

# Sniffing and Spoofing pt. 2

## Spoofing

Normalmente quando mandiamo dei pacchetti possiamo scegliere l'indirizzo di destinazione, ma non quello di sorgente, anzi sarà il computer nell'invio il pacchetto che imposta il source address.

Se vogliamo inviare un pacchetto TCP, non possiamo per esempio settare i bit SYN e FIN contemporaneamente perchè normalmente non possono essere settati insieme.

Network Attack: fulcro principale è l'invio di pacchetti costruiti inserendo nell'header delle informazioni false e mirate.

Esempi:

1. TCP SYN flooding attack: indirizzo IP della sorgente generato automaticamente
2. TCP session hijacking attack: usare IP address di altre persone, settare la giusta sequenza di numeri e i numeri di porta nell'header
3. DNS cache poisoning attack: mandare risposte fake di tipo DNS e quindi bisogna inserire indirizzo IP del server DNS corrispondente nel campo della sorgente e anche le relative info del DNS presenti nell'header e payload del DNS

### Obiettivo Principale:

creare dei pacchetti che contengano delle info, che di solito vengono messe di default seguendo le regole dell'architettura TCP/IP, in realtà false che matchano con un'altra sorgente e che siano randomiche per esempio, a seconda dell'attacco da fare.

### Tools:

4. Scapy

## Mandare normali pacchetti usando un socket

**VEDERE CODICE `udp_client.c`**

### Da ricordare:

1. creare il socket
  2. dare le info sulla destinazione
  3. mandare il pacchetto udp -> verrà costruito a seconda delle info date
- ```
```shell gcc -o udp_client udp_client.c nc -lvu 9090 ./udp_client ```
```

## Mandare pacchetti con un raw socket

Usiamo di nuovo il raw socket che ci permette di costruire l'intero pacchetto in un buffer, compreso anche l'indirizzo IP nell'header e tutti gli altri campi, che poi viene dato al socket

-> stiamo dicendo al sistema operativo di lasciarci fare da soli e inserire nell'header le informazioni che gli diamo noi senza che le mette lui.

STEP:

1. costruire tutto il pacchetto in un buffer
2. mandare il pacchetto sulla rete

**VEDERE CODICE `spoof.c`** -> seconda fase

### Da ricordare:

3. creare il raw socket: usiamo la `socket()` dove primo argomento è la famiglia dell'indirizzo, secondo argomento è il tipo di socket (datagram, stream, raw...), terzo argomento è il protocollo e settando `IPPROTO_RAW` abilitiamo anche `IP_HDRINCL` cioè l'header included
4. settare le opzioni del socket: settiamo `IP_HDRINCL` il che è anche ridondante
5. settare le info sulla destinazione: normalmente dovremmo settare IP, famiglia, porta che il sistema usa per costruire l'IP header, mentre ora basta settare famiglia e IP di destinazione grazie al quale aiutiamo il kernel a capire grazie al MAC address corrispondente, se la destinazione sta nella stessa network o meno
6. mandare il pacchetto falso: usiamo la `sendto()` dove il primo argomento è il socket, il secondo è il puntatore al pacchetto, il terzo sono i flag che influenzano la funzione e noi mettiamo 0 perchè non settiamo nessun flag, il quarto e il quinto sono la destinazione e la size. Il sistema manderà il pacchetto così com'è settato tranne per il checksum e per altri campi che vengono settati in automatico.

## Costruire pacchetti ICMP

VEDERE CODICE `spoof_icmp.c`, `checksum.c`, `spoof.c`

1. Creiamo un buffer e lo riempiamo di zero, poi creiamo il pacchetto ICMP riempiendo type e checksum
2. Riempiamo l'IP header con source fittizio e destination che vogliamo noi senza il checksum e altri campi che il sistema riempie da solo
3. Mandiamo il pacchetto

La funzione in checksum invece utilizza un accumulatore a 32 bit (sum), aggiunge parole sequenziali di 16 bit e, alla fine, ripiega tutti i bit di riporto dai 16 bit più significativi ai 16 bit meno significativi.

```
gcc -o icmp_spoof spoof_icmp.c checksum.c spoof.c
sudo ./icmp_spoof
```

Si può vedere su wireshark la richiesta e la risposta

## Costruire pacchetti UDP

VEDERE CODICE `spoof_udp.c`, `checksum.c`, `spoof.c`

1. Creiamo il buffer per l'IP packet e calcoliamo l'offset per il payload; mettiamo `data` nella regione payload nel buffer
2. Riempiamo l'UDP Header che ha tre campi: source, destination, port size, checksum
3. Riempiamo l'IP header compreso di campi di protocollo e lunghezza
4. Mandiamo il pacchetto

```
#terminale 1
nc -luv 9090
#terminale 2
sudo ./udp_spoof
```

Si può vedere su wireshark l'invio del pacchetto udp con ip fittizio

## Sniffing then Spoofing

In molti attacchi la procedura è:

1. sniffing del traffico
2. spoofing con un pacchetto fittizio come risposta

Esempio:

Facciamo sniffing di pacchetto UDP che arrivano e poi se i pacchetti catturati hanno come `dport = 9999` allora mandiamo una reply spoofed.

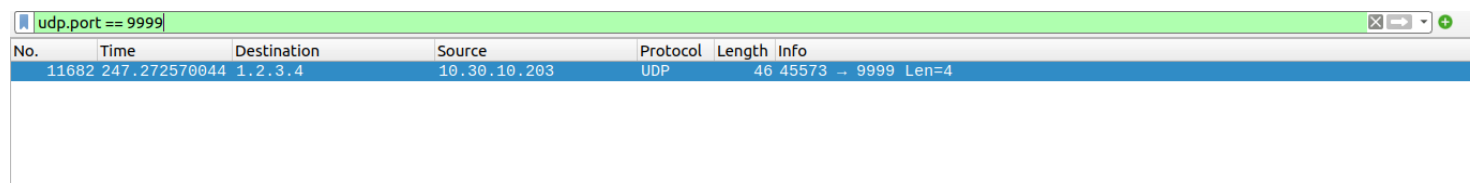
### VEDERE CODICE `sniff_spoof_udp.c`

Il codice è una funzione di callback invocata nella funzione `pcap_loop` al posto di `got_packet` che costruisce un pacchetto fittizio e lo manda

La funzione di callback prende una copia del pacchetto UDP intercettato, cambia in data il contenuto con un nuovo messaggio, fa il swap degli indirizzi e rimanda indietro.

Il campo della destinazione viene dato dal pacchetto che viene catturato.

```
nc -u 1.2.3.4 9999
aaa
#This is a spoofed reply!
```



A screenshot of the Wireshark network protocol analyzer. The top filter bar shows 'udp.port == 9999'. The packet list pane shows a single packet, No. 11682, at time 247.272570944, destined to 1.2.3.4, sourced from 10.30.10.203, with protocol UDP, length 46, and info '45573 -> 9999 Len=4'. The packet details pane is empty.

| No.   | Time          | Destination | Source       | Protocol | Length | Info                |
|-------|---------------|-------------|--------------|----------|--------|---------------------|
| 11682 | 247.272570944 | 1.2.3.4     | 10.30.10.203 | UDP      | 46     | 45573 -> 9999 Len=4 |

## Sniffing e Spoofing usando python

Python ha un modulo chiamato `Scapy` che permette di creare programmi che fanno sniffing e spoofing in modo più semplice.

```
sudo pip3 install scapy
```

### VEDERE CODICE `sniff.py`

1. viene usata la funzione `sniff` dove viene impostato il filtro BPF come quello di `pcap`
2. viene passata la funzione di callback che stampa alcune info del pacchetto

### VEDERE CODICE `icmp_spoof.py`

1. creare l'oggetto IP header riempiendo i vari campi e lasciando gli altri di default
2. creare l'oggetto ICMP header
3. fare lo stack di IP header e ICMP header con `/` in modo da creare un pacchetto ICMP con l'IP header
4. mandare il pacchetto

## VEDERE CODICE `udp_spoof.py`

1. creare l'IP header con source e destination address
2. creare l'UDP header con source e destination port
3. creare il payload per l'UDP
4. fare lo stack delle tre parti insieme
5. mandare il messaggio

```
sudo ./udp_spoof #con indirizzo wifi

nc -luv 9090
#uscirà il messaggio fittizio inviato
```

su wireshark si può vedere mettendo l'interfaccia `any`

## VEDERE CODICE `sniff_spoof_icmp.py`

1. sniff con il filtro di icmp e source address di chi manda icmp ad un altro indirizzo ip e con funzione di callback
2. funzione di callback che stampa il pacchetto originale mandato, costruisce il pacchetto spoofed e lo manda di risposta

```
ping <ip in una stessa rete>
#altro terminale
sudo ./sniff_spoof_icmp
#si vedono i pacchetti che vengono duplicati
```

Scapy mette poi a disposizione molte altre API.

## Sniffing e Spoofing usando un approccio ibrido

Problema principale:

1. Python è un interpreted programming language, è più lento a livello di processore, ma permette di costruire i pacchetti in modo semplice perchè non c'è bisogno di riempire ogni campo del pacchetto quando lo si costruisce, ma molti valori o saranno di default o sarà scapy a riempirli o sono marcati come `none`
2. C è un linguaggio molto più veloce a livello di processore, ma è molto più difficile andare a costruire i pacchetti falsi  
Siccome la velocità serve nel mandare il pacchetto finto e non nel costruirlo, possiamo costruire il pacchetto con scapy e salvarlo in un file come un template, poi possiamo prendere il pacchetto costruito e spedirlo con un programma in C che è più veloce e fa andare gli attacchi a buon fine.

## Costruire il pacchetto con Python Scapy

### VEDERE CODICE `generate_udp.py`

L'unica accortezza qui sta nell'UDP header nel campo checksum che deve essere settato a zero in modo che quando viene mandato, viene considerato un checksum perso e quindi il pacchetto viene accettato lo stesso, altrimenti o il valore lo mette scapy e se sarà incorretto il pacchetto viene scartato -

> lo stesso vale per TCP e ICMP

Mentre per l'IP header mettere 0 al checksum o meno non crea problemi.

Poi il pacchetto viene salvato in un file.

## Mandare il pacchetto con C

### VEDERE CODICE `send_premade_udp.c`

Usiamo il template del pacchetto UDP generato per creare nuovi pacchetti con IP e porta random.

```
sendto(socket, messaggio, lunghezza del messaggio, flags, client address, lunghezza  
client address)
```

Passaggi:

1. creare il raw socket
2. leggere l'udp packet dal template
3. generare source ip e source port randomly e modificare il pacchetto
4. mandare il pacchetto con la funzione apposita

```
python3 generate_udp.py  
gcc -o send_premade_udp send_premade_udp.c  
sudo ./send_premade_udp
```

Per pacchetti TCP e ICMP avremmo dovuto calcolare anche il checksum

## Spoofing software e comandi

### comandi

- `hping3` permette l'invio di pacchetti ICMP, TCP, UDP e di costruire a piacere l'IP Header; l'ultima versione di hping, `hping3`, permette di preparare degli script usando il linguaggio TCL che è un linguaggio di scripting per testare applicazioni e implementa un motore per la descrizione di pacchetti TCP/IP in formato direttamente leggibile; in tal modo, il programmatore può scrivere degli script per manipolazione e analisi di pacchetti TCP/IP a basso livello in un tempo molto breve.

```
#MANDARE PACCHETTI SEMPLICI  
#ping semplice di pacchetti icmp  
sudo hping3 -1 www.google.es  
  
#ping di pacchetti icmp con traceroute che fa vedere gli hop  
sudo hping3 -1 www.google.es -t 1 --traceroute  
  
#mandare pacchetti con il flag SYN  
#se si ottiene SA significa SYN/ACK  
#se si ottiene RA significa RST/ACK  
sudo hping3 -S www.google.es -p 80 (-c 1)  
  
#SCAN PORTA  
#ack scan  
hping3 -c 1 -V -p 80 -s 4060 -A <server>
```

#Xmas Scan -> viene impostato il numero di sequenza TCP a 0 e vengono attivati i flag URG, PSH e FIN all'interno del pacchetto. Se la porta TCP del destinatario è chiusa viene inviato un pacchetto TCP RST in risposta. Se la porta è aperta, il destinatario scarta il pacchetto TCP Xmas e non invia alcuna risposta.

```
hping3 -c 1 -V -p 80 -s 4060 -M 0 -UPF <_server_>
```

#null scan viene impostato il numero sequenziale a 0 e nessun flag è attivo all'interno del pacchetto. Il pacchetto inviato prende il nome di TCP NULL. Se la porta TCP del destinatario è chiusa si riceve in risposta il pacchetto TCP RST. Se la porta è aperta il pacchetto TCP NULL viene scartato e non viene inviata alcuna risposta.

```
hping3 -c 1 -V -p 80 -s 4060 -Y <server>
```

#smurf attack -> spoofed broadcast

```
hping3 -1 --flood -a <IP_VITTIMA INDIRIZZO_BROADCAST>
```

#dos land attack -> spoofed source

```
hping3 -V -c 1000000 -d 120 -S -w 64 -p 445 -s 445 --flood --rand-source  
<IP_VITTIMA>
```

- nmap

**Esperimento:**

**Ho due macchine: pc e raspberry pi nella stessa lan**

#DAL PC

#scanner per vedere le porte aperte

```
nmap -sS -T4 <ip dst raspberry>
```

#vedere quali device sono connessi alla tua stessa rete

```
sudo nmap -sn <tuo ip>/24
```

#1. spoofing dal raspberry pi

#sS: SYN scan è l'opzione di default ed è la più usata per buone ragioni. Può essere effettuato velocemente: effettua la scansione su migliaia di porte al secondo su una rete veloce non limitata da firewall restrittivi. Il SYN scan è relativamente nascosto e poco invasivo, poiché non completa mai le connessioni TCP.

```
sudo nmap -sS -T4 <ip spoofed> -e wlan0 -S <ip dst raspberry>
```

#2. spoofing random addresses

#Quest'opzione invoca una "decoy scan" (ovvero una scansione utilizzando esche) che agli occhi dell'host di destinazione apparirà come se provenisse dagli host specificati come decoy. In questo modo l'IDS della rete bersaglio mostrerà 5-10 port scan provenienti da indirizzi IP singoli, e non potrà capire quale IP è veramente la sorgente dell'attacco e quale IP è usato solo come mascheramento. Nonostante quest'opzione possa essere resa inutile mediante il tracciamento del percorso fatto dai router ("router path tracing"), tecniche di response-dropping e altri meccanismi attivi sono generalmente una tecnica effettiva per nascondere il proprio indirizzo IP.

```
sudo nmap -D RND:<numero addr> <ip dst raspberry>
```

| tcp.port == 80 && ip.src == 13.20.4.68 |               |               |            |          |        |            |                                     |
|--|---------------|---------------|------------|----------|--------|------------|-------------------------------------|
| No.                                    | Time          | Destination   | Source     | Protocol | Length | Info       |                                     |
| 6667                                   | 441.647021463 | 192.168.43.60 | 13.20.4.68 | TCP      | 58     | 65033 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 6682                                   | 442.252600784 | 192.168.43.60 | 13.20.4.68 | TCP      | 58     | 65034 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 8060                                   | 762.199637057 | 192.168.43.60 | 13.20.4.68 | TCP      | 58     | 33673 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 8073                                   | 762.432134407 | 192.168.43.60 | 13.20.4.68 | TCP      | 58     | 33674 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |

| tcp.port == 80 |              |               |                 |          |        |            |                                     |
|----------------|--------------|---------------|-----------------|----------|--------|------------|-------------------------------------|
| No.            | Time         | Destination   | Source          | Protocol | Length | Info       |                                     |
| 76             | 13.104994270 | 192.168.43.60 | 200.203.40.51   | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 77             | 13.105046115 | 192.168.43.60 | 178.16.245.94   | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 78             | 13.105096392 | 192.168.43.60 | 131.96.243.22   | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 79             | 13.105151153 | 192.168.43.60 | 192.168.43.35   | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 80             | 13.105205765 | 192.168.43.60 | 43.200.139.150  | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 81             | 13.105255347 | 192.168.43.60 | 32.159.42.60    | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 82             | 13.105306546 | 192.168.43.60 | 111.214.117.251 | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 83             | 13.105358160 | 192.168.43.60 | 194.161.59.7    | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 84             | 13.105408642 | 192.168.43.60 | 222.231.183.125 | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 85             | 13.105554659 | 192.168.43.60 | 187.247.200.38  | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 86             | 13.105630758 | 192.168.43.60 | 156.156.230.92  | TCP      | 58     | 50054 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 786            | 14.207868537 | 192.168.43.60 | 200.203.40.51   | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 787            | 14.207895378 | 192.168.43.60 | 178.16.245.94   | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 788            | 14.207960189 | 192.168.43.60 | 131.96.243.22   | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 789            | 14.207990877 | 192.168.43.60 | 192.168.43.35   | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 790            | 14.208018726 | 192.168.43.60 | 43.200.139.150  | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 791            | 14.208044925 | 192.168.43.60 | 32.159.42.60    | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 792            | 14.208070236 | 192.168.43.60 | 111.214.117.251 | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 793            | 14.208097115 | 192.168.43.60 | 194.161.59.7    | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 794            | 14.208124162 | 192.168.43.60 | 222.231.183.125 | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 795            | 14.208151014 | 192.168.43.60 | 187.247.200.38  | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 796            | 14.208177261 | 192.168.43.60 | 156.156.230.92  | TCP      | 58     | 50055 → 80 | [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 1513           | 14.249804110 | 192.168.43.35 | 192.168.43.60   | TCP      | 54     | 80 → 50055 | [RST, ACK] Seq=1 Ack=1 Win=0 Len=0  |

- trafgen

trafgen è un generatore di traffico di rete veloce e senza copia per il debug e la valutazione delle prestazioni.

Proviamo a inviare oltre un milione di pacchetti TCP al secondo a un IP

```
sudo apt-get update -y
sudo apt-get install -y netsniff-ng

# Send a single TCP segment with SYN bit set
sudo hping3 -c 1 -I wlp3s0 -p 443 -S <ip non del pc>

# Capture single packet, write to disk
sudo netsniff-ng --in wlp3s0 --out tcpsyn.pcap

# Convert pcap to trafgen config
sudo netsniff-ng --in tcpsyn.pcap --out tcpsyn.cfg

# Send packet x 1000
sudo trafgen --in tcpsyn.cfg --out wlp3s0 --num 1000
```

```
#generare traffico definito nel file di configurazione e mandarlo via interfaccia
indicata
trafgen --dev wlp3s0 --conf trafgen.cfg
```

## Software

- mitmproxy [proxy](#)

mitmproxy è un set di strumenti che fornisce un proxy di intercettazione interattivo con capacità SSL/TLS per HTTP/1, HTTP/2 e WebSocket.

mitmproxy è uno strumento della console che consente l'esame interattivo e la modifica del traffico HTTP. Tutti i flussi sono tenuti in memoria, il che significa che è destinato a prelevare e manipolare

campioni di piccole dimensioni. Usa il tasto ? per visualizzare la documentazione sensibile al contesto da qualsiasi schermata mitmproxy.

```
#primo test
#primo terminale
mitmproxy
#secondo terminale
curl --proxy http://127.0.0.1:8080 "http://wttr.in/Dunedin?0"
curl --proxy http://127.0.0.1:8080 "http://wttr.in/Innsbruck?0"
#sull'altro escono le get request

:
console.view.flow @focus

#secondo test: intercettare una richiesta
mitmproxy
i
#u per le richieste curl e q per le request senza response
set intercept '~u /Dunedin & ~q'
#secondo terminale
curl --proxy http://127.0.0.1:8080 "http://wttr.in/Dunedin?0"
#sul proxy
a #per accettare
X #per scartare

#terzo test modificare una richiesta
mitmproxy
i
#u per le richieste curl e q per le request senza response
set intercept '~u /Dunedin & ~q'
#secondo terminale
curl --proxy http://127.0.0.1:8080 "http://wttr.in/Dunedin?0"
#sul proxy
<enter> #sulla richiesta
e
path
Innsbruck
<esc>
q
a
#ho intercettato la richiesta e modificato il path relativo

#quarto test rispondere ad una richiesta
mitmproxy
i
#u per le richieste curl e q per le request senza response
set intercept '~u /Dunedin & ~q'
#secondo terminale
curl --proxy http://127.0.0.1:8080 "http://wttr.in/Dunedin?0"
#sul proxy
a
r
```



- **Packet builder**

For Windows: colasoft packet builder -> ha un'interfaccia grafica in cui si costruisce un pacchetto impostando ogni campo, dall'header al payload a seconda del protocollo stabilito

For Linux: Ostinato -> ha un client e un server model

**Usiamo anche il raspberry in remoto**

1.

```
sudo apt-get install ostinato
```

```
sudo ostinato #start client e server
```

```
sudo drone #start only server
```

2. attivare wireshark

3. di default ostinato mostra tutte le interfacce disponibili

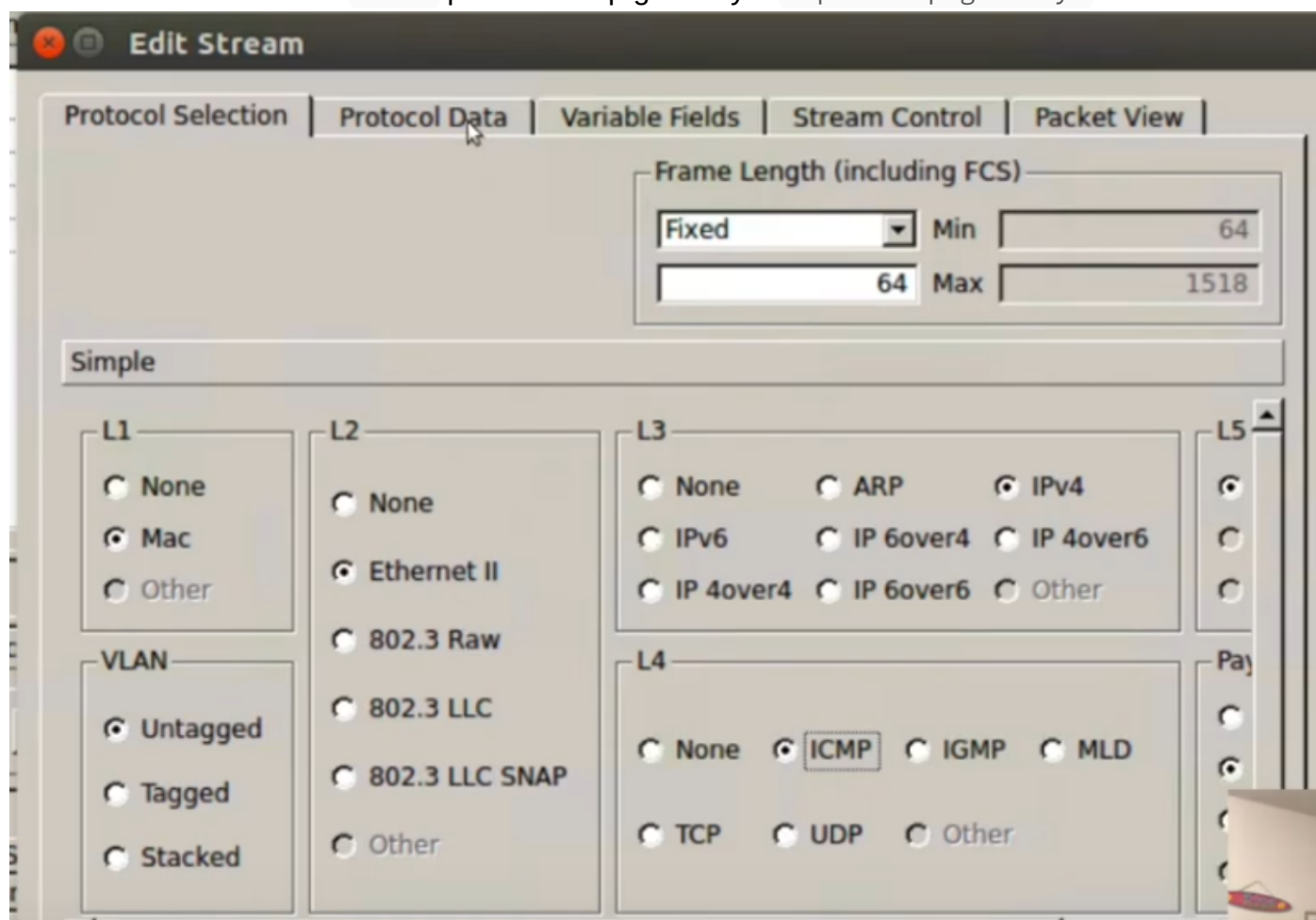
Si fa `file` -> `new stream` in alto a sinistra

Si fa `tasto destro` a destra sullo stream e `edit stream`

Ora prima di tutto mi serve l'indirizzo mac della mia destinazione:

4. se è nella stessa subnet -> `arp -a <ip dst>`

5. se è in un'altra subnet -> `route` per trovare ip gateway e `arp -a <ip gateway>`



**Edit Stream**

Protocol Selection | **Protocol Data** | Variable Fields | Stream Control | Packet View

Media Access Protocol

Ethernet II

Internet Protocol ver 4

Internet Control Message Protocol

Version: ☒ ICMPv4 ☐ ICMPv6

Type: **8 - Echo Request**

Code: **0**

☐ Checksum: **f32d**

Identifier: **1234** Sequence: **0**

Passiamo in Media Access Protocol e inseriamo indirizzi mac della source e della destination

Passiamo in Ethernet II e mettiamo Ethernet type 08 00

Passiamo all' internet protocol

**Internet Protocol ver 4**

☐ Override Version: **4**

☐ Override Header Length (x4): **5**

TOS/DSCP: **00**

☐ Override Length: **46**

Identification: **04 D2**

Fragment Offset (x8): **0**

☐ Don't Fragment ☐ More Fragments

Time To Live (TTL): **127**

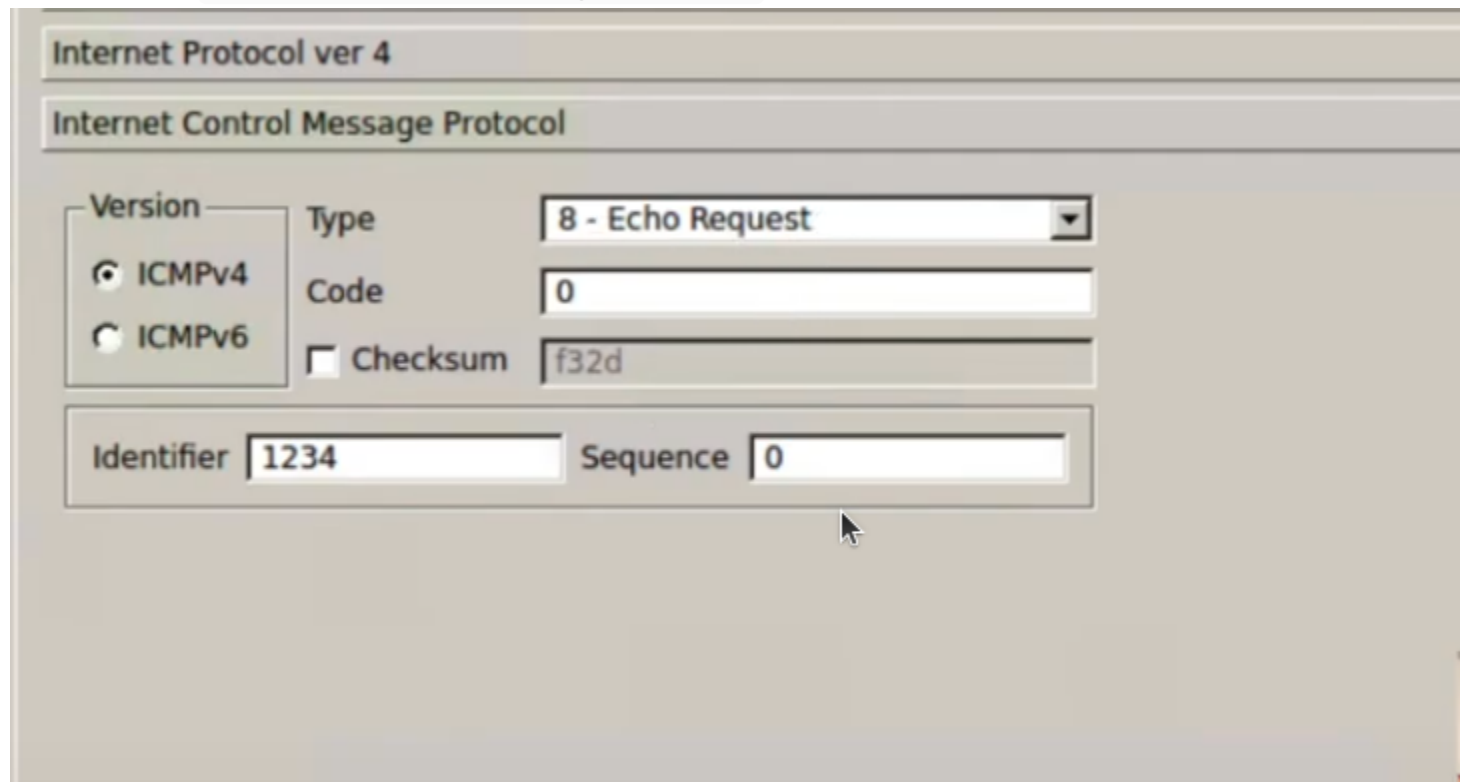
☐ Override Protocol: **11**

☐ Override Checksum: **3B A8**

|                                   | Mode         | Count     | Mask                 |
|-----------------------------------|--------------|-----------|----------------------|
| Source: <b>192.168.61.193</b>     | <b>Fixed</b> | <b>16</b> | <b>255.255.255.0</b> |
| Destination: <b>192.168.60.51</b> | <b>Fixed</b> | <b>16</b> | <b>255.255.255.0</b> |

Options: **TODO**

Passiamo all' internet control message protocol

The image shows the 'Internet Control Message Protocol' configuration window in Wireshark. It is titled 'Internet Protocol ver 4' and 'Internet Control Message Protocol'. Under the 'Version' section, 'ICMPv4' is selected with a radio button. The 'Type' dropdown menu is set to '8 - Echo Request'. The 'Code' field is set to '0'. The 'Checksum' checkbox is unchecked, and the field next to it contains 'f32d'. At the bottom, the 'Identifier' field is set to '1234' and the 'Sequence' field is set to '0'. A mouse cursor is visible over the 'Sequence' field.

Passando in alto a destra in `packet view` c'è la possibilità di vedere tutto il pacchetto costruito

Passando in alto a destra in `stream control` si può vedere il flusso dello stream.

Tornando alla schermata principale abbiamo a sinistra le interfacce, a destra `apply` per applicare il pacchetto, sotto la porta da scegliere:

1. `stop the stream`
2. `clear selected port stats`
3. `start stream`
4. `start capture`
5. `stop capture`
6. `view capture`

In questo modo parte wireshark e si vede la cattura

Per fare spoofing non mettiamo mac della sorgente e come ip mettiamo un indirizzo falso e uscirà su wireshark

## Endianess

Big Endian -> byte più significativo all'indirizzo più basso

Little Endian -> byte più significativo all'indirizzo più alto

Si segue il network order che è il big endian

Si usano le quattro funzioni per convertire da host order a network order

## Checksum

Per l'IP header il checksum è calcolato dal sistema operativo e messo in automatico, mentre per gli altri protocolli bisogna calcolare il checksum

VEDERE CODICE **checksum.c**

La prima funzione calcola il checksum di un generico buffer.

Qui nella prima funzione succede che:

abbiamo un accumulatore `sum` che accumula sequenze di parole di 16 bit che vengono messe tutte insieme; alla fine del processo "fold back" tutti i bit messi dentro dai 16 bit più significativi ai 16 bit meno significativi, cioè se c'è qualche bit generato durante l'addizione delle parole da 16 bit, allora vengono spostati verso i 16 bit meno significativi dei 32 totali in modo che tutti entrino nell'accumulatore ->

`(sum & 0xffff)` prima questo per arrivare ai bit meno significativi

`(sum >> 16)` dopo right-shifting `sum` dai 16 bit per arrivare a quelli più significativi

li sommiamo per riportare di conseguenza i bit all'interno tutti verso i bit meno significativi

`sum += (sum >> 16);` così arriviamo alla parte più significativa

e ritorniamo il complementare della `sum`

Per il protocollo TCP e UDP, il checksum è il complementare del complementare della somma dell'header, in questo caso un fake header.

La seconda funzione infatti usa la prima una volta riempito l'header fake per calcolare il checksum del buffer

## ARP spoofing, ARP cache poisoning, ARP poison routing

### Significato:

#### 1. ARP request ->

In una stessa lan quando un pc A vuole comunicare con B, A deve prima determinare l'indirizzo MAC corretto per poter accedere all'indirizzo IP del computer B. Così opera l'Address Resolution Protocol (ARP), un protocollo di rete che utilizza un modello di richiesta-risposta. Una prima richiesta di trasmissione spesso nota come "**richiesta ARP**", viene inviata in broadcast ad ogni dispositivo nella rete.

Dettagli inclusi in questa richiesta: "l'indirizzo MAC corretto è necessario affinché un computer con l'indirizzo MAC xx-xx-xx-xx-xx-xx e l'indirizzo IP yyy.yyy.yyy.yyy comunichi con un computer con l'indirizzo IP zzz.zzz.zzz.zzz".

Tutti i computer della LAN accettano la richiesta ARP.

Per evitare la necessità d'inviare una richiesta ARP prima dell'invio di ogni pacchetto, ogni computer della rete è connesso a un database locale noto come **cache ARP**. Qui, insieme all'IP allocato, vengono mantenuti brevemente tutti gli indirizzi MAC conosciuti.

Di conseguenza, nella **richiesta di trasmissione**, ogni computer della rete annota la coppia d'indirizzi del mittente consegnato. Tuttavia, solo la macchina B è necessaria per fornire una risposta broadcast e lo fa inviando una risposta ARP con i seguenti dati:

"qui si trova il computer con l'indirizzo IP IPzzz.zzz.zzz.zzz; l'indirizzo MAC desiderato è aa-aa-aa-aa-aa".

### ARP Spoofing

Attacco in cui vengono consegnati **pacchetti ARP falsi** a una LAN.

L'**ARP Spoofing** attualmente viene spesso utilizzato per **trasmettere pacchetti ARP falsi o falsificati**. Il suo scopo principale da allora è stato quello di collegare l'indirizzo MAC dell'hacker all'indirizzo IP dell'altro nodo preso di mira, impedendo in primo luogo alle informazioni di raggiungere la sua destinazione effettiva. A seconda del tipo di attacco, invece, il criminale sarà colui che inoltra il traffico al gateway predefinito o ne modifica i contenuti.

2. ARP cache poisoning -> intercettare la comunicazione tra dispositivi di rete corrompendo le cache della rete. L'autore dell'attacco invia un falso messaggio di risposta ARP al gateway di rete predefinito, informando che l'indirizzo MAC è associato all'indirizzo IP di un'altra destinazione. Quando il gateway predefinito riceve questo messaggio e trasmette le modifiche a tutti gli altri dispositivi della rete, tutto il traffico della destinazione verso qualsiasi altro dispositivo di rete passa attraverso il computer dell'autore dell'attacco. Questa azione consente all'autore dell'attacco di ispezionare o modificare il traffico prima di inoltrarlo alla destinazione prevista.

## Man in the Middle Attack simulato con docker

Consideriamo tre container:

1. Bob: ospita un server http
2. Alice: è un contenitore con Firefox in esecuzione su di esso. Per connettersi a firefox dall'host, visitare `http://localhost:5800`.
3. Eve: è un contenitore pensato per essere utilizzato tramite bash. Per eseguire i comandi, basta eseguire `docker exec -it mitm_eve /bin/bash`. Questo contenitore ha la cartella `eve_files` montata sul contenitore come `/olicyber` (TODO: cambia il nome di questa cartella)

### Docker file

```
trap: SIG TERM; sleep infinity and wait
```

Questo comando crea un processo in background che dorme indefinitamente (cioè non termina mai) e attende che gli vengano inviati segnali. Nello specifico, intrappola i segnali "TERM" e "INT" (che vengono generalmente inviati dal sistema operativo per richiedere la conclusione di un processo) e non consente loro di terminare il processo. Il comando "sleep infinity" istruisce semplicemente il processo a dormire indefinitamente, finché non viene ricevuto un segnale.

Il comando "wait" viene utilizzato qui per garantire che la shell corrente (ovvero la shell che ha eseguito il comando precedente) attenda il termine del processo in background. Ciò è utile nei casi in cui si prevede che il processo in background venga eseguito per molto tempo e la shell deve attendere che finisca prima di procedere con altre attività.

Nel complesso, questo comando crea un processo in background che sembra essere in esecuzione a tempo indeterminato, ma può essere terminato inviando un segnale diverso da "TERM" o "INT". Il processo può essere terminato trovando il suo PID (process ID) usando il comando "ps" e quindi inviando un segnale a quel PID usando il comando "kill".

### Docker Compose:

E' necessario settare il proxy trasparente, in questo modo il traffico viene indirizzato ad un proxy senza nessuna richiesta di configurazione del client; questo rende il proxy trasparente utile per quelle situazioni in cui non è possibile modificare il comportamento del client.

Per configurare il proxy trasparente, abbiamo bisogno di due nuovi componenti:

1. un meccanismo di reindirizzamento che reindirizza in modo trasparente una connessione TCP destinata a un server su Internet a un server proxy in ascolto. Questo di solito assume la forma di un firewall sullo stesso host del server proxy: iptables su Linux o pf su OSX. Quando il proxy riceve una connessione reindirizzata, vede una richiesta HTTP vanilla, senza una specifica dell'host, e qui entra in gioco il secondo elemento.

2. un modulo host che ci consente di interrogare il redirector per la destinazione originale della connessione TCP.

Quindi le fasi sono:

3. Abilitare IP forwarding

Questo assicura che la tua macchina inoltri i pacchetti invece di rifiutarli.

```
sysctl:
```

```
- net.ipv4.ip_forward=1  
- net.ipv6.conf.all.forwarding=1
```

2. Disabilitare indirizzamenti ICMP

Se il tuo dispositivo di prova si trova sulla stessa rete fisica, la tua macchina non dovrebbe informare il dispositivo che è disponibile un percorso più breve saltando il proxy.

```
sysctl:
```

```
- net.ipv4.conf.all.send_redirects=0
```

### add\_iptables\_rule.sh

3. Creare una iptables rule che reindirizza il traffico desiderato al proxy:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
```

Questo comando è una regola iptables che imposta il port forwarding, consente di specificare in che modo il traffico di rete viene diretto al proprio computer e come infine raggiunge il posto giusto e ciò avviene, in primo luogo, comunicando al proprio firewall di rete o router come indirizzare i dati di rete specifici al giusto indirizzo IP e alla giusta porta su tale indirizzo IP, dalla porta 80 alla porta 8080 per il traffico TCP in entrata sull'interfaccia eth0.

1. -t nat: questa opzione specifica la tabella da usare in iptables, che in questo caso è la tabella NAT (Network Address Translation). La tabella NAT viene utilizzata per modificare le informazioni sull'indirizzo di rete nell'instanziazione IP dei pacchetti in transito.
2. -A PREROUTING: Questa opzione aggiunge una nuova regola alla catena PREROUTING nella tabella NAT. La catena PREROUTING viene utilizzata per modificare i pacchetti non appena arrivano su un'interfaccia.
3. -i eth0: questa opzione specifica l'interfaccia di rete in entrata a cui si applica la regola. In questo caso, la regola si applica ai pacchetti che arrivano sull'interfaccia eth0.
4. -p tcp: questa opzione specifica il protocollo da abbinare, che in questo caso è TCP.
5. --dport 80: questa opzione specifica la porta di destinazione da abbinare, che in questo caso è la porta 80.
6. -j REDIRECT: questa opzione specifica la destinazione a cui saltare se la regola corrisponde, che in questo caso è la destinazione REDIRECT. La destinazione REDIRECT viene utilizzata per reindirizzare i pacchetti a un socket locale.
7. --to-port 8080: questa opzione specifica la porta a cui reindirizzare i pacchetti, che in questo caso è la porta 8080.

In sintesi, questa regola iptables reindirizza il traffico TCP in entrata sulla porta 80 alla porta 8080 sulla stessa macchina. Viene comunemente utilizzato quando si esegue un server Web sulla porta

8080 invece della porta predefinita 80 e si reindirizzano tutte le richieste in arrivo dalla porta 80 alla porta 8080. Ciò può essere utile, ad esempio, quando si esegue un'applicazione Web che richiede privilegi amministrativi per eseguire il bind alla porta 80, ma può essere eseguito sulla porta 8080 senza privilegi amministrativi.

I tre container sono collegati tra loro con una rete docker bridge chiamata `mitm`

1. Runnare `docker-compose up -d`
2. Connettersi all'istanza Firefox di Alice `http://localhost:5800` e visitare il sito `http://bob/` : mostra il sito web effettivo servito da Bob
3. Puoi anche connetterti ad Alice tramite riga di comando `docker exec -it mitm_alice /bin/sh` e vedere quale indirizzo MAC corrisponde all'indirizzo IP di Bob -> trovare su Alice `ip` e `mac` di Bob con `arp -a | grep bob`, trovare su Eve il suo indirizzo `ip` e `mac` con `ip address show`
4. Apri 2 istanze di bash sul contenitore di Eve (o, equivalentemente, usa `tmux` con due divisioni) ed esegui il comando `dig` per scoprire gli IP di Alice e Bob:

```
dig alice
dig bob
```

5. Con queste informazioni, ora esegui `arpspoof` due volte, una per ogni istanza bash  
`arpspoof` -> `dsniff`: una raccolta di strumenti per il controllo della rete e il test di penetrazione; quello che viene usato è `arpspoof` che facilita l'intercettazione del traffico di rete normalmente non disponibile per un utente malintenzionato; consente di intercettare i pacchetti su una LAN commutata e reindirizza anche i pacchetti da un host di destinazione (o tutti gli host) sulla LAN destinati a un altro host sulla LAN falsificando le risposte ARP -> `arpspoof -i <Network Interface Name> -t <Victim IP> <Router IP>`

Nella prima finestra bash:

```
arpspoof -t <alice_ip> <bob_ip>
```

Nella seconda finestra bash:

```
arpspoof -t <bob_ip> <alice_ip>
```

6. Ora puoi verificare nell'istanza `sh` di Alice che `ip neighbor` mostra che l'IP di Bob è ora associato all'indirizzo MAC di Eve, il che significa che lo spoofing ARP ha avuto successo. In ogni caso, il ricaricamento della pagina mostra ancora il normale sito Web, poiché Eve non sta ancora bloccando alcun pacchetto.
7. Ora esegui lo script `add_iptables_rule.sh` nella cartella `olicyber` dove ci sono i file di Eve. Questo aggiungerà una regola a `iptables` per inoltrare ogni pacchetto con la porta di destinazione 80 al proxy
8. Puoi verificare che il browser di Alice dia un errore quando ricarica la pagina. Questo perché Eve non blocca i pacchetti in `iptables` e li inoltra al proxy. Poiché il proxy non è ancora attivo, i pacchetti vengono semplicemente eliminati.
9. Ora attiviamo il proxy in passive mode:



```
mitmproxy -m transparent
```

10. Ricarica la pagina del browser: la pagina onesta verrà mostrata di nuovo, ma mitmproxy mostrerà che la richiesta è passata attraverso Eve
11. Ora spegniamo il proxy e lo riattiviamo, questa volta con lo script che modifica i contenuti della pagina:

```
mitmproxy -m transparent -s /eve_files/proxy.py
```

12. Ricarica la pagina del browser: l'attaccante ha modificato i contenuti del sito web.
13. Per chiudere tutto usa lo script `del_iptables_rul.sh` nella cartella `eve_files` per rimuovere la regola `iptables` e disattivare le due istanze `arp spoof`

## Sms spoofing

<https://github.com/hybrid-tech/Anony-SMS> -> source

VEDERE CODICE **sms.py**

### Point-to-point messages

Normalmente i messaggi vanno da un mobile al centro servizi SMSC relativo a quell'operatore telefonico che, se il destinatario è nella stessa rete GSM, lo consegna al terminale opportuno, altrimenti verrà consegnato al centro servizi corrispondente a quell'operatore che poi lo rende disponibile sul terminale opportuno.

Pertanto il singolo messaggio viene, in realtà, diviso in due: il messaggio dal terminale al Centro Servizi (SMS-MO, Mobile Originated), e dal Centro Servizi al destinatario (SMS-MT, Mobile Terminated). Scopo del SMSC è, ovviamente, quello di fare lo *store-and-forward* dei messaggi, anche in previsione di un'eventuale irraggiungibilità momentanea del destinatario.

Dato che la connessione tra terminale e Centro Servizi è di tipo connectionless, non si ha garanzia né sull'invio né sulla ricezione dei messaggi SMS. Tuttavia è possibile fare richiesta di una "ricevuta di ritorno", lo *Status Report* del messaggio. In questo modo, una volta che il messaggio è stato correttamente inoltrato al destinatario, viene inviata una segnalazione di "messaggio consegnato" al mittente.

### Cell broadcast service

Il servizio di SMS, normalmente operante da utente a utente, esiste anche in una variante broadcast. Ogni cella ha la possibilità di inviare brevi messaggi a tutti i terminali sotto la sua copertura, su uno specifico *canale* numerato, ciascuno dedicato a un tipo particolare di informazione, quali emergenze, informazioni sul traffico.

## SMS SPOOFING

Mandiamo una richiesta di tipo POST che andrà dal sender all'SMPP gateway.

Gli SMS o *Short Message Service*, di loro, sono alla base di questo funzionamento, e sono un metodo di invio di brevi messaggi mediante smartphone, che possono avere una **lunghezza massima di 160 caratteri** e che, nello specifico, si possono inviare anche mediante *gateway*. Questa tecnologia aiuta a facilitare l'invio di messaggi a più numeri senza impazzire a farlo **con il proprio telefono**, il che è complicato e tipicamente anche limitato.

Si tratta di **server** che sono accessibili con determinate **credenziali**, quindi una username ed una



password oppure, per i programmatori, una coppia di chiavi pubblica e privata (in genere). Ci sono anche gateway di SMS che possono essere gratuiti, ma in genere si tratta di un servizio **a pagamento**, scalabile sul numero di SMS in blocco che si vogliono inviare volta per volta.

I gateway SMS sono accessibili mediante opportune interfacce web e in genere un gateway non è altro se non una “porta” di accesso ad un determinato servizio, che è possibile programmare in almeno due modi:

- per **integrarlo** mediante codice ed opportune API dentro una nostra app (ad esempio quella che invia i messaggi di posta della newsletter, oppure il nostro sito web aziendale);
- per utilizzarlo mediante **funzionalità predefinite** per l’invio massivo di messaggi.

Ogni messaggio inviato mediante gateway di SMS possiede un contenuto, una data ed una “garanzia” di consegna, che corrisponde anche alla “qualità ” dell’invio.

## I PASSAGGI

I passaggi singoli per inviare ogni singolo SMS partono dall’alto, dalla *web based application* da cui si dipartono, in prima istanza, i messaggi che stiamo inviando e vengono così inoltrati all’ **SMS Gateway**. A questo livello, ricordiamo, è incluso anche un database dei numeri a cui inviare il messaggio, che deve essere cura dell’operatore inserire preventivamente oppure guadagnarsi mediante opportune campagne di acquisizione.

Sfruttando un opportuno protocollo, che è diverso da **HTTP** (usato solo nel primo passaggio) e che viene indicato come **SMS Protocol**, il messaggio viene inoltrato mediante internet ad un **Mobile Switching Center**, che è il centro di smistamento degli SMS. Esso effettua finalmente la “magia”: ovvero smistare gli SMS ai vari numeri destinazione mediante una **base station** adibita allo scopo. I messaggi inviati, entro pochi secondi, arrivano a destinazione.

**L’invio quindi si basa su un mix tra la rete internet e la rete GSM**, che cooperano tra di loro per garantire la consegna del messaggio e, in alcuni casi, per inviare un messaggio di notifica in caso di errore. Gestire questa catena di passaggi non è per nulla semplice, di suo, ed è proprio quello che fanno i gateway: offrono un’infrastruttura già pronta all’uso, che può essere programmata e usata da qualsiasi utente mediante opportune **API** che assicurano che:

- il messaggio non sia vuoto;
- il messaggio provenga da un utente autorizzato all’invio;
- il messaggio pervenga correttamente a destinazione;
- il sistema generi un errore che faccia capire, in caso di errori, quello che non è andato bene.