molto simile al reflected ma il codice che processa dati in modo non sicuro non è sul lato server ma già nella pagina
- **A4 - Insecure design**: Attira l'attenzione sulla necessità di usare metodi di progetto formalizzati (threat modeling, secure design patterns, reference architectures). Utilizzo del paradigma shift left
- **A5 - Security misconfiguration**: Categoria ampia che raccoglie tutti gli errori di configurazione (troppi privilegi, credenziali di default, funzioni non necessarie installate o abilitate…)
- **A6** - Vulnerable and outdated components
- **A7** - Identification and authentication failures
- **A8** - Software and data integrity failures
- **A9** - Security logging and monitoring failures
- **A10** - Server-side request forgery

Binary exploit

Lo scopo è quello di far eseguire ad un processo operazioni per cui non era stato pensato, con l'obiettivo di:

- Fermare il processo: DOS
- Dirottare il flusso di esecuzione: esecuzione di codice maligno
- Ottenere privilegi di altri utenti

Il codice più vulnerabile è quello scritto in C o C++ essendo linguaggi di livello abbastanza basso che non fanno controlli automatici. In un SO Linux lo spazio di indirizzamento di un processo in memoria è diviso in segmenti:

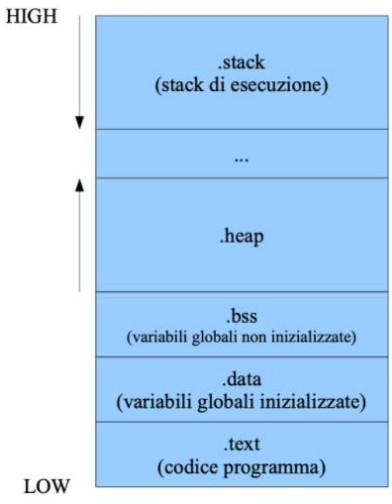
- Segmento .text: contiene il codice eseguibile
- Segmento .data: contiene i dati inizializzati
- Segmento .bss: contiene le variabili non inizializzate
- Stack di esecuzione: contiene i record di attivazione del processo e le variabili locali (cresce verso il basso)
- Heap: contiene le variabili dinamiche (può crescere dinamicamente e verso l'alto)
- Altri segmenti utili al funzionamento: .got, .ctor, .dtor...

L'architettura IA32 (quella su cui ci esercitiamo, little endian) è dotata di 4 registri da 32 bit "general purpose": EAX, EBX, ECX, EDX. Altri registri sono: ESI (source) e EDI (destination) per la copia di dati, EIP (Instruction pointer) e EFLAGS (Status register) per il controllo di flusso. Due registri sono importanti per lo stack:

- **ESP** (Stack pointer): punta all'ultima cella occupata dello stack. Sono possibili operazioni di:
  - PUSH: decrementa ESP di 4 byte e scrive il valore sulla cella puntata
  - POP: recupera il valore della cella puntata da ESP e per poi incrementarlo di 4 byte
- **EBP** (Base pointer): punta all'inizio dello stack locale. Tutte le variabili locali sono referenziate relativamente a EBP. Usato per trovare nello stack le variabili locali

La traduzione C -> Assembly adotta convenzioni standard (convenzione __cdecl):

- Inserimento sullo stack di tutti i parametri attuali di chiamata in ordine inverso rispetto alla signature del metodo
- Chiamata tramite "CALL" salvando l'indirizzo di ritorno sullo stack
- EBP contiene il riferimento per le variabili locali al chiamante, non deve essere perso, ma il chiamato ha bisogno di settarlo al proprio "sistema di riferimento". Il chiamato salva il contenuto del registro EBP ponendolo sullo stack e aggiorna il valore di EBP al contenuto attuale di ESP
- Il chiamato ritorna il risultato delle proprie computazioni nel registro EAX
- È compito del chiamato ripristinare il valore originario di EBP
- È compito del chiamante rimuovere i parametri attuali dallo stack



Stack overflow: l'obiettivo è quello di riempire un buffer allocato sullo stack oltre i suoi limiti per far "traboccare" e sovrascrivere aree già scritte. Questo è possibile per via di alcune istruzioni che non fanno controlli. L'attaccante riempie il buffer con dati fittizi (padding), sforandone i limiti, fino ad arrivare a porre sullo stack "nella posizione giusta" un nuovo indirizzo di ritorno dalla chiamata

(sovrascrivendo quello preesistente) cosicché il flusso di controllo non procede con il ritorno al chiamante "lecito" bensì al "nuovo" indirizzo di ritorno posto dall'attaccante (**shell coding** se si ha l'obiettivo di aprire una shell con i privilegi del processo attaccato).

Per mitigare il problema vengono usate diverse contromisure:

- **Canarini**: viene inserito un dato casuale nello stack, prima del buffer, cosicché se un attaccante provasse a iniettare codice maligno verrebbe scoperto perché il processo vede che il dato è stato sovrascritto. [Validi ma rallentano ed esistono modi per aggirarli]
- **NX Stack**: feature fornita dall'HW ma che deve essere supportata dal SO, permette di usare stack non eseguibili. [vari attacchi fanno uso legittimo dello stack]
- **ASLR** (Address Space Layout Randomization): protezione offerta dal SO in cui si rende casuale l'indirizzo di partenza dei segmenti. [incompleto]
- **ROP** (Return-Oriented Programming)

Sicurezza delle comunicazioni

ATTACCHI PASSIVI: non alterano i dati in transito, utili all'attaccante per ampliare la sua conoscenza violando la riservatezza:

- **Scanning**: si possono scansionare porte, host… Normalmente rumorosa, ma si possono cambiare parametri per rendere più silenziosa la scansione
- **Sniffing**: intercettazione dei pacchetti in transito su una rete (wireshark). Richiede avere l'accesso fisico alla rete.
- **MAC flooding**: Uno switch idealmente invia il traffico di dati solo alla porta del destinatario, ma in caso la sua tabella (porta – MAC address) si riempia allora elimina il LRU (Last Recent Used). Quando i due host vorranno comunicare, non sarà presente la copia in tabella, rendendo così necessaria la ricomunicazione di porta e MAC address, rendendoli così disponibili all'attaccante

Essendo le reti WiFi facilmente vittime di sniffing, ci sono quattro principali generazioni di protezione:

- **WEP** (Wired Equivalent Privacy): chiave simmetrica precondivisa
- **WPA** (WiFi Protected Access): creata per rimediare immediatamente a WEP
- **WPA2**: Considerato sicuro fino alla scoperta di una grande vulnerabilità nel 2017
- **WPA3**: lo standard de facto attuale, considerato sicuro essendo attaccabile solo in modo molto complesso

ATTACCHI ATTIVI: minacciano l'integrità, l'autenticità o la disponibilità di reti e sistemi. Essi si dividono per layer

Livello 2:

- **MAC spoofing**: l'attaccante assume l'identità di un dispositivo a livello di indirizzo fisico, per così facilmente bypassare ACL e ottenere tutto il traffico destinato alla vittima. Limitato dalla LAN
- **ARP poisoning**: convincere un host che l'IP di una vittima è associato al MAC dell'attaccante

Livello 3 (entrambi utili per bypassare ACL e per dirottare le connessioni dopo l'autenticazione):

- **IP spoofing**: l'attaccante assume l'indirizzo IP di una vittima
- **IP hijacking**: tipo la roba del Pakistan Telecom

Livello 4: UDP non fa controlli, ma TCP ha bisogno del numero di sequenza giusto per poter essere hijacked attraverso sniffing o forza bruta. Gli attacchi sono:

- **(D)DoS**: inondazione della vittima di richieste superando la sua capacità, utilizzando botnet, insiemi di computer zombie istruiti a lanciare attacchi DDoS
- **DNS tunneling**: spesso utilizzato in reti con il controllo degli accessi, il DNS lasciato libero viene utilizzato per esfiltrare dati da un computer infettato o per mettere in contatto un bot

Protezione delle comunicazioni

A livello applicativo si usa **HTTPS** (http over SSL [ora TLS]), permettendo di bloccare gli attacchi ma esistono modi per evitare che venga visualizzato l'URL effettivamente visitato o per far accettare al browser qualsiasi certificato.

SSL (Secure Sockets Layer): autenticazione challenge-response basata su crittografia asimmetrica e certificati X.509. Nelle applicazioni web è comune che solo il server provi la sua autenticità al client. Si occupa della cifratura/decifrazione dei record in modo conforme a quanto negoziato

TLS (Transport Layer Security): evoluzione di SSL in cui viene usato un diverso algoritmo di cifratura/decifrazione ma con lo stesso formato di record

Per aggirare TLS ci sono più possibili attacchi:

- **Attacchi omografici**: usare caratteri internazionali simili o uguali tra loro per generare domini (quindi anche certificati) uguali a quelli di servizi famosi. Per risolvere il problema si usa punycode, ovvero una conversione dei caratteri internazionali in sequenze di caratteri base (es: 點看 → xn--c1yn36f)
- **Certificati falsi**: è raro ma pericolosissimo, compromette una CA (Certificate Authority). Soluzioni:
  - **HTTP Public Key Pinning**: limitare le chiavi associabili a un dominio (deprecato)
  - **Certificate Transparency**: utilizza un log pubblico e non modificabile contenente tutte le modifiche di certifica
- **Stripping**: pagine HTTP che inviano dati sensibili via HTTPS possono essere modificate da un MITM (Man In The Middle). Come mitigazione esiste HSTS (HTTPS Strict Transport Security) per il quale il web server dice al client di comunicare solo usando HTTPS

Vulnerabilità a livello di protocollo:

- RC4
- Beast
- Padding Oracle Attacks
- Lucky Thirteen
- POODLE
- DROWN

Vulnerabilità a livello di implementazione:

- Heartbleed

Crittografia classica

La crittografia è un'elaborazione matematica e algoritmica della codifica delle informazioni, usata per prevenire la violazione della **riservatezza** e rilevare la violazione dell'**integrità** e **autenticità**.

Principi di Kerckhoffs:

1. Un cifrario deve essere computazionalmente se non matematicamente indecifrabile

2. L'algoritmo non deve essere segreto, il segreto deve essere la chiave
3. La chiave deve essere semplice e facile da scambiare

Metodi di attacco (crittoanalisi):

- **Forza bruta**: si tira a indovinare
- **Solo testo cifrato**: si eseguono analisi statistiche si una grande quantità di materiale cifrato e se ne usano le indicazioni per individuare quale p probabilmente corrisponde ad un dato c
- **Testo in chiaro noto**: ci si procura in qualche modo sia dei testi cifrati, sia i corrispondenti testi in chiaro e si cerca di dedurre D analizzando le varie coppie
- **Testo scelto**: si può scegliere testo da cifrare o da far decifrare per ottimizzare il procedimento di deduzione della chiave
- **Rubber hose**: minacce

Di fronte ad un testo cifrato con algoritmo noto, un crittoanalista può:

- Analizzare le proprietà statistiche del testo
    - **Robustezza**: capacità dell'algoritmo di occultare le proprietà statistiche del testo in chiaro. La robustezza è data da:
        - Confusione: misura il grado in cui la struttura della chiave viene resa irriconoscibile nel testo cifrato
        - Diffusione: misura il grado in cui le proprietà statistiche degli elementi del testo in chiaro vengono sparse sugli elementi del testo cifrato
- Cercare la chiave tra tutte quelle possibili
    - **Sicurezza assoluta**: rendere totalmente indistinguibile la chiave giusta dalle altre
    - **Sicurezza computazionale**: rendere troppo oneroso il processo di ricerca della chiave

Algoritmi classici:

- Sostituzione monoalfabetica
- Trasposizione
- Sostituzione polialfabetica
- OTP

Crittografia moderna

- **Cifrari a blocchi**
    - All'aumentare della dimensione di un blocco riduce la riconoscibilità statistica del testo in chiaro
    - Utilizzano S-box (substitution) e P-box (permutation, ovvero trasposizione) per più round
    - Standard storico DES, sostituito da AES (Advanced Enryption Standard)
    - Cifrare blocco per blocco è male, stesso testo in chiaro risulta stesso testo cifrato, facilitando l'analisi dell'attaccante. La soluzione sta nel cifrare un blocco modificandolo col contributo del cifrato precedente (CBC, CFB, OFB) oppure realizzando l'equivalente a blocchi di un cifrario a flusso
- **Cifrari a flusso**
    - Utilizzano un seed segreto, viene poi generato uno stream pseudocasuale di bit
    - Diventa quindi un OTP
- **Funzioni hash**
    - Funzioni difficilmente invertibili (lossy)

    - È presente robustezza se rispetta le seguenti proprietà:
        1. **One-way**: dato un hash non è possibile (tranne che usando forza bruta) generare un documento con lo stesso hash
        2. **Collision-free**: non si può trovare facilmente una coppia di documenti con lo stesso hash
- **RSA 1977**
    - Vengono scelti due numeri primi p e q
    - Il modulo viene calcolato $n=p \otimes q$
    - Viene scelto a caso un numero d e viene calcolato un numero e tale che $e \otimes d \mod (p-1)(q-1)=1$
    - La chiave pubblica è *(e, n)* e la chiave privata *(d, n)*

Chiavi crittografiche

Le **chiavi** sono l'elemento fondamentale per la crittografia. Il loro ciclo di vita è formato da:

- **Generazione**: per quelle simmetriche è sufficiente un buon generatore di numeri casuali, per quelle asimmetriche servono due numeri primi casuali. I computer sono però deterministici, dunque si parla di PNRG (Pseudo Random Number Generator). Esistono tre test del NIST per verificare la casualità, Frequency test, Runs test e Maurer's test
- **Memorizzazione**
    - **Chiave di memorizzazione**
        - Se persa perdi tutti i dati
        - Se fai backup aumenti l'esposizione ad attacchi
        - Accorgimenti possibili: cifratura con passphrase, HSM (Hardware Security Module), key escrow, secret sharing
    - **Chiave di firma**
        - Se persa basta generare e distribuirne una nuova
        - Non serve dunque un backup
        - Per non essere usata contro la volontà del titolare vengono usati HSM, passphrase
- **Distribuzione**
    - **Chiavi simmetriche**
        - Scambio manuale
        - KDC (Key Distribution Center): scambio manuale con KDC, il KDC negozia la chiave di comunicazione tra due utenti in un canale sicuro. Il KDC diventa critico se vengono compromesse tutte le comunicazioni del sistema
        - Scambio di Diffie-Hellman
        - Computazionalmente sicuro: il log discreto è complicato da fare
    - **RSA**: al MITM non interessa decifrare le chiavi che si stanno inviando due peer, può però distribuire la propria chiave pubblica e fare da "passacarte" tra i peer senza che loro se ne accorgano. Dunque, nel caso asincrono c'è un problema di autenticità, non riservatezza. Servono dunque delle soluzioni per mitigare il problema

- Modello **web of trust**: l'autenticità di una chiave pubblica è testimoniata da altri utenti. Il vantaggio è la non presenza di un'entità "super partes", però non è ben scalabile
- Modello **infrastrutturale**: esiste una terza parte fidata che documenta l'associazione. PKI (Public Key Infrastructure)

Le chiavi asimmetriche sono resistenti ad attacchi di forza bruta e di fattorizzazione.

Firewall

I **firewall** sono principalmente pensati per usare la rete come strumento di difesa perimetrale per:

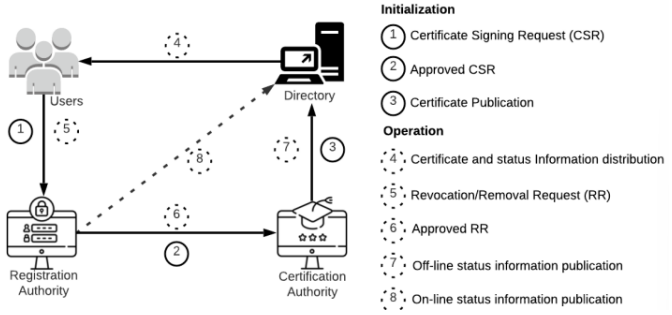
- **Ingress filtering**: impedire l'accesso ai malintenzionati
- **Egress filtering**: per impedire l'esfiltrazione di dati riservati per evitare che i propri sistemi siano usati come base per attaccarne altri

Il firewall è un'architettura che può essere implementata a livello *hw* e *sw*. È consigliato usarlo per implementare una politica **Default deny**, vietare l'accesso a tutto ciò che non è autorizzato. Tecniche di controllo:

- **Traffico**: esaminazione di indirizzi, porte e del protocollo
- **Direzione**: discriminazione a parità di servizio delle richieste che vanno dalla rete interna a quella esterna da quelle che seguono la direzione opposta
- **Utenti**: differenziazione dell'accesso ai servizi in base a chi lo richiede
- **Comportamento**: valutazione dell'uso dei servizi ammessi, per identificare anomalie

Tre tipi di firewall:

- **Packet filter** (PF)
  - Esamina l'header del pacchetto e non il payload
  - Usa regole del tipo "se <condizione> allora <azione>"
  - Vantaggi: velocità e trasparenza agli utenti
  - Svantaggi: uso di regole di basso livello con set di regole complesse, nessun meccanismo per riconoscere gli utenti, facilità di attacchi DoS
  - Vulnerabilità: frammentazione e spoofing
- **Application-level gateway** (ALG)
  - MITM buono che fa da server nei confronti del client
  - Può fare caching
  - Filtraggi avanzati come: permettere o negare specifici comandi, esaminare la correttezza degli scambi protocollari o attivare dinamicamente regole sulla base della negoziazione *C/S*
  - Analisi approfondita dei payload
  - Svantaggi: più lento e pesante di packet filter, non sempre trasparente (con TLS non può esserlo), specifico di un singolo protocollo applicativo
- **Circuit-level gateway** (CLG)
  - Spezzano la connessione a livello di trasporto e diventano endpoint (non intermediari) del traffico, inoltrando i payload senza esaminarli



        - Usato per determinare quali connessioni sono ammissibili dall'interno verso l'esterno
        - Sventa gli attacchi basati sulla manipolazione dell'header
        - Svantaggi
            - Le regole di filtraggio sono limitate a indirizzi, porte, utenti
            - Richiede la modifica dello stack dei client

Questi tipi di firewall sono utilizzabili contemporaneamente, per questo si parla di firewall come architettura.

Due collocazioni:

- **Bastion Host** (BH)
    - Firewall come unico punto di passaggio tra rete interna ed esterna
    - Può servire anche a un PF, ma tipicamente usato per realizzare un ALG o un CLG
- **Personal Firewall**
    - Installato sul client da proteggere
    - Costituiscono un'eccezione sul principio del controllo alla frontiera
    - Spesso sono sistemi "learn by doing"

Topologie di filtraggio:

- **Classica** (rete esterna) --- (firewall) --- (rete interna)
    - Non adatta a reti in cui sono presenti contemporaneamente *Client* e *Server*
- **Screened single-homed BH**
    - Un PF garantisce che solo un BH possa comunicare con l'esterno
    - Il BH implementa un ALG
    - Doppio filtraggio: a livello header (PF) e applicativo (BH)
    - Per prendere il controllo completo della rete interna serve compromettere due sistemi, anche se compromettendo il PF si ha un accesso significativo
- **Screened dual-homed BH**
    - Uguale alla topologia precedente ma ora il BH separa fisicamente due segmenti di rete
    - Compromettere il PF non da più accesso alla rete interna
    - Svantaggio: tutto il traffico dai clienti deve fluire attraverso il BH, anche quello innocuo
- **Screened subnet**
    - Uso di due PF router, uno a monte del BH, l'altro a valle
    - Rafforzamento della separazione tra esterno e interno
    - Nasconde l'esistenza di Internet alla rete privata, ma consente ai router di inoltrare il traffico "banale" senza passare dal BH
    - Nasconde completamente all'esterno l'esistenza della subnet privata, ostacolando l'enumerazione da parte degli attaccanti

IPTables: PF integrato in Linux, userspace per utilizzare NETFILTERS, modulo di Linux che espone degli hook a cui si possono associare degli handler. Gli hook sono:

- NF_IP_PRE_ROUTING
- NF_IP_ LOCAL_IN
- NF_IP_FORWARD
- NF_IP_LOCAL_OUT
- NF_IP_POST_ROUTING

Per poter interagire con questi hook deve essere creato un modulo kernel. IPTables possiede diverse tabelle (contenenti regole del tipo "se <condizione> allora <azione>"):

- **raw**: permette di bypassare conntrack così disattivando il tracciamento della connessione dei pacchetti
- **conntrack**: modulo che gestisce lo stato della connessione. Mantiene una lista di tutte le connessioni e del loro stato
- **mangle**: modica l'intestazione IP del pacchetto
- **nat**: traduce gli indirizzi di rete
- **filter**: tabella principale, decide se il pacchetto può continuare la sua marcia o se bloccarlo
- **security**: imposta contrassegni di contesto di sicurezza SELinux interni sui pacchetti

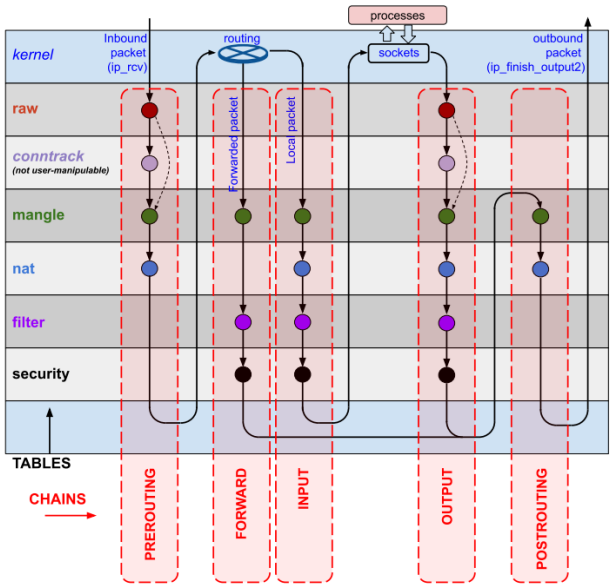
Target: sono le azioni da compiere quando avviene il match (elenco di condizioni prestabilito). Due tipi di target:

- **Terminating**
  - ACCEPT: indica a netfilter di proseguire l'analisi con le catene successive
  - DROP: indica a netfilter di scartare il pacchetto
  - RETURN: applica la policy di default della catena
  - DNAT, REDIRECT: cambia la destinazione, cambia la destinazione con la macchina locale
  - SNAT, MASQUERADE: cambia la sorgente, cambia la sorgente usando l'IP della macchina locale
- **Non terminating**
  - NO TARGET
  - LOG: molto lento e può diventare un collo di bottiglia

Monitoraggio

Le varie fasi di un attacco possono lasciare tracce, è importante individuare e rilevare tali tracce per limitare o evitare danni. Vari termini del monitoraggio:

- **IDS** (Intrusion Detection System): sistema in grado di rilevare tentativi di attacco. Utilizza due meccanismi:
  - **Signature based**: hanno un DB delle impronte degli attacchi
  - **Anomaly Detection**: riconosce ogni deviazione rispetto all'utilizzo standard
- **IPS** (Intrusion Prevention System): IDS in grado di interagire con sistemi di controllo dell'accesso per bloccare il traffico malevolo
- **SIEM** (Security Information and Event Management): sistema per gestire l'analisi e lo storage dei log e il monitoraggio real-time



Quando si rilevano gli eventi, possiamo avere:

- **Falsi positivi**: segnalazione attacco errata
- **Falsi negativi**: attacco reale che non genera segnalazione

Due principali strategie di rilevazione, di cui il primo è basata sulla rete, le altre due sono basate sull'endpoint:

- **NIDS** (Network-based IDS)
  - Vantaggi

        - Usa dati intercettati sui canali di comunicazione
        - Offre visibilità su tutto il traffico e richiede un singolo punto di installazione
        - Un malfunzionamento non incide sugli endpoint
    - Svantaggi
        - Maggiore esposizione ai falsi positivi dato che processi legittimi possono occasionalmente generare traffico anomalo
        - Molto soggetto a sovraccarico o evasione
        - Non può esaminare il traffico cifrato
- **HIDS** (Host-based IDS)
    - Vantaggi
        - Minor tasso di falsi positivi
        - Economico: sfrutta sistemi già esistenti
    - Svantaggi
        - Ruba potenza computazionale e di storage alla macchina
        - Se la macchina è compromessa può essere neutralizzato
- **EDR** (Endpoint Detection and Response)
    - HIDS fortemente integrato con il SO
    - Vantaggi
        - Raccoglie eventi dai device driver di filesystem e di rete e comunicazioni tra processi
        - Maggiori possibilità di risposta
    - Svantaggi
        - Richiede uno stretto interfacciamento con il SO
        - Difficile trovare soluzioni open e non troppo costose

Quando si fa monitoraggio tramite host, un passo fondamentale è quello dell'**Integrity check**, ovvero memorizzare una copia del sistema (del filesystem) in un database, per poter ogni tot fare un confronto con il sistema su cui si stanno facendo le operazioni. Tra i più diffusi integrity check abbiamo AIDE e AFICK.

I log di sistema sono indispensabili per la diagnostica e per rilevare attività malevole o sospetta da processi utente o kernel. ln Linux i log vengono fatti da demoni di diverso genere (tutti nati dalla famiglia di syslog), anche se tutte le distribuzioni Linux stanno piano piano iniziando ad utilizzare systemd (attivato dal boot) avendo come differenza che syslog viene avviato da systemd, pertanto spostare il logging a systemd ci permette di controllare il sistema prima.
Dalla versione 4.18 di Linux è supportato il tracciamento di ogni evento legato alle system call, il kernel invia messaggi a un demone user-space (auditd) secondo regole di configurazione.
Per l'analisi del log sono presenti diversi software: *Logwatch*, *Swatch*, *Graylog2*...

I sistemi **SIEM** permettono:

- **Aggregazione dei dati**: raccoglie log/eventi da più fonti, normalizza e consolida i dati
- **Correlazione**: collega eventi con attributi comuni in pacchetti significativi e genera automaticamente avvisi in base a condizioni specifiche
- **Monitoraggio**: grafici per visualizzare lo stato corrente e grafici relativi alla conformità
- **Ritenzione**: conservazione a lungo termine e analisi forense

Il nonno di NIDS è **SNORT**. Esso si basa su un sistema di packet sniffing (come wireshark o tcpdump) e un suo erede è SURICATA.

Autorizzazione: modelli e implementazioni

Un soggetto autenticato deve essere autorizzato a svolgere operazioni sulle risorse del sistema. È importante definire tre passaggi: definire il modello del sistema controllato, definire la politica di accesso, attuare la politica.

I modelli di controllo dell'accesso sono:

- **DAC** (Discretionary Access Control): ogni oggetto ha un proprietario che è lo stesso che decide i permessi
- **MAC** (Mandatory Access Control): permessi scelti in un DB centralizzato amministrato da un security manager
- **RBAC** (Role-Based Access Control): i permessi sono assegnati a ruoli. Questo è un modello ritenuto complesso

Dunque il controllo dell'accesso è decidere se un soggetto può eseguire una specifica operazione su di un oggetto. Grandi dimensioni della matrice che comportano una grande quantità di memoria e di overhead introdotto dal lookup nella matrice. È possibile partizionare la tabella:

- Per soggetto: **Capability List**
- Per oggetto: **Access Control List**
- Esiste anche la **Authorization Table** con entry di valori soggetto:regola:permesso

Sicurezza fisica e collocazione delle risorse

Abbiamo visto sistemi di controllo dell'accesso alle informazioni, ora è importante vedere come influisce la collocazione delle risorse (permessi sui file, chiavi crittografiche...). Due tipi di collocazione:

- **On premises**
  - Vantaggi: nessuna condivisione e relativa semplicità di separazione tra segreti e sistemi e dati
  - Svantaggi: deve essere garantito il corretto funzionamento dell'HW e l'esecuzione dei SO e software integri e autentici
- **In the cloud**
  - Vantaggi: gestione HW/SW di altissimo livello
  - Svantaggi: ambienti di memorizzazione ed elaborazione condivisi e deallocazione

È importante proteggersi anche da attacchi non informatici, come *Insider*, *Tailgating* o *Social Engineering*, così come serve proteggersi contro attentati, problemi elettrici o problemi al sistema di raffreddamento.

Ci sono dispositivi che permettono attacchi HW difficilmente rilevabili:

- BadUSB
- Thunderspy
- Keylogging e videoghosting
- Key injection
- Disk un/plugging
- Power glitching

In caso di collocazione dei dati in remoto:

- Non è possibile monitorare l'accesso al sito (bisogna fidarsi)
- Possibili tecniche di mitigazione:
    - Case chiuso a chiave e fissato al rack
    - Dispositivi di rivelazione delle intrusioni
    - Misure di protezione dei dati che rendano inutile il furto
    - Disabilitare le periferiche non utilizzate

Attacchi fisici alle risorse logiche:

- Il computer al boot carica ciò che trova
- Questo può essere compromesso in ogni step, dalla prima istruzione alla partenza del SO
- I vari step del boot possono essere divisi in:
    - **BIOS**
        - Possibile inserire password per evitare modifiche
        - Spesso le password sono facili da indovinare, non bisogna mai affidarsi ad un unico strato di protezione
    - **Boot Loader**
        - Possibile inserire password per modificare parametri o immagine del boot
        - **Measured Boot**: il TPM (Trusted Platform Module) usa i PCR (Platform Configuration Registers) per salvarsi e rendere disponibile all'utente la "firma" della procedura di avvio
        - **Trusted Boot**: *Measured Boot* in cui esiste un sistema per verificare che la firma del boot sia quella corretta
        - **Secure Boot**: implementazione UEFI di *Trusted Boot*, è un sistema a certificati, tutti i binari che vengono caricati devono essere stati firmati da un'autorità
    - **OS**
    - **init** (pid 1)