Sniffing and Spoofing pt. 1

Introduzione

Sniffing attacks: attaccanti possono "origliare" sulla rete fisica, cablata o wireless e catturano i pacchetti trasmessi sulla rete.
Spoofing attacks: attaccanti mandano pacchetti sotto falsa identità

How to receive packets in a normally way

1. Network Interface Card: lcollegamento fisico o logico tra macchine e network attraverso un MAC Address

```
sudo lshw -C network
inxi -N
```

Data Flow -> Ethernet and Wifi broadcast medium -> NIC's computer:

1. copiare il frame nella memoria di NIC
2. controllare l'indirizzo di destinazione nell'header
3. se matcha con l'indirizzo MAC della macchina -> Direct Memory Access (DMA) -> copiare nel Ring Buffer del kernel
4. la CPU viene interrotta dalla Card per informare della disponibilità di un nuovo pacchetto
5. copia il pacchetto in una coda e crea una "stanza" per nuovi pacchetti eventuali
6. In base ai protocolli le funzioni di callback processano i dati dalla coda e spedisce i pacchetti negli user-space programs

Wireshark ping 8.8.8.8

Operazioni preliminari

```
sudo dpkg-reconfigure wireshark-common
sudo chmod +x /usr/bin/dumpcap
```

Scegliere l'interfaccia di rete wifi o ethernet
Mandare `ping 8.8.8.8` da terminale
Applicare su WireShark il filtro `ICMP`

Promiscuous mode

Può succedere che i pacchetti non siano destinati a quella determinata NIC;
Allora in questo caso i pacchetti verranno automaticamente scartati senza passare ad essere processati dalla CPU;
In questo modo non è possibile fare lo sniffing di quei determinati pacchetti;
Entra in gioco la modalità `Promiscuous` dove la NIC passa ogni frame al kernel senza controllare il match tra gli indirizzi MAC.

Se lo sniffer è registrato con il kernel allora i pacchetti vanno dal kernel allo sniffer e vengono intercettati comunque.

```
sudo ifconfig <interfaccia> promisc #impostare la modalità promiscua
sudo ifconfig <interfaccia> -promisc #disabilitare la modalità promiscua
```

Monitor mode

Questo è una modalità che cerca di risolvere il problema degli wireless device;
Il problema sta nelle interferenze che ci possono essere quando si trasmettono dei pacchetti;
Perciò i wifi device trasmettono i pacchetti su canali diversi e così gli access point si connettono ai device tramite canali differenti.
Le wifi network cards quando stanno in monitor mode, riescono a catturare i pacchetti di tipo 802.11 sui vari differenti canali su cui sono in ascolto -> possibilità di perdere delle info perchè si tratta di canali differenti.

BSD Packet Filter

Questo è un filtro che può essere attaccato al socket e dice al kernel quali pacchetti scartare o meno -> in questo modo si va a risparmiare tempo
Il filtro viene:

- scritto in un formato leggibile all'uomo attraverso operatori booleani
- compilato in uno pseudo-codice
- passato al driver BPF
- interpretato da una BPF pseudo-machine che è una macchina a livello del kernel disegnata per i filtri di pacchetti

Come viene attaccato al socket? Funzione
`int setsockopt(int *socket*, int *level*, int *option_name*, const void **option_value*, socklen*t _option_len*)
-> level è il livello di protocollo: a livello di socket è SOL_SOCKET

```
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf))
```

Sniffing

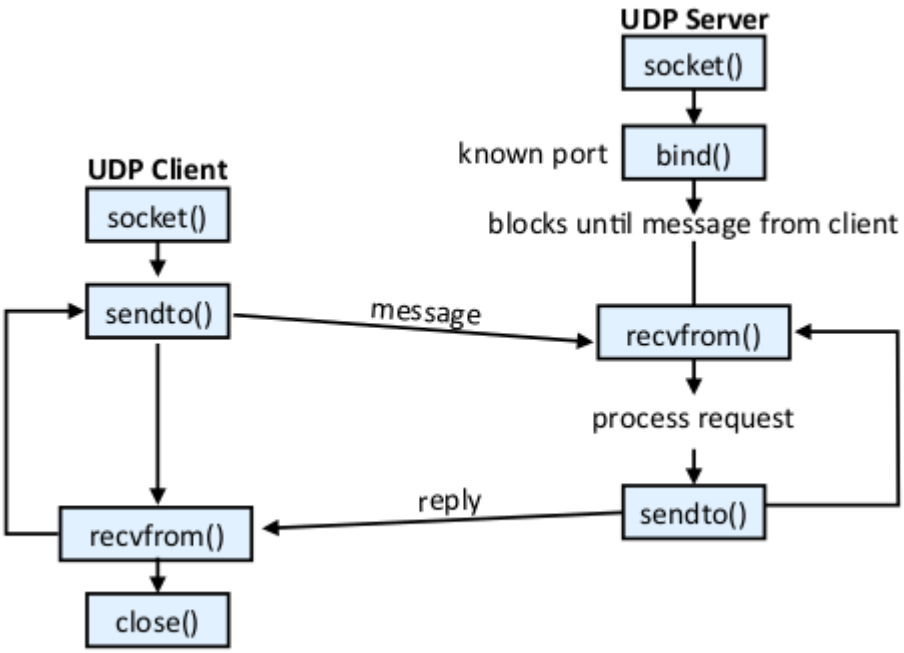
Sniffing: catturare live data che fluiscono nella rete

- amministratori: capire come funziona la rete e fare diagnosi della rete stessa
- attaccanti: fare reconnaissance -> l'**active reconnaissance** e la **passive reconnaissance**. Nel primo caso, l'intruso interagisce con la rete utilizzando tool di port scanning per individuare eventuali porte vulnerabili. Nella passive reconnaissance si ricercano vulnerabilità solo con l'analisi di traffici di dati intercettati con metodi quali la **session capture**, che consente di raccogliere enormi moli di dati grezzi; fare exploitation -> gli exploit sono software progettati per sfruttare le falle di un sistema informatico.

inviare pacchetti normali

CODICE **udp_server.c** -> come i programmi normalmente ricevono i pacchetti

1. creare il socket
2. bind per far conoscere le info sul server e in questo modo i pacchetti indirizzati su quell'indirizzo verrano inviati lì -> un computer può avere più indirizzi IP, uno per ogni network interface card, allora impostiamo INADDR_ANY e così abilitiamo a tutti gli indirizzi IP validi
3. una volta settato il codice il server usa la funzione `recvfrom` per ricevere i pacchetti



Wireshark

VEDERE **UDPServer**.c e **UDPclient.c**
Si imposta l'interfaccia `loopback`
Si imposta come filtro `udp.port = 49152`

inviare pacchetti con raw socket

Il codice precedente funziona solo se si inviano dei pacchetti a quell'indirizzo specifico
Uno sniffer programm deve catturare anche i pacchetti provenienti sulla rete non indirizzati prettamente a noi e per farlo usa il raw socket ovvero un socket grezzo.
Il socket è la coppia indirizzo ip + porta che permette lo scambio di dati in una rete.
Il raw socket è un tipo di socket di rete che consente a un'applicazione software sul computer di inviare e ottenere pacchetti di informazioni dalla rete senza utilizzare il sistema operativo del computer come intermediario; consente la comunicazione diretta tra un programma e una fonte esterna senza l'intervento del sistema operativo primario del computer;
i raw socket sono utilizzati per l'accesso diretto ai protocolli di rete sottostanti. Un socket raw bypassa la normale elaborazione TCP/IP e consente l'accesso diretto al livello di rete. I raw socket sono tipicamente utilizzati dai programmi di monitoraggio della rete e dai router.

Vantaggi: CPU molto meno "affaticata" perchè non c'è il peso dell'intermediario
Svantaggi: durante un attacco TCP, un hacker invia un bit di dati contraffatto alla rete tramite una connessione socket non elaborata. Questi dati contraffatti contengono un segnale di ripristino per la connessione TCP, che a sua volta interrompe e blocca le connessioni di rete correnti sul computer. Per questo motivo, alcuni sistemi operativi hanno ritirato il supporto per i socket non elaborati. La ragione di ciò è che può aiutare a garantire la sicurezza della rete.

VEDERE CODICE **sniff_raw.c**:

Da ricordare:
1. Creare il socket e settare il protocollo `int socket(int _domain_, int _type_, int _protocol_);` * Domain: AF_UNSPEC, AF_INET, AF_INET6, AF_UNIX * Type: tipo di servizio da fornire all'applicazione SOCK_DGRAM, SOCK_STREAM * Protocol: protocollo di comunicazione IPPROTO_UDP, IPPROTO_UDP

AF_PACKET= low level packet interface
RAW_SOCKET= socket grezzo
ETH_P_ALL= all protocols are received

- Quando i pacchetti arrivano al kernel, questo passa una copia del pacchetto al socket, prima di passare il pacchetto allo stack protocollare
- `htons()` da host byte order (little endian) a network byte order (big endian)

2. Impostare la promiscuous mode
   In questo modo tutti i pacchetti che il nostro computer può osservare sulla rete verranno intercettati e una volta che la NIC li prende e li passa al kernel, l'opzione di raw socket attivata ne farà una copia.

- PACKET_ADD_MEMBERSHIP: i socket di pacchetto possono essere usati per configurare il livello fisico modalità multicasting e promiscua; questo aggiunge un'associazione.

3. Aspettare i pacchetti con `recvfrom()`

Filtri e PCAP

1. opzione in `setsockopt()`: `SO_ATTACH_FILTER` come `option_name`
   VEDERE CODICE: **raw_filter.c**

```
gcc -o raw_filter raw_filter.c
sudo ./raw_filter <nome interfaccia>
```

2. Pcap API è stata creata per avere un'interfaccia indipendente dalla piattaforma per accedere in modo efficiente alla funzione di acquisizione dei pacchetti del sistema operativo; è un compiler che permette ai programmatori di utilizzare delle regole di filtraggio attraverso espressioni booleane. La pcap API è implementata come `libpcap` sui dispositivi Unix. tcpdump

Sniffing with pcap API

Queste API sono scritte nel linguaggio di programmazione C ed offrono la possibilità di:

- Catturare i pacchetti che circolano sulla nostra rete;
- Filtrare i pacchetti catturati a seconda di vari parametri;
- Salvare i pacchetti su un file;

filtri pcap e wireshark

VEDERE CODICE **sniff.c**

Da ricordare:

1. Aprire una `pcap` session: settare l'interfaccia in promiscuous mode e fare il bind del socket -> man page `pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf);` = interfaccia (`any o NULL`), lunghezza snapshot (copia del file del disco), promiscuous mode, timeout del pacchetto, buffer di errore
2. Settare i filtri: il filtro BPF è scritto in un linguaggio di basso livello capibile dalle macchine; con due funzioni imponiamo un compiler che compila l'espressione specifica e poi settiamo il filtro: `int pcap_compile(handle, &fp, filter_exp, 0, net);` = handle punta al fp del pacchetto catturato, fp punta ad una bpf_struct che verrà riempita dopo, exp è la stringa del filtro, 0 si riferisce all'impostare il controllo se viene eseguita l'ottimizzazione sul codice risultante, net è la netmask, ovvero la netmask che consente di stabilire l'intervallo di indirizzi IP all'interno di una sottorete e quindi dice a quale sottorete apparteniamo; `int pcap_setfilter(handle, &fp);` = handle è l'fp del pacchetto catturato, fp punta alla struttura bpf_struct.
3. Catturare i pacchetti: `int pcap_loop(handle, -1, got_packet, NULL);` = handle punta al fp del pacchetto, -1 è il numero di pacchetti da catturare cioè senza fine con quel valore, funzione di callback che fa delle cose ogni volta che viene catturato un pacchetto, NULL specifica il primo argomento da passare alla routine di callback.
4. Compilazione:

```
sudo apt-get install git libpcap-dev
gcc -o sniff sniff.c -lpcap
sudo ./sniff

#da un altro terminale con icmp[icmptype]=0
ping 8.8.8.8

#https://www.tcpdump.org/manpages/pcap-filter.7.html
```

Processo di cattura dei pacchetti

Nell'esempio precedente se il pacchetto era catturato, viene stampato solo il messaggio.
Ora stampiamo delle info sull'header del pacchetto a partire dalla funzione di callback che passiamo nel `pcap_loop`.
La funzione di callback ha per default dei parametri tra un `u_char *packet` che punta direttamente al pacchetto catturato.
Per stampare delle info sull'header non usiamo l'offset perchè a seconda del tipo di pacchetto l'offset potrebbe cambiare e quindi usiamo un cast da `u_char` a `struct` in modo da accedere facilmente ai campi che impostiamo dell'header dell'interfaccia:

```c
/* Ethernet header */
struct ethheader {
  u_char  ether_dhost[6]; /* destination host address */
  u_char  ether_shost[6]; /* source host address */
  u_short ether_type;     /* protocol type (IP, ARP, RARP, etc) */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;          ①
  if (ntohs(eth->ether_type) == 0x0800) { ... } // IP packet   ②
  ...
}
```

Se vogliamo più informazioni dall'IP header allora ci spostiamo dall'

inizio del pacchetto + la dimensione dell'header dell'interfaccia
Se vogliamo più informazioni dall'header del protocollo allora ci spostiamo dall'
inizio del pacchetto + la dimensione dell'header dell'interfaccia + lunghezza dell'IP header
perchè alcuni IP header possono contenere campi opzionali e quindi non possiamo agire con l'offset.

VEDERE CODICE **sniff_improved.c**

Comandi per fare sniffing e Wireshark

Sito per esempi: tcpdump

tcpdump

-> risultati:

- pacchetti catturati -> questo è il numero di pacchetti che tcpdump ha ricevuto ed elaborato;
- pacchetti ricevuti dal filtro -> il significato di questo dipende dal sistema operativo su cui stai eseguendo tcpdump, e possibilmente dal modo in cui il sistema operativo è stato configurato: se è stato specificato un filtro sulla riga di comando, su alcuni sistemi operativi conta pacchetti indipendentemente dal fatto che siano stati abbinati dall'espressione del filtro e, anche se sono stati abbinati dall'espressione del filtro, indipendentemente dal fatto che tcpdump li abbia già letti ed elaborati, su altri sistemi operativi conta solo i pacchetti che sono stati abbinati dall'espressione del filtro indipendentemente da se tcpdump li ha già letti ed elaborati e su altri sistemi operativi conta solo i pacchetti che sono stati abbinati dall'espressione del filtro e sono stati elaborati da tcpdump;
- pacchetti eliminati dal kernel -> questo è il numero di pacchetti che sono stati eliminati, a causa della mancanza di spazio nel buffer, dal meccanismo di acquisizione dei pacchetti nel sistema operativo su cui tcpdump è in esecuzione, se il sistema operativo riporta tali informazioni alle applicazioni; in caso contrario, verrà riportato come 0
  Con SIGINFO (CTRL+T) o SIGUSR1 si vede lo stato.

```
sudo tcpdump host <indirizzo ip>
sudo tcpdump host <indirizzo ip> and not <indirizzo ip>
#Per stampare il traffico né proveniente da né destinato a host locali
```

```
sudo tcpdump 'ip and not net 127.0.0.1'
#Per stampare i pacchetti di inizio e di fine (i pacchetti SYN e FIN) di ogni
conversazione TCP che coinvolge un host non locale.
sudo tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net
127.0.0.1'
#not ping packet
sudo tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'
```

ping

-> mandare pacchetti icmp ad un destinatario

traceroute

-> i pacchetti si muovono, all'interno delle reti informatiche, attraverso dei percorsi che prendono il nome di traceroute, con lo stesso termine vengono indicate anche le applicazioni che permettono di risalire i vari passaggi di ogni percorso, dove le tappe del percorso prendono il nome di nodi.
Importante è il campo TTL che è il time to live di un pacchetto, ovvero il numero massimo di dispositivi che un package di dati può attraversare prima della sua scadenza. Un diagramma del campo TTL registra, in pratica, l'andamento del valore numerico di TTL: definito come *n* il numero massimo di network device attraversabili da un pacchetto di dati, ogni passaggio attraverso un router determinerà un decremento unitario (*n-1*) e nel momento in cui tale valore sarà pari a 0 il Traceroute invierà, al terminale che ha formulato la richiesta iniziale, una notifica di errore ICMP (*Internet Control Message Protocol*) che è il protocollo incaricato di trasmettere le informazioni relative ai malfunzionamenti che si verificano in una rete. Nel messaggio sarà presente anche il dato riguardante l'indirizzo del router che ha determinato l'errore stesso. Lo scopo di traceroute è seguire il percorso di un pacchetto di dati dalla sua sorgente fino a destinazione e per farlo dovrà registrare ogni singolo salto del pacchetto.
In fase di esecuzione un Traceroute invia una serie di pacchetti di dati generando una modifica del valore **Max_ttl** che viene, inizialmente, impostato su 1 e incrementato via via di una unità sino a quando il pacchetto non giunge a destinazione. Lo scopo è di ottenere da ciascun nodo del percorso una notifica ICMP, "*Time Exceeded*" in cui vengano restituiti il periodo di tempo passato tra l'invio del package e la sua ricezione, in millisecondi, e l'indirizzo IP del router contattato.
Al primo package viene attribuito un valore Max*ttl pari ad un 1 hop, il record risultante è facilmente riconoscibile perché riporta generalmente l'indirizzo IP del router associato al terminale che ha lanciato l'istanza di tracerouting.*
Ricevuto il primo package (Max_TT = 1) il nodo decrementa il valore di uno ottenendo zero ed inviando, pertanto, la prima notifica ICMP.
Si susseguono poi gli ulteriori package con valore Max_ttl crescente su base unitaria (2 hop, 3 hop.. _n hop) e tale incremento prosegue fino al massimo valore consentito di hop (ad esempio non più di 30 punti di passaggio), fino all'arrivo a destinazione del package (notifica ICMP "*Echo Reply*") o fino a quando l'ultimo gateway non restituisce il messaggio "*port_unreachable*" testimoniando che il package ricevuto non può essere più utilizzato in alcun modo. Al termine dell'analisi il sistema sarà in grado di restituire informazioni dettagliate su ciascun nodo coinvolto nel transito del pacchetto dati. Come output, traceroute mostra il percorso seguito dal segnale mentre viaggiava su Internet fino al sito web e visualizza anche i tempi che sono i tempi di risposta che si sono verificati ad ogni fermata lungo il percorso.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install inetutils-traceroute
sudo traceroute www.google.it
#traceroute lavora su UDP, è possibile forzare il tracciamento tramite ICMP
utilizzando l'opzione “-I”
sudo traceroute -I www.google.it
```

ifconfig

-> visualizzare, attivare, disattivare le interfacce di network

netstat

-> visualizzare diverse info sulla rete: connessioni di rete, tabelle di instradamento, connessioni mascherate, statistiche dell’interfaccia, appartenenza multicast ...

```
#Per visualizzare le prese di ascolto e non di ascolto dell’uso della rete
netstat -a | more
#Per visualizzare solo le porte TCP
netstat -at
#Per viusalizzare solo le porte UDP
netstat -au
#Per visualizzare connessioni con porte di ascolto attive
netstat -l
#Per visualizzare connessioni TCP con porte di ascolto attive
netstat -lt
#Per visualizzare connessioni UDP con porte di ascolto attive
netstat -lu
#Per visualizzare connessioni UNIX con porte di ascolto attive
netstat -lx
#Per visualizzare le statistiche di tutti i protocolli
netstat -s
#Per visualizzare le statistiche del protocollo TCP
netstat -st
#Per visualizzare le statistiche del protocollo UDP
netstat -su
#Permette di visualizzare anche PID/nome del programma
netstat -tp
#Stampa le info net ogni tot secondi
netstat -c
#Info sulle tabelle di routing
netstat -r
#Info sulle transazioni dell'interfaccia di rete
netstat -i
netstat -ie #simile a ifconfig
netstat -g #interfaccia + appartenenza al gruppo ipv4 o ipv6
netstat -ap | grep http #visualizzare programmi in ascolto su una porta -> in
questo caso con un grep di http
netstat --statistic --raw #visualizzare statistiche di rete non elaborate
```

lsof

->viene utilizzato per scoprire quali file sono aperti da quale processo sul pc che lancia il comando

```
lsof
lsof -i TCP:22 #processi aperti sulla porta 22 con protocollo TCP
lsof -i TCP
lsof -i 4 #file di rete IPv4
lsof -i 6 #file di rete IPv6
lsof -i -u^root #abbiamo escluso l'utente root
lsof -i #listening and established connection
```

tshark

opzioni:

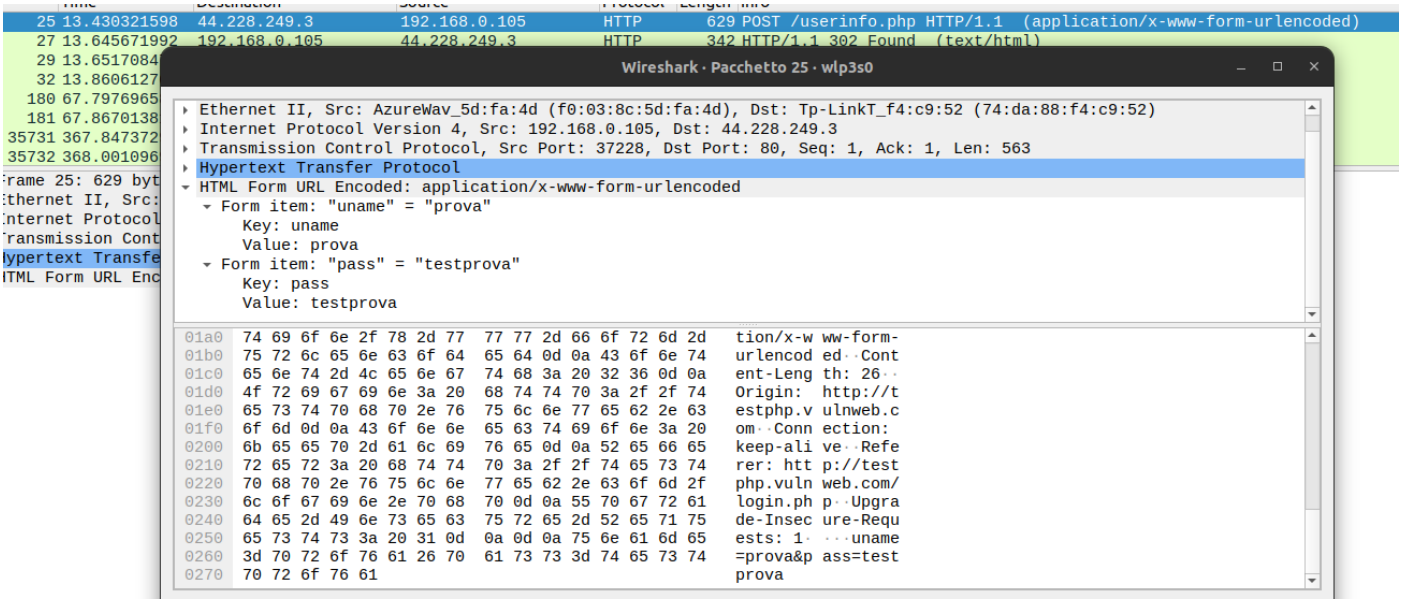
- -a -> serve per indicare se stoppare lo sniffing al verificarsi di una condizione
- -c -> il massimo numero di pacchetti da catturare
- -f -> per impostare dei filtri nel momento della cattura

```
#catturare pacchetti in un file pcap e leggerlo
tshark -i wlp3s0 -w capture-output.pcap
tshark -r capture-output.pcap

#HTTP Analysis with Tshark
#-T specifica che vogliamo estrarre campi e -e specifica i campi che vogliamo
estrarre, -Y specifica i filtri da applicare
tshark -i wlp3s0 -Y http.request -T fields -e http.host -e http.user_agent

tshark  -i wlp3s0 -Y http.request -T fields -e http.host -e ip.dst -e
http.request.full_uri

#estrarre query DNS
tshark -i wlp3s0 -f "src port 53" -n -T fields -e dns.qry.name

#estrarre la password da un TCP stream attraverso la HTTP POST
tshark -i wlan0 -Y 'http.request.method == POST and tcp contains "password"' |
grep password

tshark -d tcp.port==8888,http #will decode any traffic running over TCP port 8888
as HTTP.

tshark -d tcp.port==8888:3,http #will decode any traffic running over TCP ports
8888, 8889 or 8890 as HTTP.

tshark -d tcp.port==8888-8890,http #will decode any traffic running over TCP ports
8888, 8889 or 8890 as HTTP.
```

Simulazione sniffing credenziali con Wireshark

Problema HTTP e HTTPS: con HTTP il sito non è sicuro perchè i dati vengono mandati in chiaro:

1. andare su questo sito: http://testphp.vulnweb.com/login.php
2. mettere in ascolto wireshark con il filtro `http`
3. aggiornare la pagina
4. visualizzare su wireshark la richiesta e la risposta
5. inseriamo nome utente e password fittizi sul sito di prova
6. sniffiamo con wireshark la richiesta e vediamo i dati del pacchetto specifico



Sniffing con un fake access point

Usiamo un raspberry per creare un finto access point che si appoggia alla rete wireless.
L'access point è ciò che offre un punto di accesso wireless a chi ne ha bisogno senza avere la possibilità di collegarsi direttamente alla linea, perciò ha la necessità di essere connesso ad un router che invece si occupa di indirizzare i pacchetti; modalità di operare:

- **Bridge** (quando viene creata una connessione punto a punto tra due access point per connettere due reti cablata tra loro)
- **Repeater** (che consente di aumentare la copertura di una rete wireless senza dover usare altri cavi o modem/router)
- **Client** (che permette di far collegare alla rete senza fili anche i dispositivi senza scheda di rete wireless effettuando il collegamento via cavo di questi ultimi all'access point)

1. impostare su raspberry l'ip statico

If you wish to disable automatic configuration for an interface and instead configure it statically, add the details to */etc/dhcpcd.conf*. For example:

```
interface eth0
static ip_address=192.168.0.4/24 (replace 192.168.0.4 with the IP address that you want to assign to your Raspberry Pi)
static routers=192.168.0.254 (replace with your router address)
static domain_name_servers=192.168.0.254   8.8.8.8 (8.8.8.8. to use the Google DNS)
```

(if you want to set the static IP for your "eth0" (Ethernet) connector)

```
interface wlan0
static ip_address=192.168.0.4/24
static routers=192.168.0.254
static domain_name_servers=192.168.0.254   8.8.8.8
```

(if you want to set the static IP for your "wlan0" (WiFi) connection)

2. installare i tool

```
sudo apt-get install hostapd dnsmasq
```

1. HostAPD (Host access point daemon) è un software per la gestione dell'interfaccia wireless e anche un server di autenticazione
2. DNSMasq è un server DHCP che eroga anche servizi di cache DNS.

```
sudo systemctl stop dnsmasq
sudo systemctl stop hostapd
```

2. impostare ip statico del raspberry

```
sudo nano /etc/dhcpcd.conf

#all'interno si va nella configurazione dell'ip statico
interface wlan0
static_ip_address...
```

3. configurazione del DHCP server
   Il servizio è erogato da `dnsmasq`
   Il file originale contiene delle info, bisogna riscrivere da zero un nuovo file, facendo prima un opportuno backup del file originale.

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
sudo nano /etc/dnsmasq.conf

#all'interno
interface=wlan0       # Use the require wireless interface - usually wlan0
dhcp-range=192.168.3.2,192.168.3.20,255.255.255.0,24h

sudo systemctl start dnsmasq
```

Il server dhcp a questo punto è già pronto ad erogare ai client gli indirizzi ip, inclusi nel range compreso tra 192.168.3.2 e 192.168.3.20

4. configurazione dell'hostapd

```
sudo nano /etc/hostapd/hostapd.conf

#all'interno
interface=wlan0
driver=nl80211
ssid=**TestReteWifi**
hw_mode=g #banda di frequenza a 2.4GHz
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
```

```
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=**PasswordDiReteWifi**
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

5. attivare la configurazione

```
sudo nano /etc/default/hostapd

#all'interno
DAEMON_CONF="/etc/hostapd/hostapd.conf"

sudo service dhcpcd restart

sudo systemctl unmask hostapd
sudo systemctl disable hostapd
sudo systemctl enable hostapd
sudo update-rc.d hostapd enable
sudo systemctl start hostapd

sudo systemctl status hostapd
sudo systemctl status dnsmasq
```

6. configurazione del routing

```
sudo nano /etc/sysctl.conf

#all'interno
net.ipv4.ip_forward=1
```

A questo punto il sistema sarà abilitato a ruotare i pacchetti da un'iterfaccia ad un altra, ma ancora non sarà possibile navigare in quanto gli ip dei client appartengono ad una rete privata, quindi sarà necessario mascherarli/nattarli per consentire il traffico in uscita sull'interfaccia eth0, che supponiamo sia connessa ad esempio ad un router abilitato alla navigazione internet. Per nattare la rete verso il traffico su eth0 sarà sufficiente eseguire questo comando.

```
sudo iptables -t nat -A  POSTROUTING -o eth0 -j MASQUERADE

#per rendere le modifiche definitive
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Aggiungiamo al file /etc/rc.local la seguente riga in modo che al reboot della raspberry la configurazione di iptables venga caricata opportunamente.

```
iptables-restore < /etc/iptables.ipv4.nat
```

7. Far partire l'access point

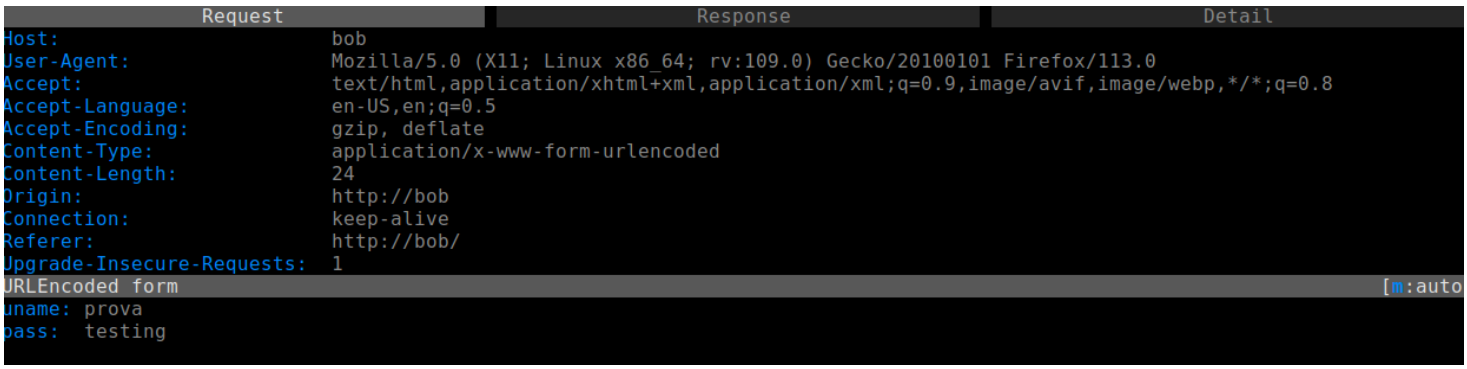
```
sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf #per far partire l'access point
```

MitM Sniffing with mitmproxy

```
docker-compose up -d
```

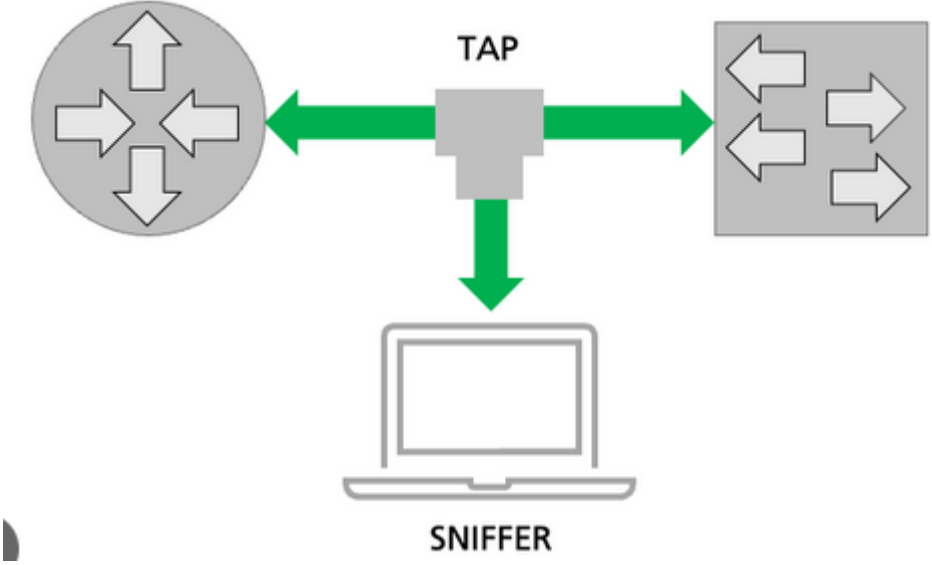
Andiamo su `http://localhost:5800`
Andiamo da lì su `http://bob:8080`

```
docker exec -it alice /bin/sh #X1
docker exec -it eve /bin/sh #X3

#in alice
arp -a

#in eve 1
ip address show
dig alice
dig bob

arpspoof -t <ip alice> <ip bob>

#in eve 2
arpspoof -t <ip alice> <ip bob>

#in alice
arp -a #vedere se il mac address combacia

#in eve 3
cd home/lunaticyber/Scrivania/sniff
./add_iptables_rule.sh
mitmproxy -m transparent

#sul sito alice mette delle credenziali
#su mitmproxy vengono sniffate
```



LAN TAP DEVICE

Effettuare un attacco tramite Fake Access Point è un po' obsoleto al giorno d'oggi. Infatti un nuovo tipo di hardware usato negli attacchi è il Lan Tap Device.



Cos'è? Hardware che permette di fare sniffing, composto tra due porte Ethernet e una porta USB-C; sostanzialmente permette di intercettare le comunicazioni che avvengono tra i due dispositivi connessi attraverso Ethernet grazie al collegamento di un terzo dispositivo tramite la porta USB-C, che può essere un cellulare il quale sfrutta un' app scaricabile monita di permessi di root, che cattura i pacchetti e facendone una copia, crea un file `.pcap`, come quelli creati con Wireshark, intercettando così tutto il traffico tra i due dispositivi.