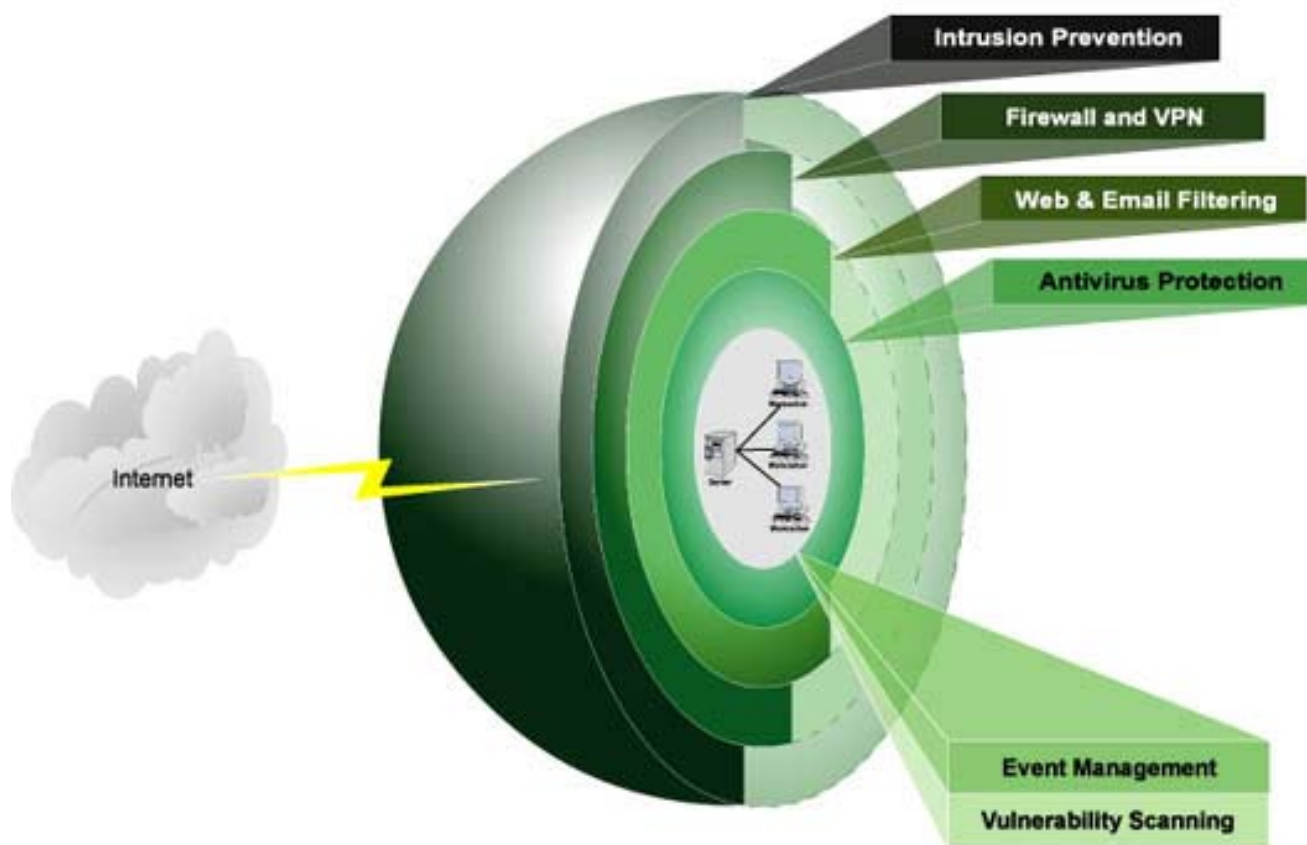


---

# ***Sicurezza nelle Reti***

## ***Appunti***



*Autore: Fabio Angelini*

---

Questo documento è un riassunto delle lezioni del corso “Sicurezza nelle Reti”  
del prof. Marco Cremonini, anno accademico 2008-2009.  
L'autore non si assume nessuna responsabilità sulla veridicità e validità  
di quanto scritto e consente la distribuzione gratuita totale o parziale del contenuto.

*Fabio Angelini*  
[fabioang@iol.it](mailto:fabioang@iol.it)

---

<b>1. Problemi di sicurezza nelle reti .....</b>	<b>5</b>
<b>1.1. ARP Poisoning.....</b>	<b>5</b>
Funzionamento.....	5
<b>1.2. Frammentazione .....</b>	<b>6</b>
1.2.1. Firewall Static Packet Filter.....	6
1.2.2. Firewall Stateful.....	6
1.2.3. Attacchi basati sulla frammentazione .....	7
DoS Stateless Attack .....	7
Tiny Fragment Attack .....	7
Overlap Attack .....	7
Teardrop Attack .....	7
<b>1.3. Tecniche di scan .....</b>	<b>8</b>
1.3.1. Specifiche TCP: RFC 793 .....	8
1.3.2. TCP scan .....	8
1.3.3. SYN scan .....	8
1.3.4. TCP FIN, Xmas, Null scan .....	9
1.3.5. ACK scan.....	9
1.3.6. SYN scan + ACK scan.....	9
1.3.7. UDP scan .....	10
1.3.8. Idle scan.....	10
1.3.9. OS fingerprint .....	10
<b>1.4. IP spoofing .....</b>	<b>11</b>
<b>1.5. TCP session hijacking.....</b>	<b>11</b>
<b>1.6. Vulnerabilità.....</b>	<b>11</b>
Punti di attenzione.....	11
Categorizzazione.....	12
Categorie.....	12
Ciclo di vita di una vulnerabilità .....	12
<b>1.7. Trojan Horse .....</b>	<b>13</b>
1.7.1. Evoluzione dei Trojan Horse .....	13
<b>1.8. Internet Worm .....</b>	<b>14</b>
<b>2. Tecnologia per la sicurezza nelle reti.....</b>	<b>15</b>
<b>2.1. Firewall .....</b>	<b>15</b>
<b>2.2. Firewall Static Packet Filter .....</b>	<b>15</b>
2.2.1. Sintassi Cisco .....	16
Formato ACL Standard .....	16
Formato ACL Extended .....	16
2.2.2. Politiche di Static Packet Filter .....	17
2.2.3. Esempi di Static Packet Filter .....	18
FTP Attivo (default).....	18
FTP Passivo .....	18
RPC .....	19
Firewall e Least Privilege.....	19
<b>2.3. Stateful Firewall.....</b>	<b>20</b>
Stati di una Sessione TCP e Stateful Filtering .....	20
Politiche di Stateful Firewall.....	20
<b>2.4. Proxy .....</b>	<b>21</b>
2.4.1. Reverse Proxy .....	21
2.4.2. Proxy Firewall.....	21
2.4.3. FTP Bounce Attack.....	22
2.4.4. Articolo “A Quantitative Study of Firewall Configuration Errors” .....	22
<b>2.5. Network Address Translation (NAT).....</b>	<b>22</b>
<b>2.6. Disegno di una architettura per la sicurezza .....</b>	<b>23</b>
2.6.1. Modelli Architeturali .....	23
Architettura con Firewall in Serie .....	23

---

Architettura con Firewall in Parallelo .....	23
<b>2.7. Sdoppiamento di Servizi.....</b>	<b>24</b>
2.7.1. Mail server e Mail relay .....	24
2.7.2. DNS Server Interno ed Esterno.....	24
2.7.3. DNS Attacks .....	24
DNS Cache Poisoning.....	25
DNS ID Spoofing.....	26
<b>2.8. netfilter/iptables .....</b>	<b>27</b>
2.8.1. Filtraggio con iptables.....	28
Considerazioni su FTP canali dati.....	29
<b>3. Sistemi IDS.....</b>	<b>30</b>
3.1.1. Anomaly Detection : caratteristiche.....	30
3.1.2. Anomaly Detection : tecniche.....	30
Profili d'uso .....	30
Algoritmi di Hashing (solo HIDS).....	30
Protocol Anomaly Detection (solo NIDS) .....	30
3.1.3. Misuse Detection : caratteristiche .....	31
3.1.4. Misuse Detection : tecniche .....	31
Signature Detection.....	31
3.1.5. Falsi positivi e Falsi negativi .....	31
3.1.6. Risposte Automatiche .....	31
<b>3.2. Tecniche di Evasione .....</b>	<b>32</b>
3.2.1. Attacco basato sul timeout .....	32
Caso 1: Timeout di riassettaggio dell'IDS minore di quello della vittima .....	32
Caso 2: Timeout di riassettaggio dell'IDS maggiore di quello della vittima .....	33
3.2.2. Attacco basato sul TTL.....	33
3.2.3. Attacco basato sulla sovrapposizione (overlapping).....	34
<b>3.3. Snort.....</b>	<b>35</b>
3.3.1. Struttura delle firme di Snort .....	35
reference, classtype, sid, rev .....	35
flow .....	35
flags.....	36
content.....	36
nocase.....	36
depth.....	36
offset .....	37
pcre.....	37
fragbits .....	37
id .....	37

# 1. Problemi di sicurezza nelle reti

## 1.1. ARP Poisoning

L'**ARP poisoning** è una tecnica che consente a un'attaccante di manipolare l'associazione tra MAC address e indirizzo IP. Questa tecnica, di tipo Man-in-the-Middle, consiste nell'inviare intenzionalmente e in modo forzato *risposte ARP* contenenti dati inesatti o, meglio, non corrispondenti a quelli reali. In questo modo la tabella ARP (*ARP entry cache*) di un host conterrà dati alterati (da qui i termini *poisoning*, letteralmente, "avvelenamento" e *spoofing*, "raggiro").

**Criticità:** Come tutti gli attacchi man-in-the-middle, è una tecnica difficile da individuare poiché non ci sono cambiamenti visibili ai due partecipanti della comunicazione.

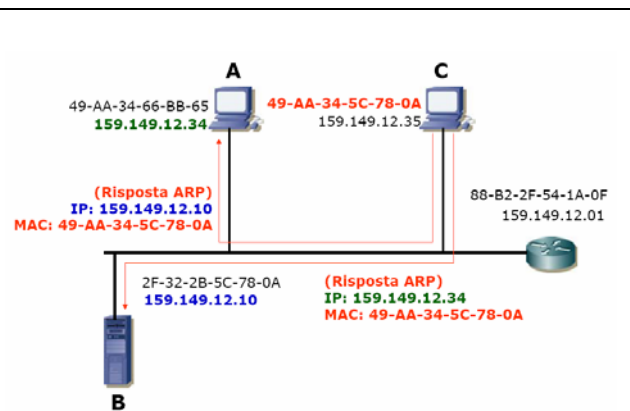
**Limiti:** L'attaccante deve trovarsi sullo stesso segmento di rete.

**Contromisure:** Utilizzare tabelle ARP statiche con conseguente riduzione della flessibilità e semplicità di manutenzione.

### Funzionamento

1. A e B vogliono comunicare
2. A richiede il MAC dell'indirizzo IP di B
3. B richiede il MAC dell'indirizzo IP di A
4. C risponde ad A e B in modo tale che gli indirizzi IP richiesti corrispondano al MAC di C
5. A trasmette un pacchetto con IP destinazione di B ma con MAC di C
6. C riceve il pacchetto e lo trasmette a B con IP sorgente di A
7. B riceve il pacchetto e trasmette un pacchetto di risposta con IP destinazione di A ma con MAC di C

A e B *sembrano* comunicare normalmente, in realtà A comunica con C e C con B.

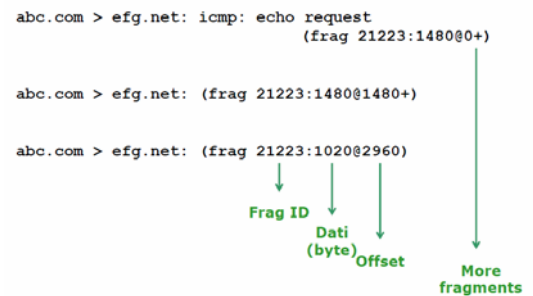


## 1.2. Frammentazione

Un datagramma IP, quando viene passato al livello fisico, viene incapsulato nel payload Ethernet per formare un frame fisico. Il problema è che le dimensioni massime di questo frame sono limitate. Il valore di questo limite è chiamato MTU (maximum transfer unit). Se il datagramma IP è troppo grande per stare in una trama lunga al massimo MTU byte allora viene eseguita una *frammentazione*, cioè il datagramma viene suddiviso in tanti frammenti abbastanza piccoli da entrare nel frame della rete. Ogni frammento possiede un proprio header molto simile all'header IP del datagramma originale, ma con i campi *Identification*, *Flags*, *Fragment Offset* e *Total Length* settati opportunamente. Il frammento iniziale è l'*unico* frammento ad avere l'header di trasporto TCP, UDP o ICMP (ricordiamo che il payload IP frammentato è il pacchetto TCP, UDP o ICMP).

Nell'header IP sono importanti i seguenti campi:

- **Identification**: valore identificativo, uguale per tutti i frammenti di un pacchetto originario che dovrà essere riassemblato a destinazione
- **Flags**: un bit **MF** per indicare *more fragments*, settato a **1** per tutti i frammenti tranne l'ultimo che ha il flag a **0**. Un secondo bit **DF** per indicare *don't fragment*, cioè per istruire i router che, se fosse necessaria la frammentazione, il datagramma va scartato e non frammentato
- **Offset**: rappresenta la posizione relativa dei dati nei diversi frammenti rispetto la posizione dei dati nel datagramma originario.
- **Total Length**: indica la dimensione (in byte) dell'intero datagramma, comprendendo header e dati.



La frammentazione è un problema sia per i firewall *stateless* sia *stateful*.

### 1.2.1. Firewall Static Packet Filter

Un firewall *static packet filter* (o *stateless*) esegue il controllo del traffico basandosi unicamente sulle informazioni contenute negli *header* dei singoli pacchetti TCP. Questi firewall erano originariamente configurati per applicare la politica *solo* sul primo frammento e lasciare passare traffico frammentato privo di header. La soluzione era considerata “sicura” perché i frammenti privi del primo frammento non potevano essere riassemblati dal sistema destinatario che li avrebbe scartati. Questo tipo di configurazione poteva però dare luogo ad attacchi di tipo DoS sul sistema destinatario (es. Dos Stateless Attack) oppure consentire di “scavalcare” le politiche di sicurezza ACL (es. Overlap Attack).

### 1.2.2. Firewall Stateful

Un *stateful firewall* esegue il controllo del traffico basandosi sulle informazioni contenute negli *header* dei singoli pacchetti e sullo stato delle connessioni attive. Questi firewall riassemblano i frammenti per ricostruire il pacchetto e quindi mantenere traccia delle connessioni attive.

Il processo di riassettaggio dei firewall potrebbe essere sfruttato per portare a termine attacchi di tipo DoS (es. TearDrop Attack).

### 1.2.3. Attacchi basati sulla frammentazione

#### DoS Stateless Attack

Se il firewall rifiuta il primo frammento che contiene l'header TCP (su cui può applicare una politica di sicurezza) e lasciare passare gli altri frammenti che non hanno header TCP, un host che riassembla i frammenti, posto in cascata dopo il firewall potrebbe subire un attacco di tipo DoS (Denial-of-Service). L'host, in uno scenario del genere, riceve molti frammenti con stesso FRAG\_ID che impediscono il time-out ma degradano le prestazioni stesse dell'host.

#### Tiny Fragment Attack

Se la dimensione del frammento è abbastanza piccola da obbligare la suddivisione dell'header TCP in due frammenti, le regole di filtraggio del firewall potrebbero essere superate perché applicabili solo al primo frammento. Il pacchetto a questo punto riuscirebbe a oltrepassare il firewall.

#### Overlap Attack

Questo tipo di attacco prevede che il secondo frammento del pacchetto contenga un offset errato. Più precisamente l'offset è impostato in modo tale da sovrascrivere il numero di porta destinazione presente nel primo pacchetto (quello con l'header TCP). L'attacco ha dunque lo scopo di "scavalcare" le politiche di sicurezza del firewall.

#### Teardrop Attack

Questo tipo di attacco prevede l'inserimento di false informazioni nell'offset nei frammenti. Di conseguenza, durante il riassettaggio, ci sono vuoti o sovrapposizioni di frammenti che possono rendere il sistema instabile.

### 1.3. Tecniche di scan

Una informazione necessaria a preparare molte intrusioni consiste nel conoscere lo *stato delle porte*:

- **Accepted** o **Open**, l'host destinatario genera una risposta che indica che un servizio è in ascolto su quella porta.
- **Denied** o **Closed**, l'accesso all'host è bloccato oppure non c'è un servizio in ascolto su tale porta.

Lo scenario è quello di un client (*scanner*) che cerca di scoprire lo stato delle porte di un server (*target*).

#### 1.3.1. Specifiche TCP: RFC 793

Ordine di valutazione ↓	1. Un segmento in arrivo con il flag RST viene sempre <i>scartato senza alcuna risposta</i>
	2. Se una porta è nello stato di <b>closed</b>
	a. Se il segmento in arrivo <i>non</i> ha il flag RST attivo (vedi punto 1), allora viene inviato come risposta al mittente un pacchetto con il flag RST
	3. Se una porta è nello stato di <b>listen</b>
	a. Se il segmento in arrivo contiene un ACK allora viene risposto un RST
	b. Se il segmento in arrivo contiene un SYN allora viene inviato come risposta al mittente un pacchetto con i flag SYN e ACK
	c. Se nessuno dei precedenti è vero allora il segmento viene <i>scartato senza risposta</i>

#### 1.3.2. TCP scan

Nel **TCP scan** lo scanner cerca di aprire una normale connessione TCP attraverso la chiamata di sistema `connect()` e di analizzarne il risultato.

- se la porta è *open*, il target risponde con un SYN,ACK e viene portato a termine il three-way handshake
- se la porta è *closed*, il target risponde con un RST

Questo tipo di scan è facilmente identificabile, infatti, i *log* del target mostreranno queste connessioni completate.

Porta aperta

Scanner	Target
SYN →	
	← SYN, ACK
ACK →	
FIN →	
	← ACK
	← FIN
ACK →	

Porta chiusa

Scanner	Target
SYN →	
	← RST

#### 1.3.3. SYN scan

Nel **SYN scan** lo scanner genera un pacchetto SYN:

- se la porta è *open*, il target risponde con un SYN,ACK e il client termina la connessione con un RST
- se la porta è *closed*, il target risponde con un RST

La differenza principale rispetto al TCP scan è che non viene terminato correttamente il three-way handshake e dunque non tutti i target sono in grado di tracciare tale sequenza.

Porta aperta

Scanner	Target
SYN →	
	← SYN, ACK
RST →	

Porta chiusa

Scanner	Target
SYN →	
	← RST



### 1.3.4. TCP FIN, Xmas, Null scan

Nel **FIN scan** lo scanner pacchetti isolati con il flag **FIN** a 1

Nel **Xmas scan** lo scanner pacchetti isolati con i flag **FIN**, **URG** e **PSH** a 1

Nel **Null scan** lo scanner pacchetti isolati con **tutti i flag** a 0

- se la porta è *open* i pacchetti vengono scartati senza alcuna risposta
- se la porta è *closed* il server risponde con un RST

Porta aperta

Scanner	Target
FIN, URG, PSH →	
	✕ Nessuna risposta

Porta chiusa

Scanner	Target
FIN, URG, PSH →	
	← RST

### 1.3.5. ACK scan

Nel **ACK scan** lo scanner genera pacchetti isolati con il flag **ACK** a 1

- se la porta è *raggiungibile*, il target risponde con un RST
- se la porta è *non raggiungibile*, i pacchetti vengono scartati senza alcuna risposta

Porta raggiungibile

Scanner	Target
ACK →	
	← RST

Porta non raggiungibile

Scanner	Target
ACK →	
	✕ Nessuna risposta

Questa tecnica ha lo stesso scopo del comando PING, cioè stabilire se un target è raggiungibile o meno. E' utilizzata al posto del comando PING perchè spesso i firewall sono configurati per bloccare il protocollo ICMP (che è utilizzato dal comando PING)

### 1.3.6. SYN scan + ACK scan

Se usati insieme **SYN scan** e l'**ACK scan** si può ottenere:

```
scan.net.34567 > server.com.22: S 465834:465834(0)
fw.server.com > scan.net: icmp: host target.host unreachable - admin prohibited
(il SYN viene rifiutato per qualche politica di filtraggio lato server)
scan.net.34567 > server.com.22 : ack 1379777
server.com.22 > scan.net.34567 : R
(l'ACK ritorna come risposta RST, cioè server raggiungibile)
```

Se il server è stato raggiunto significa che il meccanismo di filtraggio del server non è in grado di distinguere tra un pacchetto di ACK isolato e un pacchetto di ACK inserito all'interno di una sessione TCP. Da qui si deduce che il firewall esegue solo un filtraggio a livello di singolo pacchetto e *non mantiene lo stato della sessione*.

### 1.3.7. UDP scan

Nel **UDP scan** lo scanner genera pacchetti UDP con 0 byte di dati (pacchetti vuoti)

- se la porta è *aperta*, il target non risponde
- se la porta è *chiusa*, viene restituito un messaggio *ICMP port unreachable*

### 1.3.8. Idle scan

Nell'**IDLE scan** lo scanner non invia direttamente pacchetti alla vittima ma utilizza come intermediario un host attivo e raggiungibile, detto “zombi”. Questa tecnica basa la sua implementazione su indirizzi spoofed e sul campo ID dell’header IP. E’ importante ricordare che il campo ID è uguale per ogni frammento appartenente allo stesso pacchetto mentre è solitamente incrementato di uno per ogni diverso pacchetto.

<b>Idle scan di una porta aperta</b>		
<b>Step 1</b> Lo scanner invia un SYN-ACK allo zombie. Lo zombie, che non si aspetta il SYN-ACK, risponde con un RST e un certo ID	<b>Step 2</b> Lo scanner invia alla vittima un SYN con indirizzo sorgente dello zombie. La vittima risponde allo zombie con un SYN-ACK (porta aperta). Lo zombie, che non si aspetta il SYN-ACK, risponde con un RST e l’ID incrementato di uno	<b>Step 3</b> Lo scanner invia un SYN-ACK allo zombie. Lo zombie, che non si aspetta il SYN-ACK, risponde con un RST e l’ID incrementato di <i>due</i> rispetto allo <i>Step 1</i> , quindi la porta è aperta
<b>Idle scan di una porta chiusa</b>		
<b>Step 1</b> Lo scanner invia un SYN-ACK allo zombie. Lo zombie, che non si aspetta il SYN-ACK, risponde con un RST e un certo ID	<b>Step 2</b> Lo scanner invia alla vittima un SYN con indirizzo sorgente dello zombie. La vittima risponde allo zombie con un RST (porta chiusa). La vittima, che non si aspetta il RST, non risponde lasciando l’ID invariato	<b>Step 3</b> Lo scanner invia un SYN-ACK allo zombie. Lo zombie, che non si aspetta il SYN-ACK, risponde con un RST e l’ID incrementato di <i>uno</i> rispetto allo <i>Step</i> , quindi la porta è chiusa

Osservazione: La risposta a un ACK isolato (non di sessione) è sempre un RST indipendentemente dallo stato della porta (RFC 793).

### 1.3.9. OS fingerprint

E’ il processo per determinare il sistema operativo usato dal target.

Tool come **nmap** e molti altri, utilizzano

- sequenze di *pacchetti anomali*, non rispondenti alle specifiche del protocollo TCP
- pacchetti/sequenze note per provocare reazioni dipendenti dalle *singole implementazioni* dello stack TCP

In questo modo, analizzando le risposte, riescono a capire, con discreta approssimazione, il sistema operativo e talvolta la sua specifica versione.

## 1.4. IP spoofing

Lo **spoofing** è una qualsiasi tecnica che consente a un client di presentarsi con l'identità altrui.

Nell'**IP spoofing** un client utilizza l'indirizzo IP di un'altro host. E' importante sottolineare che lo spoofing ha successo *se il client (intrusore) riesce a vedere le risposte del server*, come ad esempio quando è nella stessa sottorete (le risposte verranno comunque inoltrate all'host corretto). Un client che si trova su una diversa sottorete potrebbe attivare l'opzione di *Source Routing* per ricevere le risposte.

Il **Source Routing** consente a un mittente di specificare quale routing dovrà seguire un pacchetto.

Due sono le tipologie di *Source Routing*

- **Loose Source Routing**: Vengono specificati alcuni indirizzi IP attraverso i quali il pacchetto dovrà passare e permesso il routing anche su altri indirizzi oltre a quelli specificati.
- **Strict Source Routing**: Il pacchetto dovrà attraversare solo gli indirizzi IP specificati.

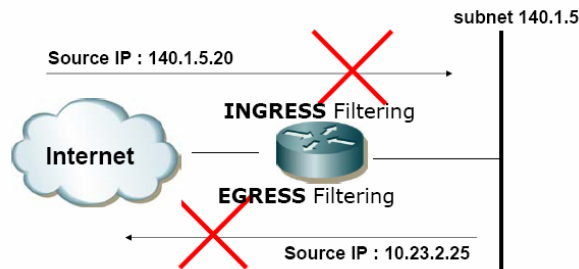
Per combattere l'IP spoofing è possibile configurare i seguenti filtri sul router.

**INGRESS Filtering** (filtraggio del traffico in ingresso)

- Vengono bloccati i pacchetti in ingresso con IP address sorgente *appartenente* alla rete interna.

**EGRESS Filtering** (filtraggio del traffico in uscita)

- Vengono bloccati i pacchetti in uscita con IP address sorgente *non appartenente* alla rete interna.



## 1.5. TCP session hijacking

Nel **TCP Session Hijacking** una sessione attiva tra un client e un server viene “dirottata” da un intrusore che:

- Impersona un client legittimo
- Prosegue con il server la sessione apparendo il client legittimo

Spesso si accompagna con la necessità di rendere il client legittimo inattivo.

Lo scopo di questa tecnica è quello di bypassare la fase di autenticazione o impersonare l'identità altrui e sfruttare gli stessi privilegi o accedere ad informazioni riservate.

Le contromisure sono:

- Comunicazioni crittate (es. SSL)
- Protocolli sicuri (es. SSH)
- Limitare i tipi di connessione (FIREWALL)
- Limitare la possibilità di accessi remoti
- Metodi di autenticazione forti

## 1.6. Vulnerabilità

Il termine **vulnerabilità** indica un comportamento non previsto del sistema che un attaccante può sfruttare per violare le misure di sicurezza e compiere azioni indesiderate e/o pericolose.

### Punti di attenzione

- Nessuna delle vulnerabilità è causata esclusivamente da problemi delle tecnologie di rete protocolli: TCP/IP, numeri di sequenza, ecc...
- Tutte le vulnerabilità si riferiscono a problemi dei singoli sistemi: gestione della memoria (vari casi di *overflow*), interfacciamento con database (*SQL injection*), gestione degli input (*directory traversal*), shell e esecuzione remota di comandi

## Categorizzazione

Avere una categorizzazione delle vulnerabilità è importante per poter fissare delle priorità nelle *contromisure* da adottare.

Elementi per classificare una vulnerabilità	Pericolosità alta	Pericolosità bassa
<i>Diffusione</i>	sistema molto diffuso	sistema poco diffuso
<i>Tipo di sistema</i>	server	client
<i>Tipo di privilegio</i>	root	normal user
<i>Tipo di configurazione</i>	standard	custom
<i>Criticità/valore</i>	componente critico	componente non critico
<i>Impatto sull'infrastruttura di rete</i>	perdita di connettività	nessuna perdita di connettività
<i>Grado di difficoltà</i>	facile da sfruttare	difficile da sfruttare
<i>Probabilità di attacchi</i> basati sulla vulnerabilità	tool automatici	script manuali

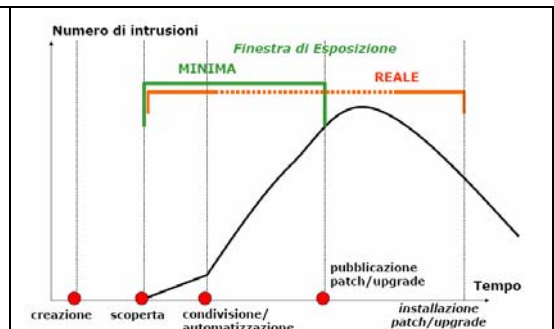
## Categorie

Tipo	Descrizione	Contromisure
Critica	Facile da sfruttare e/o impatto grave	Occorre adottare <i>immediate contromisure</i> , anche a scapito delle funzionalità offerte dall'organizzazione
Alta	Meno facili da sfruttare con impatto da valutare	Se non si adottano immediate contromisure, occorre comunque un <i>monitoraggio costante</i> per verificarne l'evoluzione, nonché un piano d'intervento pronto per essere applicato
Moderata	Complesse da sfruttare e/o impatto non grave	<i>Analisi costi/benefici</i> dei possibili interventi. Esisteranno diverse contromisure, occorre valutare quale sia la più conveniente rispetto al rischio che si stima l'organizzazione corra <sup>i</sup>
Bassa	Molto difficili da sfruttare e/o di impatto ridotto	<i>Non ignorarle a priori</i> perché analizzando il contesto specifico potrebbero rivelarsi di categoria superiore

## Ciclo di vita di una vulnerabilità

- **Creazione:** Un errore viene introdotto nel codice nel corso dello sviluppo di un sistema, un servizio, una applicazione.
- **Scoperta:** Qualcuno scopre l'errore presente nel codice e intuisce che questo ha conseguenze sulla sicurezza. Solo ora si parla di vulnerabilità anziché di errore nel codice.
- **Condivisione/Automatizzazione:** La conoscenza di tale vulnerabilità viene fatta prima circolare in ambito ristretto poi si diffonde grazie anche allo sviluppo di tool automatici che ne fanno uso.
- **Pubblicazione Patch/Upgrade:** Il produttore del sistema corregge l'errore emettendo una patch o una nuova versione del codice. La presenza di tale vulnerabilità, insieme alla presenza della patch disponibile viene resa pubblica.

La **finestra temporale di esposizione** di un sistema è il tempo nel quale un sistema può risultare vittima di attacchi informatici a causa di una vulnerabilità. Il tempo *minimo* è quello compreso tra la scoperta della vulnerabilità e la pubblicazione della patch, il tempo *reale* è superiore perché non sempre la patch viene applicata tempestivamente.



Le patch rappresentano una soluzione spesso *inefficace*

- scarsa consapevolezza di molti sistemisti
- frequenza di emissione troppo elevata
- spesso causa di malfunzionamenti o nuovi problemi

<sup>i</sup> La vulnerabilità DoS è considerata moderata perché non è un'intrusione

## 1.7. Trojan Horse



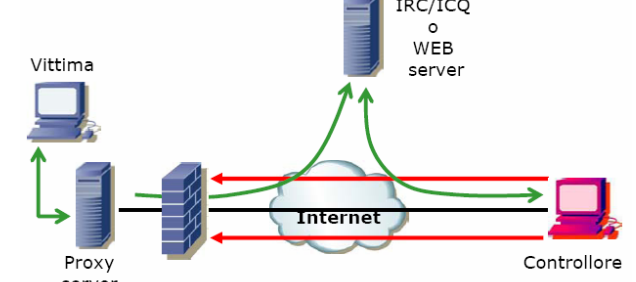
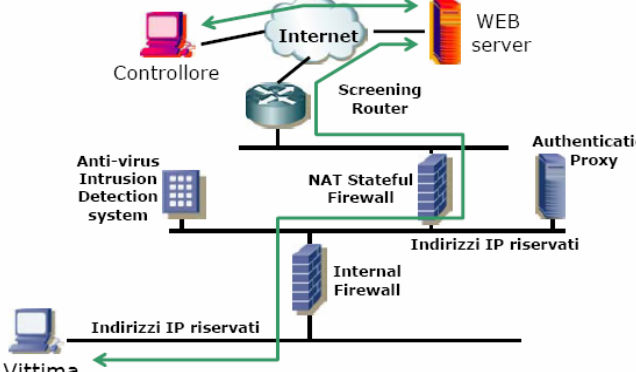
Un *trojan horse* è un programma apparentemente legittimo ma che nasconde funzionalità non dichiarate.

Un trojan horse spesso nasconde una funzionalità detta *backdoor*, cioè un accesso non controllato ad un computer attivato in maniera non autorizzato.

- Esempio tipico è l'attivazione servizi di rete su porte non convenzionali per consentire l'accesso remoto e il facile controllo di una macchina.
- Esempi reali sono l'abilitazione di server SSH, suite complete per il controllo remoto (Back Orifice, SubSeven).

### 1.7.1. Evoluzione dei Trojan Horse

Le contromisure adottate per difendersi da questi attacchi hanno fatto evolvere i *meccanismi di comunicazione* tra i malware e il software di controllo/gestione.

<p><b>Primo Modello:</b> indirizzi IP pubblici, nessuna protezione perimetrale.</p> <p>Il controllore apre connessioni verso il malware</p>	
<p><b>Secondo Modello:</b> indirizzi IP pubblici, protezione rispetto alle connessioni entranti.</p> <p>Il malware apre connessioni verso il controllore</p>	
<p><b>Terzo Modello:</b> indirizzi IP riservati, filtraggio delle connessioni input/output, autorizzate le connessioni solo verso indirizzi fidati.</p> <p>Il malware comunica con il controllore sfruttando un servizio esterno non filtrato (esempio IRC) e utilizzando il protocollo applicativo dello stesso servizio.</p>	
<p><b>Quarto Modello:</b> indirizzi IP riservati, filtraggio delle connessioni input/output, meccanismi e infrastruttura di sicurezza evoluta.</p> <p>Il malware comunica con il controllore utilizzando il protocollo HTTP che non è filtrato perchè spesso utilizzato per motivi lavorativi.</p>	

Osservazione: la differenza principale tra terzo e quarto modello è data dalla limitazione più stringente imposta dal quarto livello, nel quale è consentita la connessione in uscita solo attraverso HTTP.

## 1.8. Internet Worm

Un *internet worm* è un software che automatizza l'intero processo di una intrusione.

Possiede generalmente le seguenti caratteristiche:

- capacità di *scanning*
  - Solitamente mirato a ricercare host con servizi corrispondenti alle vulnerabilità per le quali possiedono gli script che ne automatizzano l'attacco
  - Esempi tipici: telnet, ftp, ssh, netbios, web server, database server
- capacità di *sniffing*
  - Intercettano traffico dalla rete alla ricerca di informazioni (es. username/password)
  - Registrano gli input della tastiera (keystroke logger) e successivamente inviano quanto registrato al controllore
- tecniche di *intrusione automatizzate (exploit)*
  - Vengono implementati script che permettono, una volta identificati nuovi target, di compromettere un sistema vulnerabile e propagarsi
  - Alcuni casi di worm possiedono più tecniche di intrusione avendo implementati exploit per diverse vulnerabilità. Questo ne aumenta il potenziale di diffusione
- capacità di *comunicazione e propagazione* attraverso Internet o reti locali
  - Mutuano dai Trojan le diverse tecniche di comunicazione e di controllo remoto.

Il numero di intrusioni effettuate da questi malware ha raggiunto un numero spropositato: negli ultimi anni la loro *propagazione* in tempi rapidissimi (*Fast Worm*) ha completamente intasato il traffico mondiale, presentandosi come la maggior minaccia per la sicurezza delle infrastrutture aziendali.

Un tasso di propagazione così alto rende *inutili le contromisure manuali*, di conseguenza le contromisure devono essere automatiche o contromisure di prevenzione.

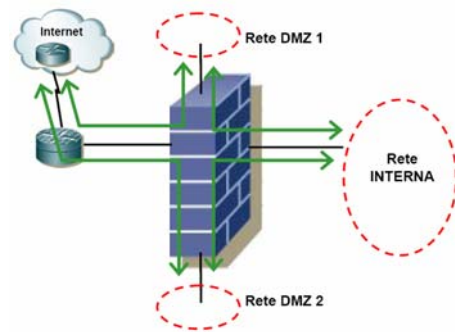
## 2. Tecnologia per la sicurezza nelle reti

### 2.1. Firewall

Un **firewall** è un componente di difesa perimetrale che può anche svolgere funzioni di collegamento tra due o più tronconi di rete.

Un **firewall** è un componente che filtra tutti i pacchetti entranti ed uscenti, da e verso una rete o un computer, applicando regole che contribuiscono alla sicurezza della stessa.

Un **firewall** è un componente composto da due o più interfacce di rete, su ogni interfaccia è collegata una rete diversa.



Nella progettazione delle politiche di filtraggio è necessario attenersi alla regola generale di **least privilege** secondo la quale: compatibilmente con le specifiche relative alla fornitura dei servizi, la politica di un firewall deve essere *la più stringente possibile*.

### 2.2. Firewall Static Packet Filter

Un firewall **static packet filter** esegue il controllo del traffico basandosi unicamente sulle informazioni contenute negli **header** dei singoli pacchetti. I valori dei parametri degli header dei pacchetti vengono confrontati con le regole definite in una **ACL** (Access Control List) e ammessi o scartati secondo il risultato del confronto.

Ogni pacchetto viene quindi esaminato singolarmente, indipendentemente dai pacchetti precedentemente ricevuti e da quelli successivi.

Le ACL definiscono le regole per il filtraggio statico dei pacchetti in transito.

- Semantica **ACCEPT/DENY**
- Criterio **TOP-DOWN** di filtraggio:
  - la prima regola che viene verificata produce la decisione sul pacchetto
  - il test del pacchetto continua fino a che una regola corrisponde alle caratteristiche del pacchetto oppure fino a che la lista di regole termina
  - di norma esiste una regola di **DEFAULT**

### 2.2.1. Sintassi Cisco

Secondo gli **standard Cisco** si hanno:

- **STANDARD ACL**
  - Numerate tra 0 e 99.
  - Filtrano *solo* gli indirizzi *IP sorgente*
- **EXTENDED ACL**
  - Numerate tra 100 e 199
  - Filtrano indirizzi *IP sorgente, destinatario, protocollo, porte UDP e TCP e tipo/codice* messaggi ICMP

#### Formato ACL Standard

**Access-list numero azione sorgente [wild card] | any**

**numero:** da 0 a 99 per ACL Standard

**azione:** *permit* oppure *deny*

**sorgente:** indirizzo IP sorgente

**wild card:** determina la parte dell'indirizzo IP da verificare e quella da ignorare (inverso della netmask)

**any:** qualunque valore

#### Formato ACL Extended

**Access-list numero azione tipo sorgente [wild card] opzioni destinazione [wild card] [log]**

**numero:** da 100 a 199 per ACL Extended

**azione:** *permit* oppure *deny*

**sorgente:** indirizzo IP sorgente

**destinazione:** indirizzo IP sorgente

**tipo:** IP, UDP o TCP

**wild card:** determina la parte dell'indirizzo IP da verificare e quella da ignorare (inverso della netmask)

**opzioni:** porte TCP/UDP, Tipo/Codice ICMP, operatori speciali

**log:** scrive un messaggio in un log per ogni pacchetto verificato da una regola (opzionale)



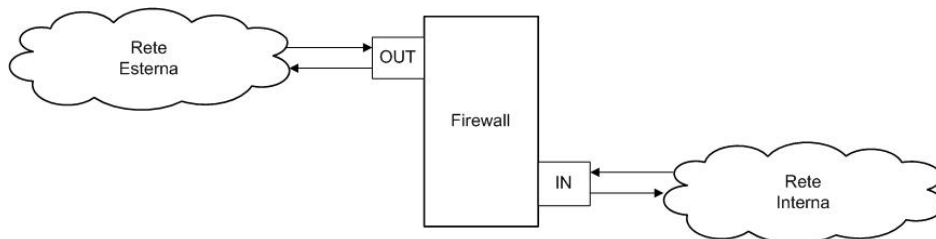
### 2.2.2. Politiche di Static Packet Filter

Una ACL Static Packet Filter stabilisce le regole di sicurezza di *due interfacce di rete*.

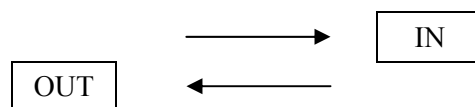
Ogni regola stabilisce la politica da applicare al pacchetto *entrante* in una interfaccia (eth0) e *uscente* dall'altra interfaccia (eth1).

Al flusso dati che transita dalle due interfacce viene *arbitrariamente* attribuito un nome:

- IN: eth0 → eth1, flusso *entrante* in eth0
- OUT: eth1 → eth0, flusso *entrante* in eth1



Una ACL è definita da una tabella come la seguente:



Interfaccia1

Interfaccia2

N°	Direzione IN/OUT	Protocol	Source IP	Source Port	Destination IP	Destination Port	Flag ACK	Azione

Il formalismo di riferimento è il seguente:

- **Direzione:** indica la direzione del traffico
  - se non ambiguo, IN/OUT
  - nei restanti casi indicazione delle zone sorgente e destinataria (es. DMZ->Internet), oppure delle interfacce (es. eth0->eth1)
- **Protocollo**
  - TCP, UDP, ICMP, IP
- **IP Sorgente/Destinatario:** valori degli indirizzi attraverso l'uso di variabili
  - DMZ := 159.149.10.0/24
  - WebServer := 159.149.70.11 and 159.149.70.12
  - ANY per indicare qualsiasi indirizzo
- **Porta Sorgente/Destinataria:** numero della porta
  - valore (es. 21)
  - range (es. >1023)
  - ANY per indicare qualsiasi numero
- **Flag ACK**
  - TCP:
    - 0/1 quando IP Sorgente stabilisce la connessione verso l'IP Destinatario
    - 1 quando IP Sorgente (passivo) risponde all'IP Destinatario
  - per tutti gli altri protocolli viene utilizzato \*\*
- **Azione**
  - Permit/Deny

Tips: 192.168.0.0/24 è un indirizzo privato di classe B utile nella configurazione di una rete interna

### 2.2.3. Esempi di Static Packet Filter

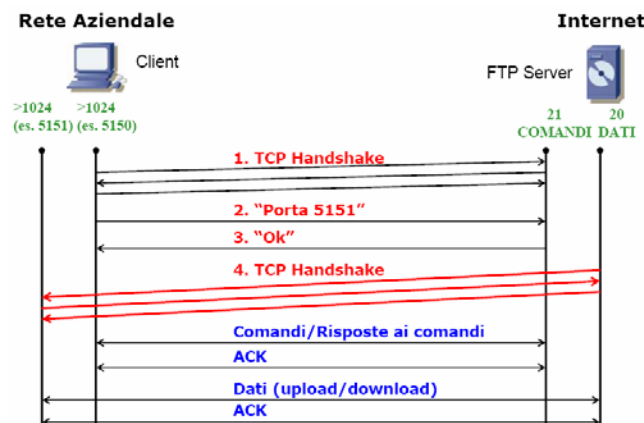
Applicazioni come Telnet, SSH, rlogin, etc. sono semplici da gestire:

- per loro natura implicano *ruoli ben definiti*: client e server
- il *pattern di scambio di messaggi* è un semplice **request/reply**

Altre applicazioni possono avere protocolli più elaborati, ad esempio perché prevedono ruoli client e server multipli o perché basate su pattern di scambio di messaggi più sofisticati.

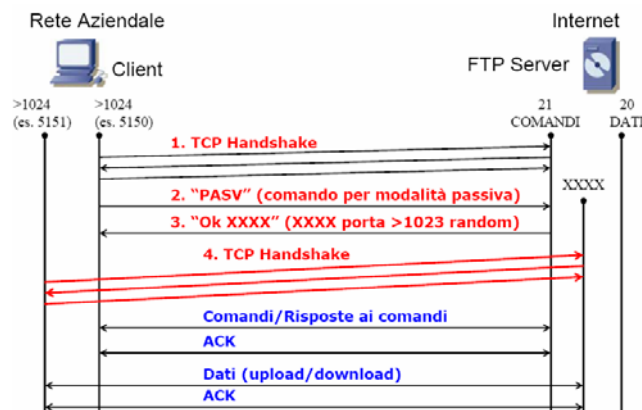
#### FTP Attivo (default)

- l'FTPClient apre una connessione per i comandi verso l'FTPServer
  - da porta >1023 → porta 21
- l'FTPServer apre una connessione per i dati verso l'FTPClient
  - da porta 20 → porta >1023 (porta negoziata sul canali comandi)



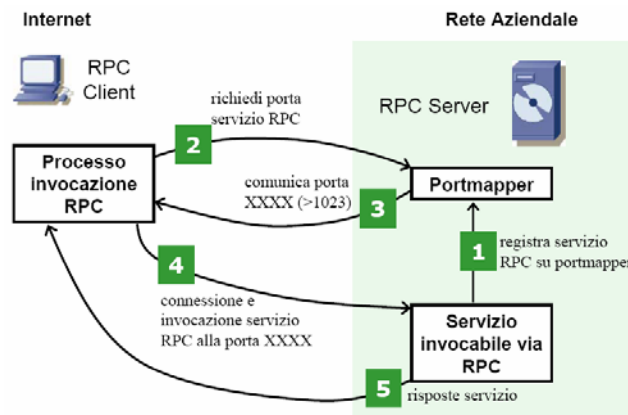
#### FTP Passivo

- l'FTPClient apre una connessione per i comandi verso l'FTPServer
  - da porta >1023 → porta 21
- l'FTPClient apre una connessione per i dati verso l'FTPServer
  - da porta >1023 → porta >1023 (porta negoziata sul canali comandi)



## RPC

- l'RPCClient apre una connessione verso il Portmapper per richiedere la porta associata a un servizio
  - da porta >1023 → porta 111
- l'RPCClient apre una connessione verso l'RPCServer
  - da porta >1023 → porta >1023 (porta ottenuta tramite il precedente comando)



## Firewall e Least Privilege

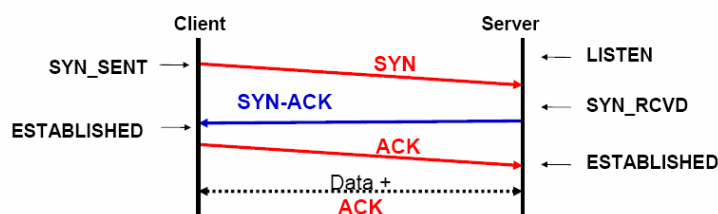
L'*allocazione dinamica* delle porte in ascolto rappresenta un problema per la sicurezza perché non è possibile garantire il principio di *least privilege*. Infatti, non è possibile stabilire in modo puntuale quali porte abilitare e dunque non è possibile definire delle politiche di sicurezza stringenti.

## 2.3. Stateful Firewall

Un *stateful firewall* esegue il controllo del traffico basandosi sulle informazioni contenute negli *header* dei singoli pacchetti e sul contenuto della *connection table*, che mantiene in memoria lo stato delle connessioni attive. Ogni pacchetto viene analizzato sia singolarmente sia in relazione con quelli precedentemente ricevuti, appartenenti alla stessa sessione attiva.

Il principio di funzionamento è il seguente:

- se il server/porta è nello stato di LISTEN (NEW) si deve controllare l'ACL
- se il server/porta è nello stato di ESTABLISHED i pacchetti possono essere autorizzati verificando le informazioni della connection table



**Osservazione:** a differenza dello static packet filter, un stateful firewall blocca anche i pacchetti isolati, cioè quei pacchetti che non fanno parte di nessuna sessione.

### Stati di una Sessione TCP e Stateful Filtering

- Ricezione di un SYN (stato NEW) → verifica dell'ACL (come per Packet Filter)
  - Se connessione non autorizzata → Deny
  - Se connessione autorizzata (stato ESTABLISHED) → Accept e scrittura di una entry nella connection table contenente le informazioni della *sessione*
- Ricezione di pacchetti successivi → verifica della *connection table* (dove sono presenti anche le ACL)

Una *sessione* è univocamente identificata da: *IP\_Src+Port\_Src+IP\_Dst+Port\_Dst+Protocol*

Esempio di Connection Table				
Src_IP	Src_Prt	Dst_IP	Dst_Prt	Timeout
192.168.1.202	1783	192.168.1.207	137	18/40
192.168.1.202	1885	192.168.1.207	80	43/50
192.168.1.202	1884	192.168.1.207	80	43/50
192.168.1.202	1797	192.168.1.207	23	35/50
192.168.1.202	1796	192.168.1.207	22	35/50
192.168.1.202	1795	192.168.1.207	21	35/50
192.168.1.202	1798	192.168.1.207	25	35/50
192.168.1.202	1907	192.168.1.207	80	43/50

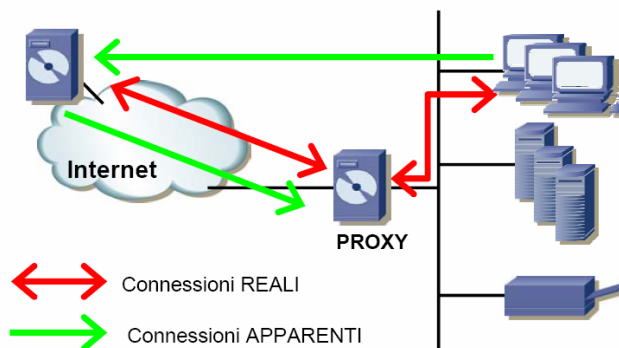
### Politiche di Stateful Firewall

Le politiche per un stateful firewall sono le seguenti:

- **TCP:** Vengono definite le regole relative agli *stati* di una comunicazione TCP. Non serve specificare nelle regole il valore che assumono i bit del campo FLAG.
- **UDP:** Viene gestito uno *pseudo-stato* correlando semplicemente indirizzi IP e porte (sorgente/destinazione). Non avendo un protocollo di terminazione viene settato un *time-out* predefinito.

## 2.4. Proxy

Un **proxy** è un componente che si interpone nella comunicazione tra un client ed un server, inoltrando le richieste e le risposte dall'uno all'altro. Il client si collega al proxy invece che al server, e gli invia delle richieste. Il proxy a sua volta si collega al server e inoltra la richiesta del client, riceve la risposta e la inoltra al client. Un proxy disaccoppia la comunicazione rendendola indiretta.



I più comuni tipi di proxy sono:

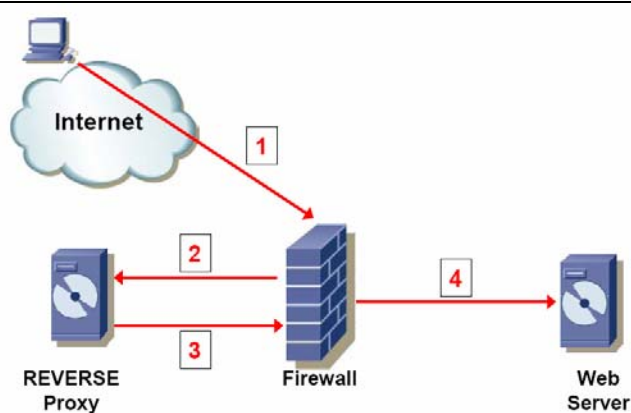
- **Web Proxy**, che funziona da cache dei pagine web
- **Anonymizing Proxy**, utilizzato per rendere anonima una connessione web
- **Forward Proxy (Proxy)**, usato per garantire l'accesso a risorse esterne alla rete da parte di utenti interni
- **Reverse Proxy**, usato per garantire l'accesso a risorse interne alla rete da parte di utenti esterni
- **Proxy Firewall**, mediano connessioni applicative e gestiscono aspetti di sicurezza dei protocolli

### 2.4.1. Reverse Proxy

Un **reverse proxy** dirotta le richieste provenienti dall'esterno della rete verso risorse interne alla rete.

Un tipico utilizzo è il seguente:

1. l'utente esterno chiede una connessione verso un Web Server
2. la connessione viene direzionata verso il Reverse Proxy
3. verificata l'autorizzazione e filtrata la connessione, la richiesta passa nuovamente per il firewall
4. la richiesta viene inoltrata al Web Server



Ci sono diversi vantaggi nell'utilizzo di un reverse proxy:

- può fornire un livello aggiuntivo di difesa ulteriore schermando il web server
- è possibile limitare le vulnerabilità tipiche di una piattaforma di un server

### 2.4.2. Proxy Firewall

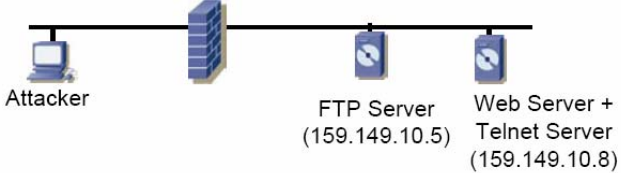
Un **proxy firewall** è un proxy che applica politiche di sicurezza a livello di protocollo **applicativo** (FTP, HTTP, SMTP). Esso, quindi, deve *interpretare* il protocollo applicativo.

Le sue performance sono molto critiche ma fornisce un'analisi migliore del traffico applicativo rispetto ad uno stateful firewall.

### 2.4.3. FTP Bounce Attack

L'**FTP Bounce Attack** è un esempio di attacco che sfruttando la vulnerabilità del protocollo FTP riesce a bypassare un stateful firewall ma potrebbe essere bloccato da un firewall applicativo.

In questo attacco l'FTPClient utilizza il comando PORT per indicare all'FTPServer quale *indirizzo IP* e *numero porta* utilizzare per il trasferimento dati.

<p>L'FTP Bounce Attack avviene in questo modo:</p> <ol style="list-style-type: none"> <li>1. FTPClient si connette all'FTPServer (deve essere autorizzato a farlo)</li> <li>2. FTPClient indica, attraverso il comando PORT, l'indirizzo della macchina vittima e la porta del servizio da attaccare</li> <li>3. l'FTPServer apre una connessione verso la macchina vittima che è protetta da attacchi esterni ma non da quelli interni (si suppone che l'FTPServer e la vittima siano nella stessa zona di sicurezza)</li> <li>4. FTPClient chiede il trasferimento di un file dall'FTPServer, contenente comandi dannosi per la vittima</li> <li>5. l'FTPServer trasferisce il file richiesto verso la macchina vittima</li> </ol>	
--	--

Possibili *contromisure*:

- impedire che l'indirizzo IP specificato dal comando PORT sia diverso da quello dell'FTPClient
  - impedire che il numero di porta sia < 1023 per evitare attacchi a servizi standard quali smtp, pop3, etc...
- Entrambi i controlli possono essere effettuati dall'FTPServer o da un Firewall applicativo

### 2.4.4. Articolo “A Quantitative Study of Firewall Configuration Errors”

- 1) No regola stealth: ogni comunicazione non consentita deve essere droppata
- 2) Verifica regole implicite: gestire cose standard come DNS, udp, icmp, tcp
- 3) Gestione insicura del FW: usare protocolli autenticati e crittografati
- 4) Gestire la configurazione della rete da poche macchine
- 5) Non usare macchine esterne alla rete per gestire il FW
- 6) Utilizzo del netBIOS: non utilizzare netBIOS e RPC
- 7) Zone-Spanning: regole su zone ben definite di indirizzi IP (interni, esterni, di interfacce diverse)
- 8) Non utilizzare mai ANY su connessioni entranti/ uscenti.

Generalmente gli attacchi sono efficaci perchè la configurazione dei FW è stata fatta in modo poco attento e con regole lasche

## 2.5. Network Address Translation (NAT)

Il **Network Address Translation (NAT)** consente di convertire gli indirizzi IP nel passaggio tra le due interfacce del firewall. Tipicamente viene usato per sfruttare le *classi di indirizzi IP riservate* e non instradabili (172.16, 10, e 192.168)

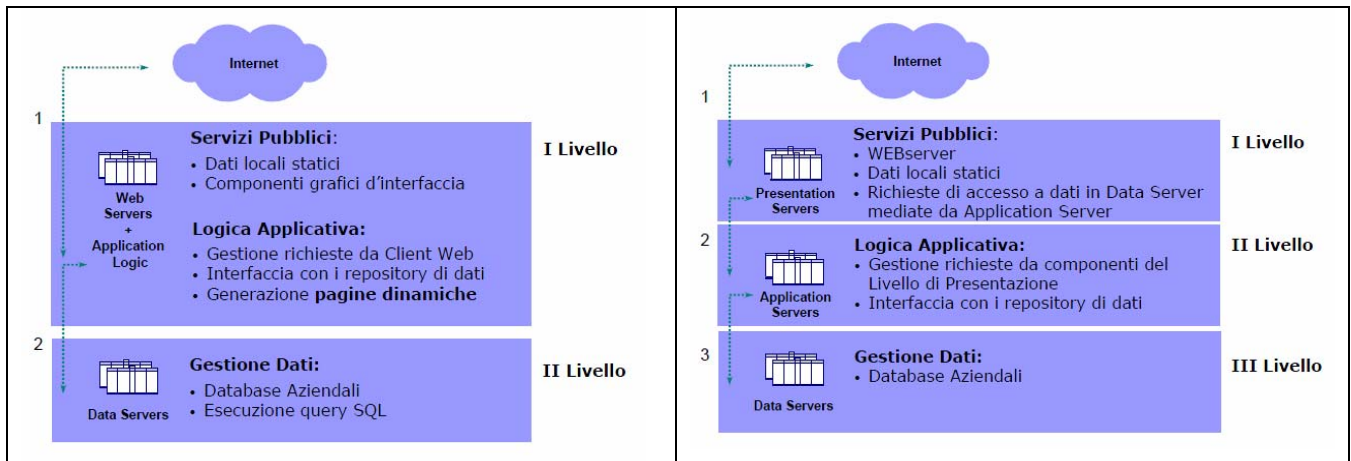
Il NAT non è propriamente una soluzione per la sicurezza della rete aziendale, ma una tecnica di gestione della rete. Fornisce un beneficio rilevante: *maschera gli indirizzi* effettivamente utilizzati all'interno della rete.

Tipologie di NAT:

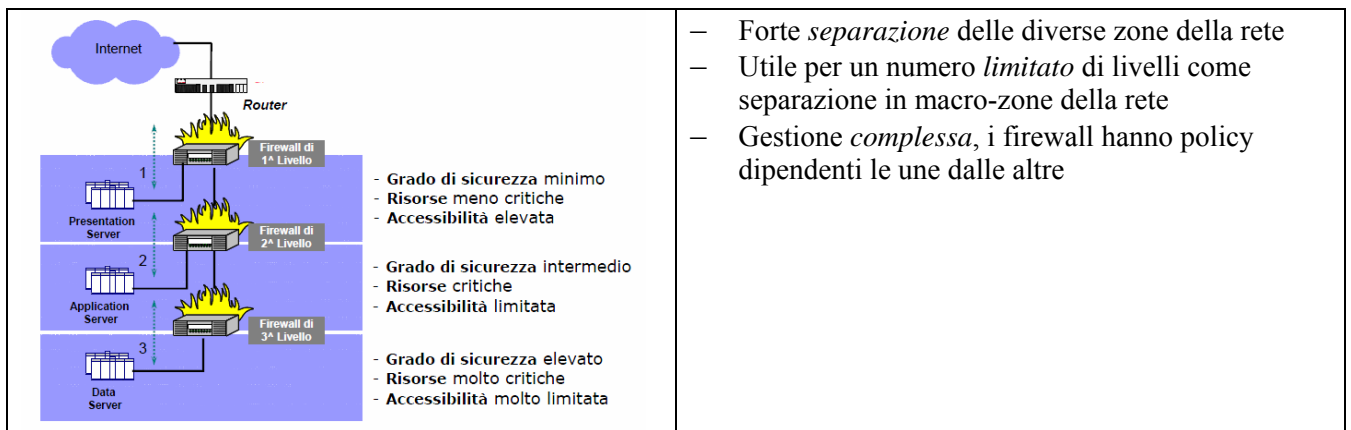
- **NAT Statico**  
Indirizzi IP interni vengono mappati staticamente in indirizzi IP pubblici (l'associazione è fissa)
- **NAT Dinamico**  
L'associazione tra indirizzo IP interno e indirizzo IP pubblico avviene dinamicamente a run-time
- **PAT (Port Address Translation)**  
L'associazione tra connessione interna (Host → Proxy) e connessione esterna (Proxy → Internet) avviene modificando la porta sorgente

## 2.6. Disegno di una architettura per la sicurezza

### 2.6.1. Modelli Architetturali

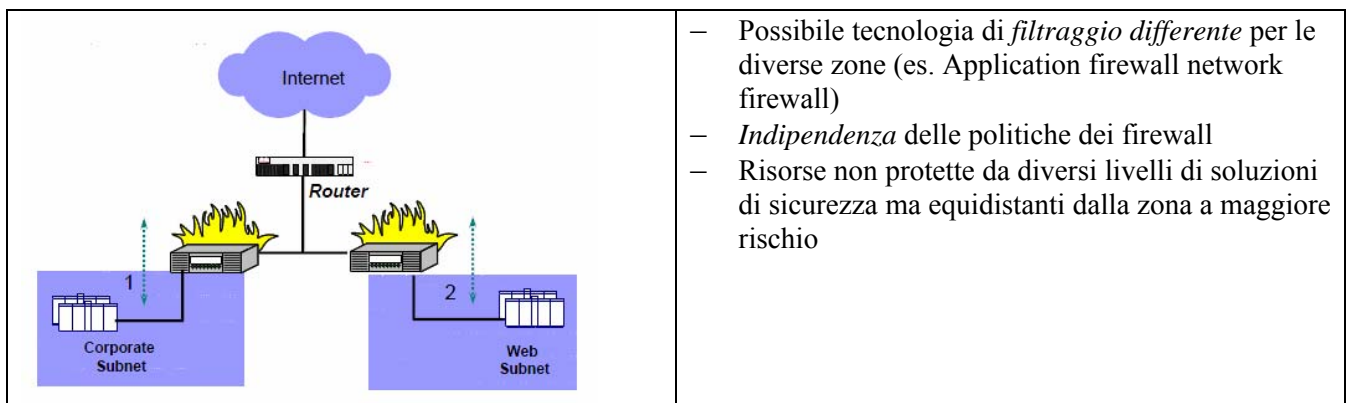


### Architettura con Firewall in Serie



- Forte *separazione* delle diverse zone della rete
- Utile per un numero *limitato* di livelli come separazione in macro-zone della rete
- Gestione *complessa*, i firewall hanno policy dipendenti le une dalle altre

### Architettura con Firewall in Parallelo



- Possibile tecnologia di *filtraggio differente* per le diverse zone (es. Application firewall network firewall)
- *Indipendenza* delle politiche dei firewall
- Risorse non protette da diversi livelli di soluzioni di sicurezza ma equidistanti dalla zona a maggiore rischio

## 2.7. Sdoppiamento di Servizi

La suddivisione tra zone di sicurezza per *servizi pubblici* e per *servizi interni* coinvolge servizi che sono utilizzati da entrambe le zone. Un caso tipico sono la posta elettronica e DNS , dove occorre scegliere se continuare a fornirli con un *unico componente* o sdoppiarli in *due componenti* da sincronizzare.

### 2.7.1. Mail server e Mail relay

Il servizio di posta elettronica può essere gestito da due mail server:

**Mail server primario:**

- Inserito nella zona dei *servizi interni*, utilizzato come repository dati, gestisce il traffico vero e proprio.

**Mail server relay:**

- Inserito nella zona dei *servizi pubblici*, utilizzato come relay o gateway tra l'esterno e l'interno.

Questo modello di architettura offre i seguenti vantaggi:

- avere due componenti - uno nella zona pubblica, uno nella zona privata - permette di scegliere *software differenti*
- ognuno dei due componenti può essere *configurato* in maniera appropriata e specifica
- il componente nella zona pubblica agisce da semplice *relay*, quindi le email rimangono nella zona più protetta
- il mail relay può agire da *gateway antivirus e antispam*

### 2.7.2. DNS Server Interno ed Esterno

Il servizio DNS può essere gestito da due DNS server:

**DNS Esterno**

- Serve richieste di utenti esterni alla rete aziendale e fornisce le informazioni pubbliche (es. da nslookup)
- Riceve query da utenti esterni per informazioni riguardo host pubblicamente accessibili della rete aziendale, incluso l'MX server (il Mail Relay)

**DNS Interno**

- Serve gli utenti e i servizi della rete interna
- Riceve query da utenti interni per informazioni su host sia della intranet aziendale che di Internet
- Per le query che il DNS Interno non è in grado di risolvere può sia contattare il DNS Esterno che DNS predefiniti (es. DNS dell'ISP). Queste si chiamano *query ricorsive*.

I due DNS mantengono *informazioni differenti* (l'esterno solo quelle pubbliche l'esterno, l'interno tutte quelle della intranet) e hanno connessioni con zone a diverso grado di sicurezza.

Questo modello di architettura offre i seguenti vantaggi:

- *separazione fisica delle informazioni* riguardante servizi pubblici da quelle riguardanti servizi della intranet
- assegnazione a *diverse zone di sicurezza* per la protezione delle informazioni
- *isolamento del DNS pubblico* dalla rete interna nel caso di compromissione

### 2.7.3. DNS Attacks

I DNS sono da sempre uno dei target preferiti per intrusioni.

Tra i motivi la natura *pubblica* del servizio, che se compromesso, può essere usato come testa di ponte per estendere l'intrusione a *componenti interni* (così come per qualunque altro servizio pubblico).



## DNS Cache Poisoning

Un DNS server conserva in maniera permanente solo i record delle macchine del dominio sul quale è autoritativo. Per ogni altro nome deve generare una query ad altri DNS (protocollo UDP).

Le risposte da altri DNS vengono conservate in una cache per un certo tempo. Le informazioni mantenute in cache possono essere soggette a compromissione (*cache poisoning*).

1. L'attacker configura un DNS per un proprio dominio (es. *www.attacker.org*) contenente associazioni fasulle (es. *www.abc.it* → *<IP attacker>*)

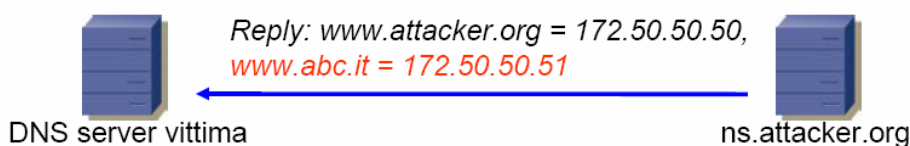
2. L'attacker invia una DNS query al DNS server vittima chiedendo di risolvere *www.attacker.org*



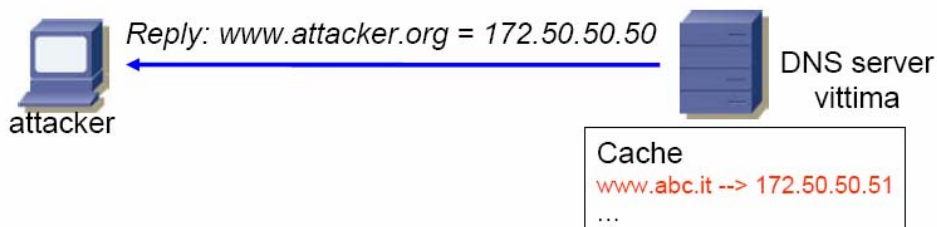
3. Il DNS server vittima, a sua volta contatta il DNS autoritativo per il dominio *attacker.org*, ovvero il DNS dell'attacker (*ns.attacker.org*)



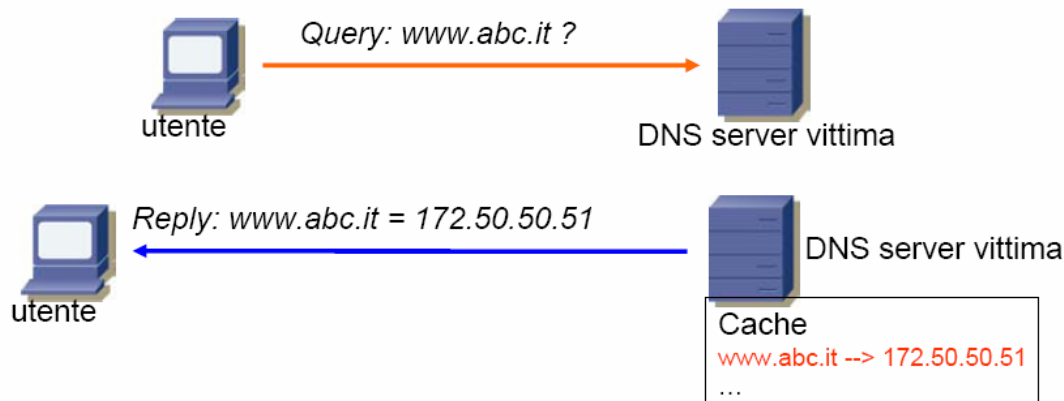
4. Il DNS dell'attacker (*ns.attacker.org*) risolve la query e nella risposta restituisce non solo l'IP richiesto, ma anche un'associazione fasulla (es. *www.abc.it* → *<IP attacker>*), questa operazione prende il nome di *zone transfer* (protocollo TCP)



5. Il DNS server vittima ora può rispondere all'attacker ma contemporaneamente memorizza in cache le associazioni ottenute, compresa quella fasulla



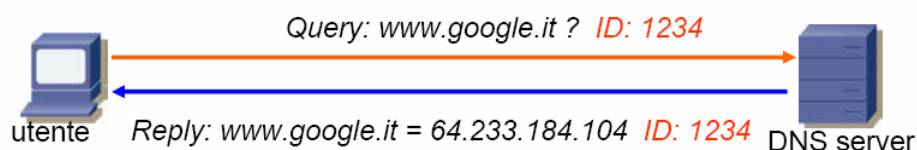
6. Il DNS server vittima, fino a che la cache non viene aggiornata, restituirà un'associazione fasulla a tutti i client che lo interrogheranno



**Contromisure:** È una tecnica molto comune, evitabile semplicemente configurando i DNS a non accettare alcun zone transfer da DNS (impedire connessioni TCP sulla porta 53) oppure accettarlo solo da DNS autenticati.

## DNS ID Spoofing

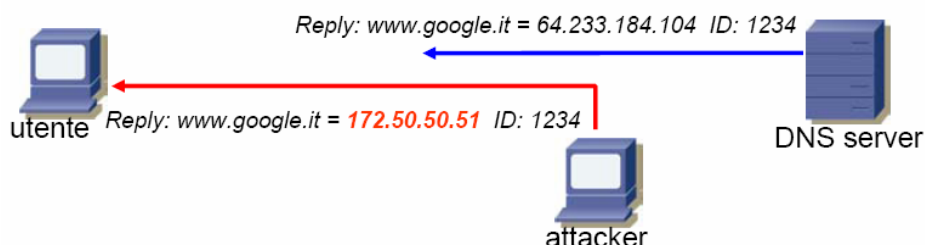
Un client quando esegue una DNS query genera anche un identificativo pseudo-casuale che il DNS server copierà nella risposta. Tale identificativo serve per riconoscere la connessione fra client e server, dato che il protocollo DNS utilizza UDP come trasporto.



1. L'attacker è tra utente e DNS server ed è in grado di sniffare la DNS Query, leggendone l'ID



2. L'attacker può facilmente creare un pacchetto UDP con una risposta DNS, contenente l'ID corretto e una falsa associazione (la risposta deve arrivare prima di quella del DNS Server corretto)



Questo attacco è difficile da eseguire perchè vincolato a limitazioni fisiche:

- la risposta dell'attacker deve giungere al client prima di quella del DNS server
- l'attacker deve essere in grado di sniffare la query DNS

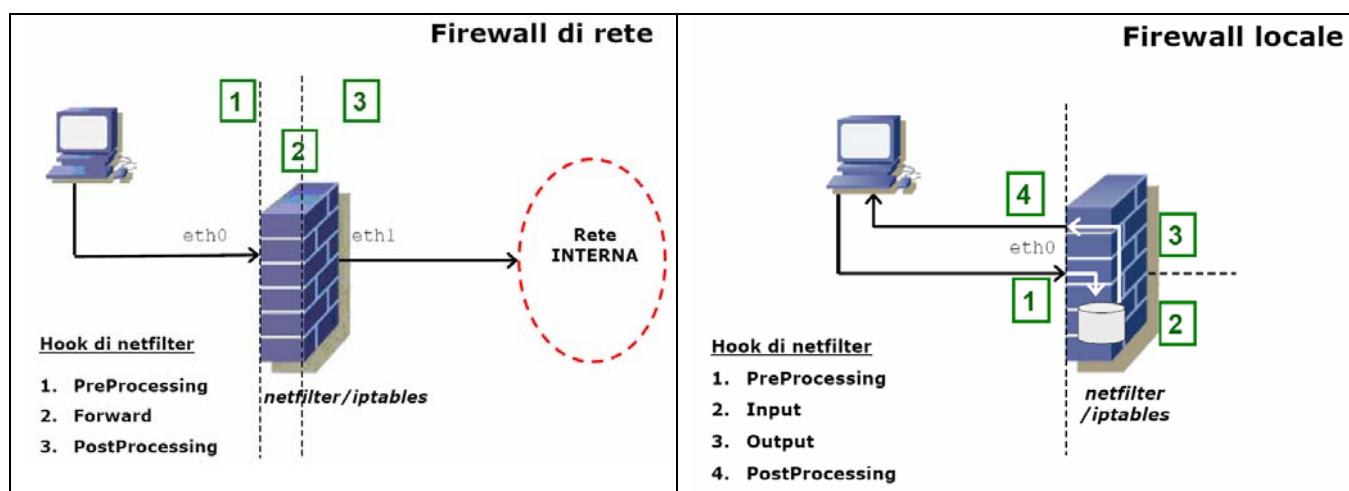
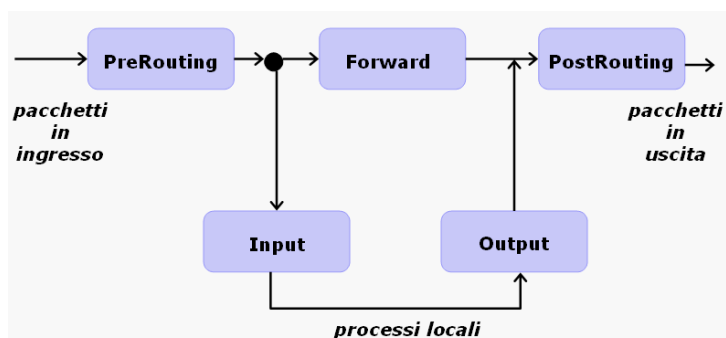
**Criticità:** Il principale motivo di successo di tale attacco è l'assenza di autenticazione del DNS server (l'indirizzo IP usato dall'attacker sarà spoofed), tuttavia per il suo successo ci sono forti limitazioni fisiche.

## 2.8. netfilter/iptables

*netfilter-iptables* fornisce la possibilità di accedere ai pacchetti mentre vengono processati dallo stack IP.

Inoltre definisce degli *stati* durante i quali i pacchetti possono essere manipolati:

- **PREROUTING** – il pacchetto è stato ricevuto dalla scheda di rete ma non ancora processato dallo stack IP. Potrebbe essere destinato sia ad un *processo locale* che destinato ad una *rete* connessa ad una diversa interfaccia del firewall
- **POSTROUTING** – il pacchetto sta per essere immesso sulla rete, sia che sia stato generato da un processo locale che forwardato da una diversa interfaccia del firewall
- **FORWARD** – il pacchetto viene *forwardato* ad una diversa interfaccia del firewall
- **INPUT** – pacchetto *destinato* a un processo locale
- **OUTPUT** – pacchetto *originato* da un processo locale



### 2.8.1. Filtraggio con iptables

Una regola iptables per il filtraggio dei pacchetti ha il seguente formato:

```
iptables [-t filter] <operation> <chain> <parameters> -j <action>
```

Operation	(on a chain)
-A	appendi
-I	inserisci
-D	cancella
-R	sostituisci

Chain	
INPUT	pacchetto <i>destinato</i> a un processo locale
OUTPUT	pacchetto <i>originato</i> da un processo locale
FORWARD	pacchetto <i>forwardato</i> ad una diversa interfaccia

Action	
ACCEPT	pacchetto <i>accettato</i>
DROP	pacchetto <i>scartato</i>
REJECT	pacchetto <i>rifiutato</i> generando una risposta ICMP
LOG	logging

Parameters	
-i <interface>	interfaccia di ingresso
-o <interface>	interfaccia di uscita
-p [tcp udp icmp]	protocollo tcp, udp, icmp
-s <ind_IP>	indirizzo IP sorgente
-d <ind_IP>	indirizzo IP destinazione

Parameters TCP/UDP	
--sport <num>	porta sorgente
--sport <from>:<to>	
--dport <num>	porta destinazione
--dport <from>:<to>	

Parameters TCP	
--tcp-flags <mask_flags> <active_flags>	confronta la maschera dei flags con quella dei flag attivi --tcp-flags SYN,ACK,FIN ACK (SYN=0, ACK=1, FIN=0) --tcp-flags SYN,ACK,RST SYN (SYN=1, ACK=0, RST=0)

Parameters ICMP	
--icmp-type <type>/<code>	verifica il pacchetto con <b>type</b> e <b>code</b>

Parameters for stateful firewall	
-m state --state <match_state>	verifica il pacchetto negli stati indicati da <b>match_state</b> (separati da virgola)
NEW	il pacchetto è il <i>primo di una connessione</i> TCP (primo pacchetto dell'handshake, flag SYN attivo)
ESTABLISHED	il pacchetto è parte di una <i>connessione attiva</i> . lo stato diventa <b>ESTABLISHED</b> : – TCP: alla ricezione del SYN/ACK – UDP: alla ricezione del primo pacchetto di risposta
RELATED	il pacchetto è associato ad una connessione attiva, pur non facendone realmente parte (es. pacchetto ICMP di segnalazione di una condizione di errore)
INVALID	pacchetto che non viene riconosciuto appartenere a nessuna delle precedenti (es. pacchetto malformato, protocollo non riconosciuto, ecc...)

Parameters (others)	
<b>-m iprange --src-range &lt;from&gt;--&lt;to&gt;</b>	indirizzi sorgente da <i>from</i> a <i>to</i>
<b>-m iprange --dst-range &lt;from&gt;--&lt;to&gt;</b>	indirizzi destinazione da <i>from</i> a <i>to</i>
<b>-m multiport --source-port &lt;ports&gt;</b>	porte sorgente (separate da virgola)
<b>-m multiport --destination-port &lt;ports&gt;</b>	porte destinazione (separate da virgola)
<b>-m multiport --port &lt;ports&gt;</b>	porte sorgente e destinazione (separate da virgola)
<b>-m length --length &lt;min&gt;:&lt;max&gt;</b>	lunghezza del pacchetto compresa <i>min</i> fra e <i>max</i>

**-m <modulo> <commando> <parametri>** specifica il caricamento del *modulo* per eseguire il *commando* con i *parametri*

Parameters for action REJECT	
<b>-j REJECT --reject-with &lt;options&gt;</b>	
tcp-reset icmp-net-unreachable icmp-host-unreachable icmp-port-unreachable icmp-protocol-unreachable icmp-net-prohibited icmp-host-prohibited	N.B I messaggi generati da REJECT devono essere autorizzati attraverso un'ulteriore regola

Osservazione: i messaggi generati da **REJECT** devono essere autorizzati attraverso un'ulteriore regola

Parameters for action LOG	
<b>-j LOG --log-prefix &lt;message&gt;</b>	
<i>message</i>	messaggio di testo tra doppi apici

Common preliminary commands	
<b>iptables -F INPUT</b>	cancella la catena INPUT
<b>iptables -F OUTPUT</b>	cancella la catena OUTPUT
<b>iptables -F FORWARD</b>	cancella la catena FORWARD
<b>iptables -P INPUT DROP</b>	imposta la politica default deny per la catena INPUT
<b>iptables -P OUTPUT DROP</b>	imposta la politica default deny per la catena OUTPUT
<b>iptables -P FORWARD DROP</b>	imposta la politica default deny per la catena FORWARD

#### Comandi utili:

<b>sudo tail -f /var/log/messages</b>	consente di visualizzare i messaggi di LOG in tempo reale
<b>sudo cat /proc/net/ip_conntrack</b>	consente di vedere lo stato delle connessioni
<b>sudo modprobe -c   sort   grep ip_</b>	consente di vedere i moduli ip_ che sono caricati
<b>sudo modprobe ip_conntrack</b>	consente di caricare il modulo per la gestione dello stato
<b>sudo modprobe ip_conntrack_ftp</b>	consente di caricare il modulo per la gestione FTP

### Considerazioni su FTP canali dati

Regola “lasca” in assenza di *ip\_conntrack\_ftp* (*nf\_conntrack\_ftp* in ubuntu!!).

```
IPTABLES -A OUTPUT -o $INET_IFACE -s $INET_IP -p tcp -m state --state NEW,
ESTABLISHED, RELATED -j ACCEPT
$IPTABLES -A INPUT -i $INET_IFACE -d $INET_IP -p tcp -m state --state ESTABLISHED,
RELATED -j ACCEPT
```

Regola “lasca” in assenza di *ip\_conntrack\_ftp* (*nf\_conntrack\_ftp* in ubuntu!!).

```
IPTABLES -A OUTPUT -o $INET_IFACE -s $INET_IP -p tcp -m state --state
ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A INPUT -i $INET_IFACE -d $INET_IP -p tcp -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

Notare che, nonostante l'apparente similitudine, la seconda consente l'apertura di porte TCP esclusivamente verso un host su cui sia già stata aperta una connessione FTP (*ip\_conntrack\_ftp* entra nel merito del protocollo FTP).

### 3. Sistemi IDS

Gli IDS sono sistemi di **monitoraggio**

- sistemi *passivi* non attivi
- *generano allarmi* non prevengono intrusioni

Esistono due tipologie di IDS che si differenziano in base al tipo di informazioni che raccolgono:

**NIDS (Network Intrusion Detection System):** sistemi che utilizzano le informazioni raccolte da analizzatori di traffico di rete.

**HIDS (Host-based Intrusion Detection System):** sistemi che analizzano informazioni relative all'attività locale ad un computer (log di sistema, accesso a file critici, ecc...)

Gli IDS sono fondamentalmente basati su due approcci differenti: **Anomaly Detection** e **Misuse Detection**

#### 3.1.1. Anomaly Detection : caratteristiche

**Anomaly detection:** si considera lo stato normale del sistema monitorato e se ne verificano eventuali scostamenti.

Le anomalie possono essere rilevate rispetto a:

- **eventi singoli:** esempio azioni “anomale” di un utente rispetto un profilo d'uso predefinito
- **dati aggregati:** tipicamente, deviazioni rispetto dati statistici

<b>Vantaggi</b>	<b>Svantaggi</b>
<ul style="list-style-type: none"> <li>– Non dipende dalla conoscenza puntuale di tutte le modalità di intrusione/vulnerabilità</li> <li>– Anche intrusioni effettuate con tecniche o vulnerabilità non note possono venire rilevate</li> </ul>	<ul style="list-style-type: none"> <li>– Molto complesso da realizzare e oneroso da gestire</li> <li>– Non fornisce informazioni sullo scopo di un attacco</li> </ul>

#### 3.1.2. Anomaly Detection : tecniche

##### Profili d'uso

Vengono definiti profili d'uso dei sistemi nello stato “normale” (non compromesso, privo di utilizzi anomali, ecc.).

- È stato il primo approccio studiato (profili di uso degli utenti)
- Molti altri approcci (analisi di sequenze di system call, stati di un sistema distribuito, ecc.)
- Impiego di tecniche come data mining, sistemi esperti, data fusion, analisi bayesiana, ecc...

<b>Vantaggi</b>	<b>Svantaggi</b>
Monitoraggio dello stato dei sistemi anche a run-time	Elevata complessità nel definire con accuratezza il comportamento normale e nel gestire i cambiamenti

##### Algoritmi di Hashing (solo HIDS)

Vengono calcolati gli hash (MD5, SHA-1) dei file di sistema presi da una distribuzione originale e li si confrontano periodicamente con gli hash del sistema da monitorare. (es. Tripwire, [www.tripwire.com](http://www.tripwire.com))

<b>Vantaggi</b>	<b>Svantaggi</b>
Monitoraggio dell'accesso ai file critici con granularità al singolo file Utili per controllare la gestione dei server	Utilizzabili solo dopo che le modifiche sono avvenute

##### Protocol Anomaly Detection (solo NIDS)

Viene analizzato il traffico di rete rispetto alle specifiche del protocollo applicativo

- Analisi dello scambio di messaggi e del contenuto del payload dei pacchetti
- Applicazione del vecchio concetto di proxy applicativo, rivisitato e utilizzato diversamente (monitoraggio anziché filtraggio)

<b>Vantaggi</b>	<b>Svantaggi</b>
Specifiche standard per molti dei protocolli più diffusi Utili per rilevare tunnel applicativi	Implementati solo per alcuni dei protocolli più comuni Elevato carico computazionale

### 3.1.3. Misuse Detection : caratteristiche

**Misuse Detection:** si da una descrizione dei casi indesiderati e si verifica l'eventuale comparsa

Lo scopo è di rilevare utilizzi non conformi alle specifiche di protocolli (di rete, applicativi) e applicazioni

<b>Vantaggi</b>	<b>Svantaggi</b>
– Semplicità di funzionamento rispetto a tecniche di anomaly detection	– Rilevano solo attacchi noti – Limitazioni di un approccio di tipo “permit all”

### 3.1.4. Misuse Detection : tecniche

#### Signature Detection

Definizione di **pattern** (firme, signature) predefiniti di usi non conformi e analisi degli eventi (di rete, di sistema, nei log) rispetto tale elenco di pattern. È la tecnica che si è affermata e diffusa nella maggior parte dei sistemi in produzione.

<b>Vantaggi</b>	<b>Svantaggi</b>
– Semplicità di funzionamento rispetto a tecniche di anomaly detection – Aggiornamenti incrementali e centralizzati da parte del produttore del sistema	– Rilevano solo attacchi che corrispondono a <i>regole note</i> , non rilevano nuove vulnerabilità e varianti – Necessità di <i>personalizzazione</i> , l'elenco di firme deve essere adattato alle specificità della rete monitorata – <i>Complessità di gestione/configurazione</i> , problema del bilanciamento tra falsi positivi e negativi

### 3.1.5. Falsi positivi e Falsi negativi

**Falsi Positivi:** Allarmi provocati da eventi legittimi. Sono falsi allarmi dovuti al meccanismo di rilevazione che ha interpretato un evento legittimo come illegittimo

**Falsi Negativi:** Allarmi mancati in presenza di eventi illegittimi. Sono i casi in cui il sistema interpreta un evento illegittimo come legittimo e pertanto non genera alcun allarme

L'*efficienza* e la *qualità* di un sistema di monitoraggio dipendono dall'entità di questi due parametri. Le due grandezze sono in *relazione inversa*, seppur non necessariamente proporzionale.

Al contrario di quello che si potrebbe pensare l'*accuratezza* dell'IDS è dominato dalla percentuale di falsi positivi.

Rilevazione accurata (firme dettagliate)	– Problema di performance – Falsi Negativi a causa di varianti o mutazioni
Rilevazione lasca (firme generiche)	– Performance meno critiche – Falsi Positivi dovuti a eventi simili ma non analoghi all'evento analizzato

### 3.1.6. Risposte Automatiche

La tipica risposta di un NIDS al verificarsi di un evento che verifica una firma è la *generazione di un allarme*. Tuttavia è possibile configurare un NIDS in modo tale che al verificarsi di un evento che verifica una firma questo generi automaticamente un'*azione* allo scopo di rispondere attivamente ad una presunta intrusione senza richiedere l'intervento diretto di un operatore.

Le tecniche più tradizionali sono:

- **Reset di Sessioni (Session Sniping):**
  - il NIDS invia un pacchetto contenente un **RST** ad entrambe le parti coinvolte nella connessione
  - tali pacchetti devono apparire ai ricevitori come *inviati dalle corrispondenti controparti*, non dal NIDS, altrimenti verrebbero ignorati; devono quindi contenere *valori corretti per i numeri di sequenza, numero di ack, ecc...*
- **Aggiornamento del Firewall**
  - la rilevazione di un allarme può essere sfruttata per riconfigurare automaticamente le regole di un firewall

### 3.2. Tecniche di Evasione

Le principali tecniche per bypassare i controlli di un NIDS, che definiamo *tecniche di evasione*, si basano sul processo di frammentazione IP.

Se un pacchetto IP è troppo grande per il livello sottostante, questo può essere suddiviso in tanti frammenti. Un host che riceve un frammento deve mantenerlo in memoria e quando ha ricevuto tutti i frammenti può riassemblare il pacchetto originale.

Partiamo con alcune definizioni utili a capire le differenti tecniche:

- **TTL**: Quando un router riceve un pacchetto/frammento decrementa di 1 il valore del campo TTL, se tale valore risulta uguale a 0, il pacchetto/frammento viene scartato e inviato un messaggio ICMP “Time Exceeded TTL” al mittente (ICMP type=11, code=0)
- **Timeout di riassettaggio**: Un frammento viene mantenuto da un host per un tempo massimo, trascorso questo tempo, il frammento viene scartato e inviato un messaggio ICMP “Time Exceeded Timeout” al mittente (ICMP type=11, code=1). Il messaggio ICMP *non deve* essere inviato al mittente se l’host non ha in memoria il primo frammento.

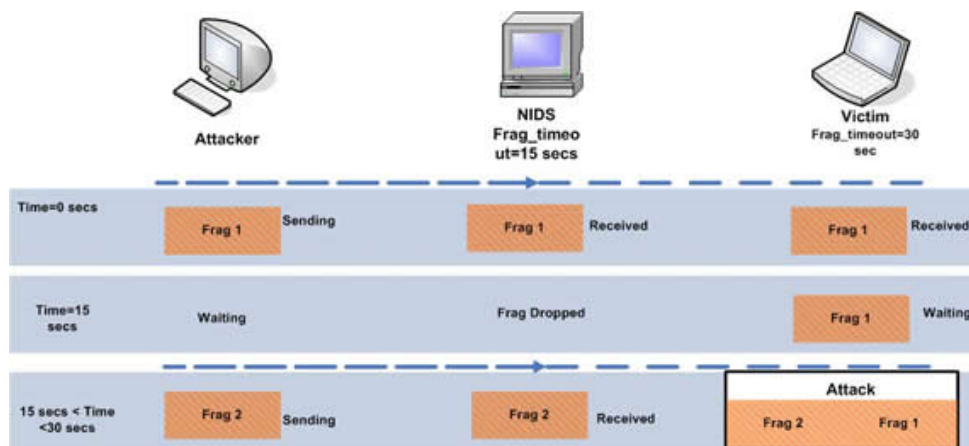
#### 3.2.1. Attacco basato sul timeout

##### Caso 1: Timeout di riassettaggio dell’IDS minore di quello della vittima

Supponiamo che il timeout di riassettaggio del NIDS sia 15 secondi e quello della vittima sia 30 secondi e il pacchetto sia formato da 2 frammenti.

L’attaccante procede in questo modo:

1. invia il frammento 1
2. attende 15 secondi (timeout NIDS), a questo punto il NIDS scarta il frammento 1
3. dopo 15 secondi ma entro i 30 secondi (timeout vittima) invia il frammento 2, la vittima riassetta i frammenti 1 e 2 come facenti parte di un unico pacchetto



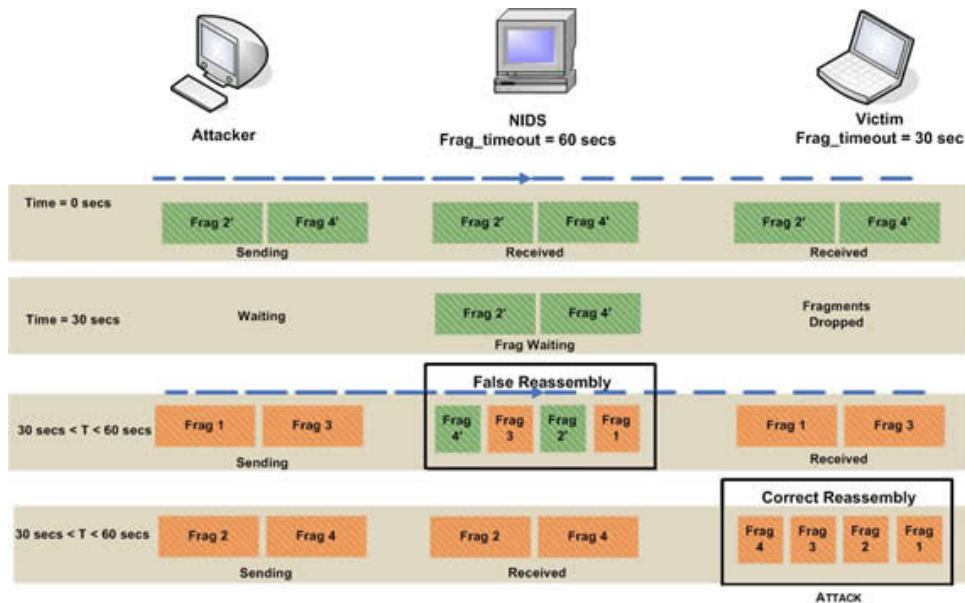


## Caso 2: Timeout di riassettaggio dell'IDS maggiore di quello della vittima

Supponiamo che il timeout di riassettaggio del NIDS sia 60 secondi e quello della vittima sia 30 secondi e il pacchetto sia formato da 4 frammenti.

L'attaccante procede in questo modo:

1. invia i frammenti 2' e 4' con payload falso
2. attende 30 secondi (timeout vittima), a questo punto la vittima scarta i frammenti 2' e 4'
3. dopo 30 secondi ma entro i 60 secondi (timeout NIDS) invia i frammenti 1 e 3, il NIDS riassetta i frammenti 1, 2', 3, 4' come facenti parte di un pacchetto innocuo, la vittima riceve i frammenti 1 e 3
4. dopo 30 secondi ma entro i 60 secondi (timeout NIDS) invia i frammenti 2 e 4, la vittima riassetta i frammenti 1, 2, 3, 4 come facenti parte di un unico pacchetto



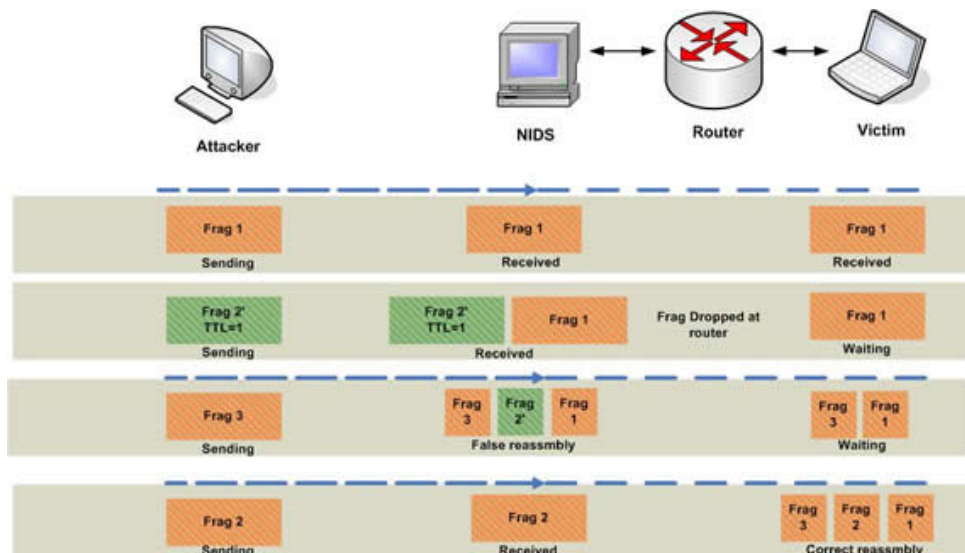
**Osservazione:** L'attaccante inizia l'attacco con i frammenti 2 e 4 perchè la vittima quando scarta i i frammenti non è tenuta a segnalare tale condizione anomala con un messaggio ICMP. Infatti, la segnalazione di errore avviene unicamente quando il ricevente ha in memoria il frammento 1 (primo).

### 3.2.2. Attacco basato sul TTL

Questo tipo di attacco avviene quando è presente un router fra NIDS e vittima. Supponiamo che il pacchetto sia formato da 3 frammenti.

L'attaccante procede in questo modo:

1. invia il frammento 1
2. invia il frammento 2' con TTL=1 e payload falso, il router scarta il frammento che non arriva alla vittima
3. invia il frammento 3, il NIDS riassetta i frammenti 1, 2', 3 come facenti parte di un pacchetto innocuo
4. invia il frammento 2, la vittima riassetta i frammenti 1, 2, 3 come facenti parte di un unico pacchetto



### 3.2.3. Attacco basato sulla sovrapposizione (overlapping)

Questo tipo di attacco avviene quando il NIDS e la vittima utilizzano differenti politiche di riassettaggio dei frammenti.

Un sistema quando riceve il frammento  $x$  che è già stato ricevuto, può adottare due politiche:

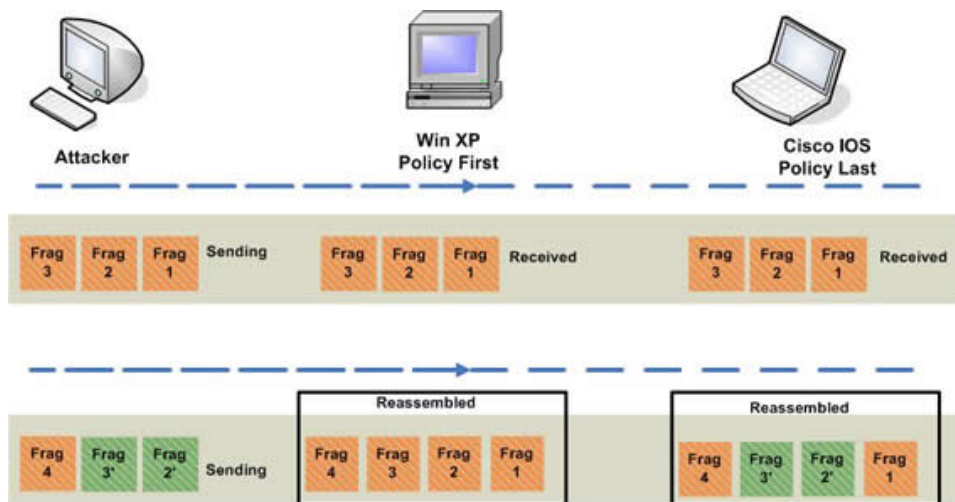
- *first reassembly policy*, mantenere il primo frammento  $x$  (e scartare l'ultimo ricevuto)
- *last reassembly policy*, mantenere l'ultimo frammento  $x$  (e scartare il primo ricevuto)

Windows 95/98/NT4/ME/W2K/XP/2003 utilizzano la politica *first reassembly policy*.

Cisco IOS utilizza la politica *last reassembly policy*.

L'attaccante procede in questo modo:

1. invia i frammenti 1, 2, 3
2. invia i frammenti 2', 3', 4, il NIDS (WinXP) riassettra i frammenti 1, 2, 3, 4, la vittima riassettra i frammenti 1, 2', 3', 4



### 3.3. Snort

*Snort* è il più noto e diffuso NIDS utilizzato in moltissime realtà aziendali al pari di molti tool commerciali. Le definizioni delle firme sono visibili, possono essere analizzate e modificate a piacere.

#### 3.3.1. Struttura delle firme di Snort

```
1. alert tcp $EXTERNAL_NET any -> $HOME_NET any \
2. (msg:"SCAN SYN FIN"; \
3. flags:SF; \
4. reference:arachnids,198; classtype:attempted-recon; sid:624; rev:2;)
```

**Header della firma (riga 1):**

- **azione** (*alert*) eseguita in corrispondenza di un match positivo
- **protocollo** da analizzare (es. *tcp*)
- **mittente** e **destinatario** (indirizzi IP, netmask e porte)

**Body (righe 2-4):**

- **messaggio** di alert (es. "SCAN SYN FIN")
- **condizioni** per il pattern matching (es. *Flags:SF*)
- **informazioni** riguardanti la firma stessa (es. *reference, classtype, sid, rev*)

Il meccanismo di *pattern matching* esegue un puro confronto tra stringhe tra i valori di ogni pacchetto in transito e header/opzioni della firma stessa.

#### reference, classtype, sid, rev

Keyword	Description	Format
reference	The reference keyword allows rules to include references to external attack identification systems	<b>reference:</b> <id system>,<id>;
classtype	The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class	<b>classtype:</b> <class name>;
sid	The sid keyword is used to uniquely identify Snort rules.	<b>sid:</b> <snort rules id>;
rev	The rev keyword is used to uniquely identify revisions of Snort rules.	<b>rev:</b> <revision integer>;

#### flow

The flow keyword is used in conjunction with TCP stream reassembly(Stream4). It allows rules to only apply to certain directions of the traffic flow.

This allows rules to only apply to clients or servers. This allows packets related to \$HOME NET clients viewing web pages to be distinguished from servers running in the \$HOME NET.

The established keyword will replace the flags: A+ used in many places to show established TCP connections.

Option	Description
to_client	Trigger on server responses from A to B
to_server	Trigger on client requests from A to B
from_client	Trigger on client requests from A to B
from_server	Trigger on server responses from A to B
established	Trigger only on established TCP connections
stateless	Trigger regardless of the state of the stream processor

#### Format

```
flow: [(established|stateless)][,(to_client|to_server|from_client|from_server)]
```

#### Example

```
alert tcp !$HOME_NET any -> $HOME_NET 21 (msg:"cd incoming detected"; \
flow:from_client; content:"CWD incoming"; nocase;)

alert tcp !$HOME_NET 0 -> $HOME_NET 0 (msg: "Port 0 TCP traffic"; \
flow:stateless;)
```

## flags

The flags keyword is used to check if specific TCP flag bits are present.

The following bits may be checked:

**F** - **FIN** (LSB in TCP Flags byte)

**S** - **SYN**

**R** - **RST**

**P** - **PSH**

**A** - **ACK**

**U** - **URG**

**1** - **Reserved bit 1** (MSB in TCP Flags byte)

**2** - **Reserved bit 2**

**0** - **No TCP Flags Set**

The following modifiers can be set to change the match criteria:

**+** - **match on the specified bits, plus any others**

**\*** - **match if any of the specified bits are set**

**!** - **match if the specified bits are not set**

To handle writing rules for session initiation packets such as ECN where a SYN packet is sent with the previously reserved bits 1 and 2 set, an option mask may be specified. A rule could check for a flags value of S,12 if one wishes to find packets with just the syn bit, regardless of the values of the reserved bits.

### Format

**flags: [!|\*|+]<FSRPAU120>[,<FSRPAU120>];**

### Example

This example checks if just the SYN and the FIN bits are set, ignoring reserved bit 1 and reserved bit 2.

```
alert tcp any any -> any any (flags:SF,12;)
```

## content

It allows the user to set rules that search for specific content in the packet payload. it can contain mixed text and binary data.

The binary data is generally enclosed within the pipe (|) character and represented as bytecode. Bytecode represents binary data as hexadecimal numbers and is a good shorthand method for describing complex binary data.

### Format

**content: [!] "<content string>;"**

### Example

Mixed Binary Bytecode and Text in a 'content' keyword

```
alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)
```

Negation Example

```
alert tcp any any -> any 80 (content:!"GET";)
```

The content keyword has a number of modifier keywords: **nocase**, **offset**, **depth**, etc...

### nocase

The nocase keyword allows the rule writer to specify that the Snort should look for the specific pattern, ignoring case. nocase modifies the previous 'content' keyword in the rule.

### Format

**nocase;**

### Example

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

### depth

The depth keyword allows the rule writer to specify how far into a packet Snort should search for the specified pattern.

A depth of 5 would tell Snort to only look for the specified pattern within the first 5 bytes of the payload.

### Format

**depth: <number>;**

## offset

The offset keyword allows the rule writer to specify where to start searching for a pattern within a packet. An offset of 5 would tell Snort to start looking for the specified pattern after the first 5 bytes of the payload.

### Format

**offset:** <number>;

### Example

Combined Content, Offset and Depth Rule. Skip the first 4 bytes, and look for cgi-bin/phf in the next 20 bytes

```
alert tcp any any -> any 80 (content: "cgi-bin/phf"; offset:4; depth:20;)
```

## pcre

The pcre keyword allows rules to be written using perl compatible regular expressions. For more detail on what can be done via a pcre regular expression, check out the PCRE web site <http://www.pcre.org>

### Format

**pcre:**[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUB]";

### Example

This example performs a case-insensitive search for the string BLAH in the payload.

```
alert ip any any -> any any (pcre:"/BLAH/i");
```

## fragbits

The fragbits keyword is used to check if fragmentation and reserved bits are set in the IP header.

The following bits may be checked:

**M** - More Fragments

**D** - Don't Fragment

**R** - Reserved Bit

The following modifiers can be set to change the match criteria:

**+** match on the specified bits, plus any others

**\*** match if any of the specified bits are set

**!** match if the specified bits are not set

### Format

**fragbits:**[+\*!]<[MDR]>;

### Example

This example checks if the More Fragments bit and the Do not Fragment bit are set.

```
fragbits:MD+;
```

## id

The id keyword is used to check the IP ID field for a specific value. Some tools (exploits, scanners and other odd programs) set this field specifically for various purposes, for example, the value 31337 is very popular with some hackers.

### Format

**id:**<number>;

### Example

This example looks for the IP ID of 31337.

```
id:31337;
```