

# Report progetto finale modulo M1 W4D4

## Indice

<i>Introduzione</i>	<i>Pagina 1</i>
<i>Assegnazione indirizzi IP alle macchine</i>	<i>Pagina 2</i>
<i>Attivazione servizio HTTPS, HTTP e DNS</i>	<i>Pagina 4</i>
<i>Test connettività</i>	<i>Pagina 6</i>
<i>Intercettazione pacchetti con Wireshark</i>	<i>Pagina 7</i>
<i>Considerazioni finali</i>	<i>Pagina 13</i>

## Introduzione

In questo progetto dovremo mettere in comunicazione due macchine virtuali, in modo che una faccia da client (Windows) e l'altra funga da server DNS, HTTP e HTTPS (Kali).

Dovremo poi analizzare lo scambio dei pacchetti tramite Wireshark ed evidenziare le differenze tra i due protocolli utilizzati.

# Assegnazione indirizzi IP alle macchine

Per prima cosa assegneremo gli IP previsti dall'esercizio alle macchine Kali e Windows:

- Kali: 192.168.32.100
- Windows: 192.168.32.101

## Cambio IP macchina Kali

- **Step 1** Accendiamo la macchina virtuale e modifichiamo l'IP e il default gateway come da figura sottostante, tramite il comando `sudo nano /etc/network/interfaces`.

```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.32.100/24
gateway 192.168.32.1
```

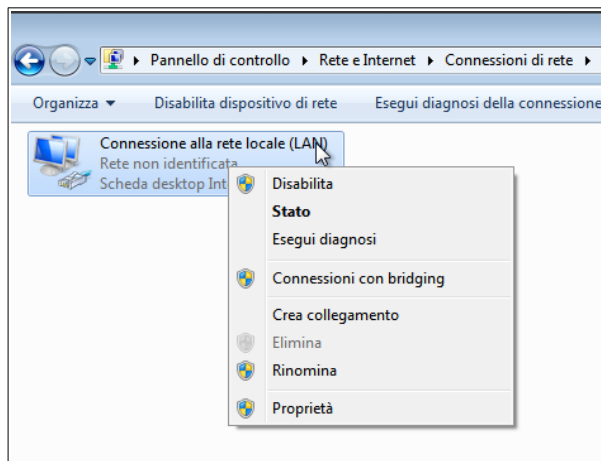
- **Step 2** salviamo le modifiche e riavviamo il network tramite il comando `sudo systemctl restart networking`, verificare che le modifiche siano state applicate correttamente tramite il comando `ifconfig`.

```
(kali@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.32.100 netmask 255.255.255.0 broadcast 192.168.32.255
    inet6 fe80::a00:27ff:fe6e:136e prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:6e:13:6e txqueuelen 1000 (Ethernet)
    RX packets 78 bytes 12910 (12.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 30 bytes 6938 (6.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

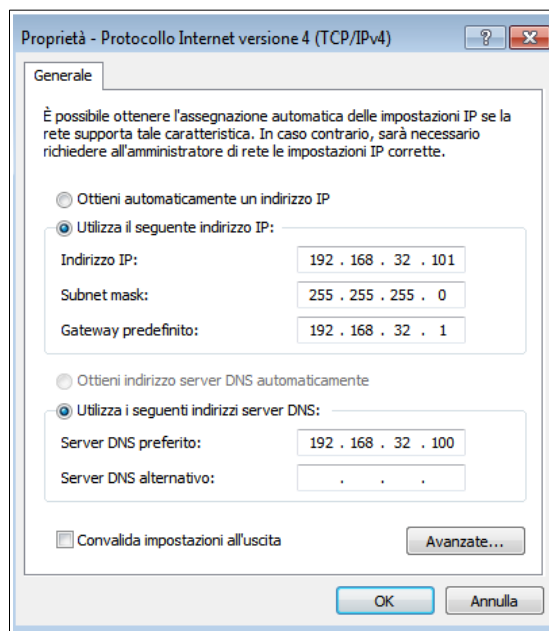
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 480 (480.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 480 (480.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

## Cambio IP della macchina Windows

- **Step 1** Accendere la macchina Windows e da *pannello di controllo/rete e internet/connessioni di rete* cliccare con il tasto destro su *Connessione alla rete locale (LAN)* e selezionare *proprietà*.



- **Step 2** Dalla sezione *proprietà* del protocollo *TCP/IPv4*, modifichiamo i campi come da figura sottostante. Il campo del DNS dovrà essere popolato con l'IP assegnato a Kali in precedenza perché dovrà svolgere la funzione di server DNS.



## Attivazione servizio HTTPS, HTTP e DNS

Per simulare una richiesta DNS come richiesto dalla consegna da macchina Windows a macchina Kali, dovremo attivare tre diversi servizi: HTTPS, HTTP e DNS. Andremo ora nel dettaglio per ogni servizio da attivare.

### Attivazione servizio HTTPS e HTTP

- **Step 1** Da macchina Kali, diamo il comando `sudo nano /etc/inetsim/inetsim.conf`, assicuriamoci che i servizi HTTPS e HTTP non siano commentati e che quindi siano attivi.

```
#start_service dns
start_service http
start_service https
#start_service smtp
#start_service smtps
#start_service pop3
#start_service pop3s
#start_service ftp
#start_service ftps
```

- **Step 2** Spostiamoci qualche riga più in basso e anche qui togliamo il simbolo del `#` davanti alla riga `service_bind_address` e modifichiamola come da figura, in questo modo metteremo inetsim in ascolto sull' IP assegnato a Kali. Infine salviamo le modifiche.

```
#####
# service_bind_address
#
# IP address to bind services to
#
# Syntax: service_bind_address <IP address>
#
# Default: 127.0.0.1
#
service_bind_address 0.0.0.0
```

- **Step 3** Tramite il comando `sudo inetsim`, facciamo partire la simulazione dei servizi selezionati.

```
(kali@kali)-[~]
└─$ sudo inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory: /var/log/inetsim/
Using data directory: /var/lib/inetsim/
Using report directory: /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 19198) ===
Session ID: 19198
Listening on: 0.0.0.0
Real Date/Time: 2025-03-21 16:57:01
Fake Date/Time: 2025-03-21 16:57:01 (Delta: 0 seconds)
Forking services ...
 * http_80_tcp - started (PID 19202)
 * https_443_tcp - started (PID 19203)
done.
Simulation running.
```

## Attivazione servizio DNS

Normalmente potremmo attivare il servizio DNS da *inetsim* con la modalità vista in precedenza, però nella versione corrente di Kali il servizio non è funzionante.

Nel nostro caso potremo ovviare al problema utilizzando *DNSmasq*.

- **Step 1** Impostare la macchina Kali su *scheda con bridge* nelle impostazioni di rete di Virtual Box ed impostare un IP dinamico tramite linea di comando, salvare la modifiche e riavviare il network.

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
address 192.168.32.100/24
gateway 192.168.32.1
```

- **Step 2** Ora la nostra macchina Kali potrà scaricare DNSmasq tramite il comando `sudo apt -y install dnsmasq`, (assicuriamoci prima che il database apt sia aggiornato).

```
(kali@kali)-[~]
$ sudo apt install -y dnsmasq
dnsmasq is already the newest version (2.91-test9-1).
The following packages were automatically installed and are no longer required:
crackmapexec libbft9 libhdf5-103-1t64 libqt5sensors5 python3-ntlm-auth
firebird3.0-common libgdal35 libhdf5-hl-100t64 libqt5webkit5 python3-setproctitle
firebird3.0-common-doc libgeos3.13.0 libjxl0.9 libqt5x11extras5 python3.12
imagemagick-6.q16 libgl1-mesa-dev libldap-2.5-0 libsuperlu6 python3.12-dev
libbfio1 libglapi-mesa libmagiccore-6.q16-7-extra libtag1v5 python3.12-minimal
libc++1-19 libgles-dev libmagiccore-6.q16-7t64 libtag1v5-vanilla python3.12-venv
libc++abi1-19 libgles1 libmagicwand-6.q16-7t64 libtag0 ruby-zeitwerk
libcapstone4 libglvnd-core-dev libmbedcrypto7t64 libunwind-19 ruby3.1
libconfig+9v5 libglvnd-dev libmsgraph-0-1 libwebRTC-audio-processing1 ruby3.1-dev
libconfig9 libgtksourceview-3.0-1 libnetcdf19t64 libx265-209 ruby3.1-doc
libdirectfb-1.7-7t64 libgtksourceview-3.0-common libpaper1 openjdk-23-jre
libegl-dev libgtksourceviewmm-3.0-0v5 libpoppler140 openjdk-23-jre-headless
libflac12t64 libgumbo2 libpython3.12-dev python3-appdirs
Use 'sudo apt autoremove' to remove them.

Summary:
Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

- **Step 3** Rimettere la macchina su *rete interna* in modo che possa comunicare solo con l'ambiente virtuale e modificare l'IP in *statico*.
- **Step 4** Tramite il comando `sudo vim /etc/dnsmasq.d/nomefile.conf` creeremo un file contenente il nostro nome di dominio `epicode.internal` e l'IP di Kali usando la sintassi presente in figura. Salviamo le modifiche tramite il comando `:wq` (write and quit).

```
File Actions Edit View Help
address=/epicode.internal/192.168.32.100
~
~
```

- **Step 5** Riavviare il servizio di DNSmasq tramite il comando `systemctl restart dnsmasq` ed il servizio di server DNS sarà operativo.

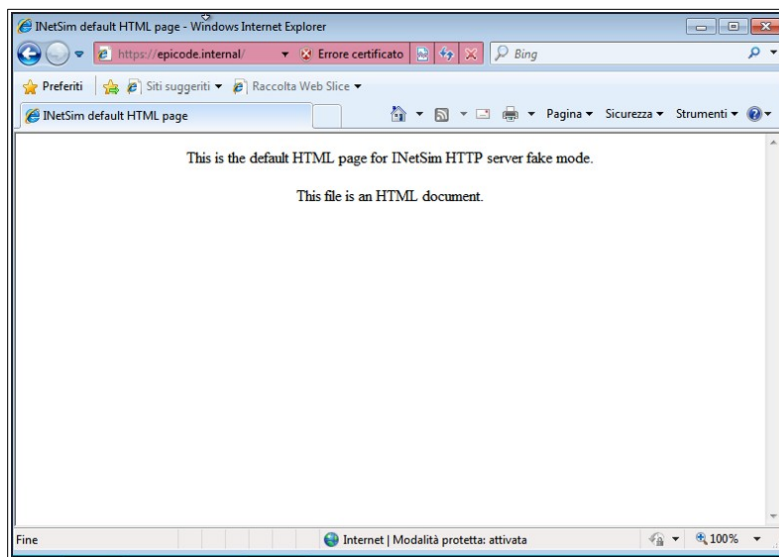
## Test connettività

Ora andremo a verificare che tutti i servizi funzionino correttamente.

Per farlo basterà aprire *internet explorer* da macchina Windows (assicurandoci che inetsim sia in funzione).

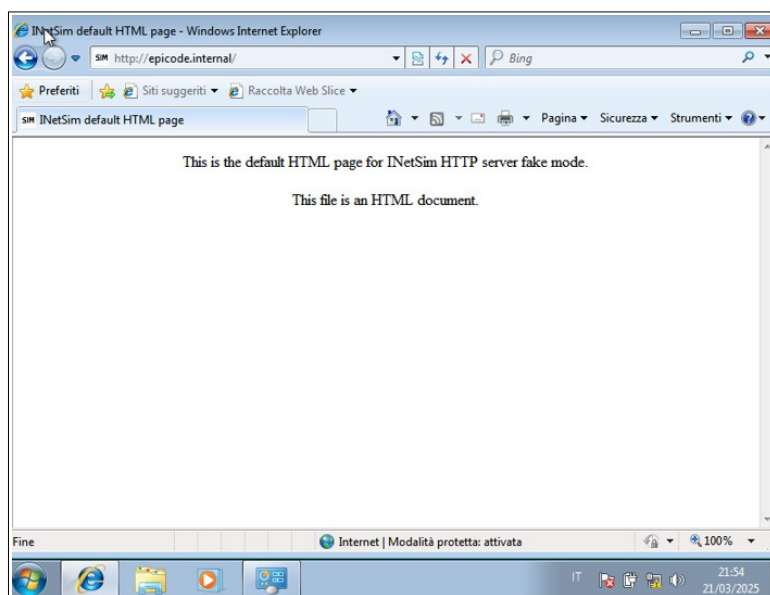
## Test HTTPS

Inseriamo nella barra di ricerca prima *https://epicode.internal/*, accettiamo il messaggio di avviso di sicurezza e potremo visualizzare la pagina web fittizia fornita da inetsim.



## Test HTTP

Ripetiamo il passaggio precedente ma stavolta digitiamo *http://epicode.internal/*.



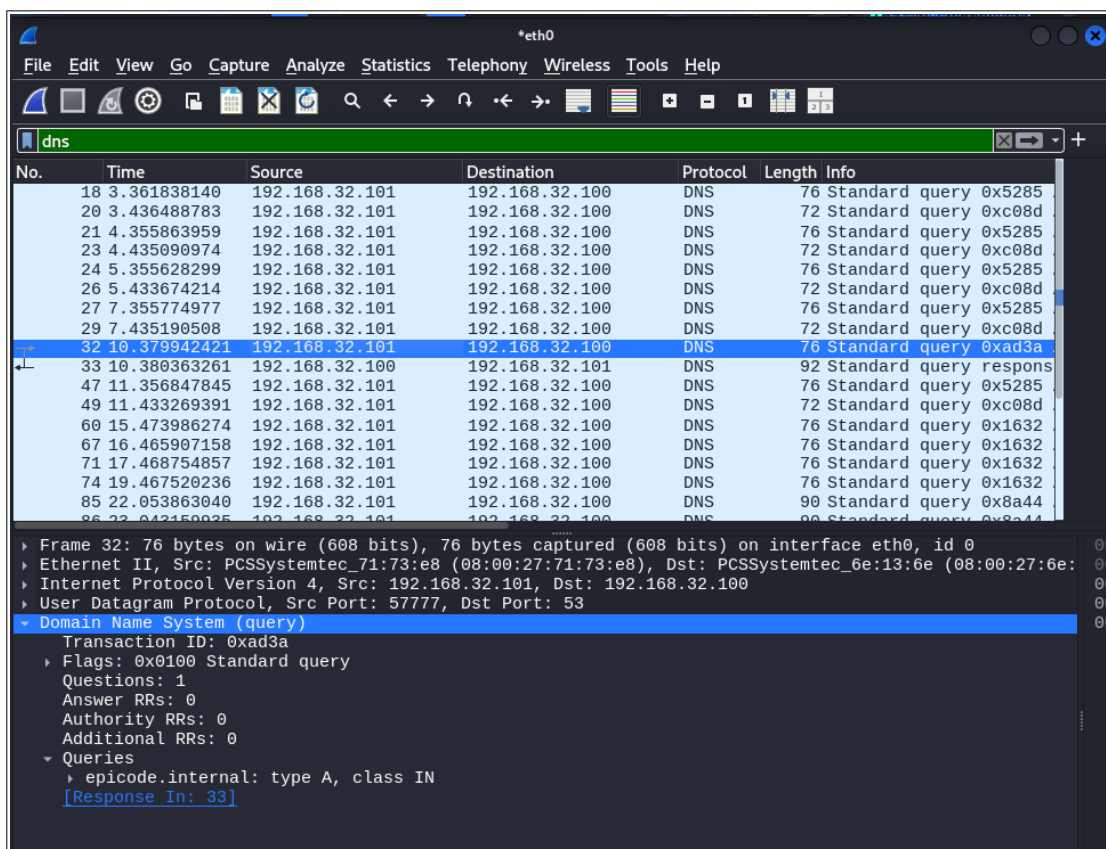
## Intercettazione pacchetti con Wireshark

In questa ultima sezione dell'esercizio ci concentreremo sull'analisi dei pacchetti intercettati con Wireshark da macchina Kali, in ascolto su *eth0*.

Per ogni analisi dovremo ripetere le connessioni HTTPS e HTTP mentre Wireshark è in esecuzione.

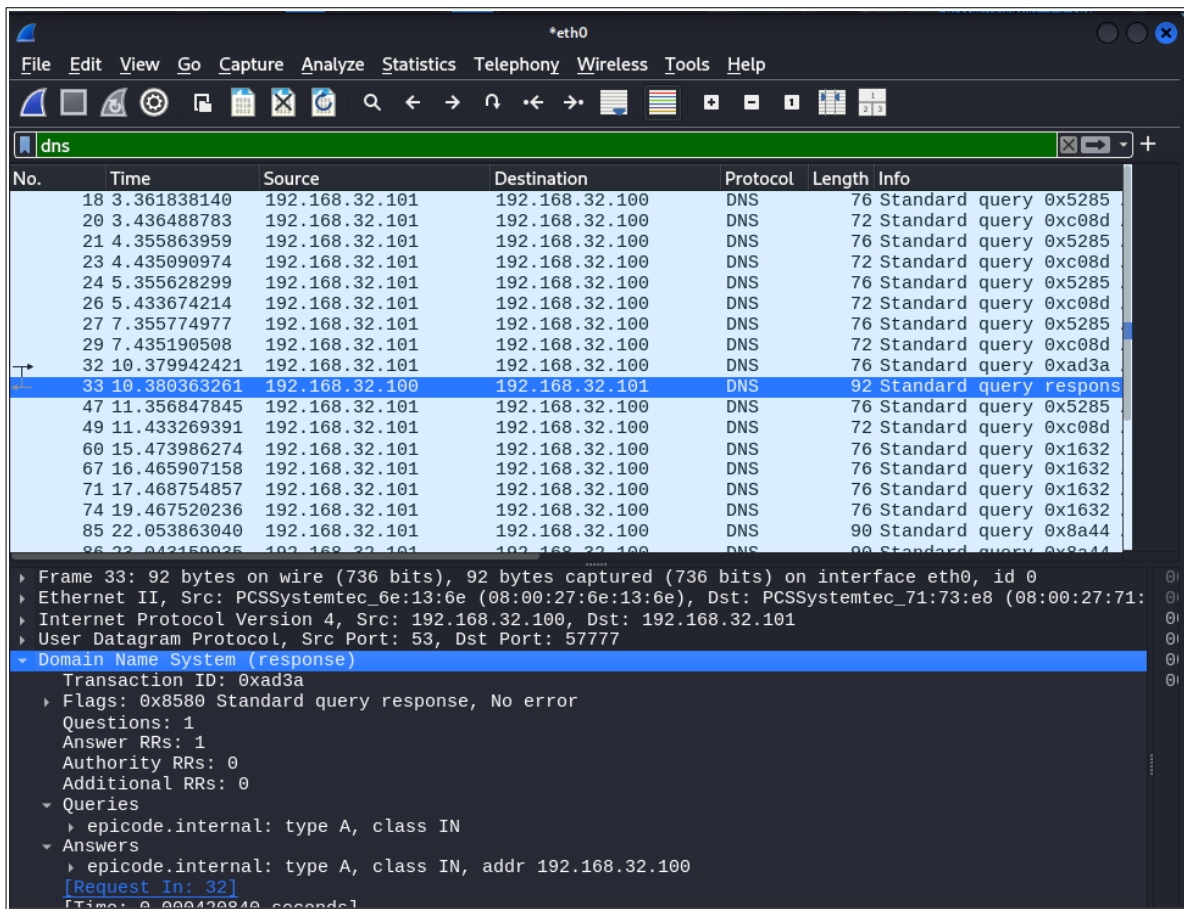
### Analisi pacchetti DNS

Impostando il filtro DNS ed analizzando i pacchetti possiamo vedere come nella riga 32 abbiamo una query DNS da 192.168.32.101 (Windows) verso 192.168.32.100 (Kali).





Nella riga 33 avremo la DNS response corretta da parte di Kali verso Windows.



No.	Time	Source	Destination	Protocol	Length	Info
18	3.361838140	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x5285
20	3.436488783	192.168.32.101	192.168.32.100	DNS	72	Standard query 0xc08d
21	4.355863959	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x5285
23	4.435090974	192.168.32.101	192.168.32.100	DNS	72	Standard query 0xc08d
24	5.355628299	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x5285
26	5.433674214	192.168.32.101	192.168.32.100	DNS	72	Standard query 0xc08d
27	7.355774977	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x5285
29	7.435190508	192.168.32.101	192.168.32.100	DNS	72	Standard query 0xc08d
32	10.379942421	192.168.32.101	192.168.32.100	DNS	76	Standard query 0xad3a
33	10.380363261	192.168.32.100	192.168.32.101	DNS	92	Standard query response
47	11.356847845	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x5285
49	11.433269391	192.168.32.101	192.168.32.100	DNS	72	Standard query 0xc08d
60	15.473986274	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x1632
67	16.465907158	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x1632
71	17.468754857	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x1632
74	19.467520236	192.168.32.101	192.168.32.100	DNS	76	Standard query 0x1632
85	22.053863040	192.168.32.101	192.168.32.100	DNS	90	Standard query 0x8a44
86	22.042159025	192.168.32.101	192.168.32.100	DNS	90	Standard query 0x8a44

Frame 33: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id 0

Ethernet II, Src: PCSSystemtec\_6e:13:6e (08:00:27:6e:13:6e), Dst: PCSSystemtec\_71:73:e8 (08:00:27:71:73:e8)

Internet Protocol Version 4, Src: 192.168.32.100, Dst: 192.168.32.101

User Datagram Protocol, Src Port: 53, Dst Port: 57777

Domain Name System (response)

Transaction ID: 0xad3a

Flags: 0x8580 Standard query response, No error

Questions: 1

Answer RRs: 1

Authority RRs: 0

Additional RRs: 0

Queries

epicode.internal: type A, class IN

Answers

epicode.internal: type A, class IN, addr 192.168.32.100

[Request In: 32]

[Time: 0.000420040 seconds]



## Analisi pacchetti HTTPS

Prendendo in esame la riga 140, 141 e 142 possiamo vedere la three way handshake (syn, syn+ack, ack), il protocollo utilizzato utilizzato per la trasmissione (TCP), e la porta in ascolto sul servizio (443).

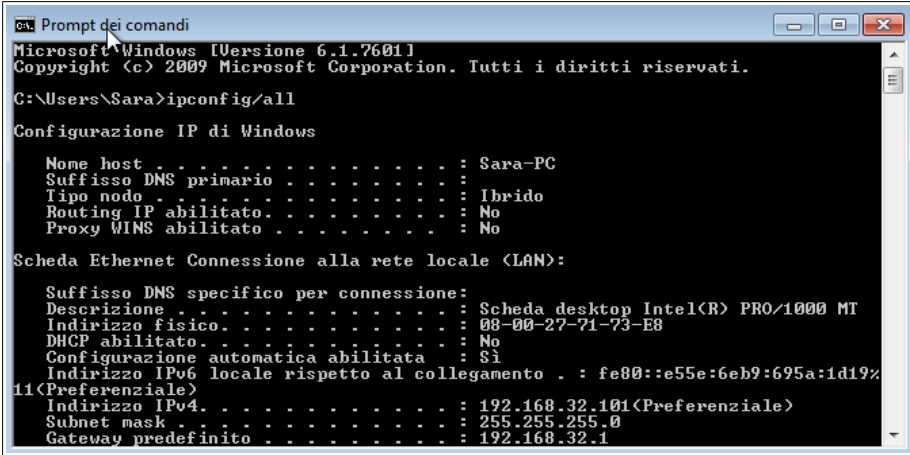
The image shows a Wireshark packet capture on interface eth0. The packet list pane displays several packets, with packets 140, 141, and 142 highlighted in blue. Packet 140 is a SYN packet from 192.168.32.101 to 192.168.32.100. Packet 141 is a SYN, ACK packet from 192.168.32.100 to 192.168.32.101. Packet 142 is an ACK packet from 192.168.32.101 to 192.168.32.100. The packet details pane for packet 140 is expanded, showing the following information:

- Frame 140: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0, id 0
- Ethernet II, Src: PCSSystemtec\_71:73:e8 (08:00:27:71:73:e8), Dst: PCSSystemtec\_6e:13:6e (08:00:27:6e:13:6e)
- Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100
- Transmission Control Protocol, Src Port: 49166, Dst Port: 443, Seq: 0, Len: 0
  - Source Port: 49166
  - Destination Port: 443
  - [Stream index: 2]
  - [Stream Packet Number: 1]
  - [Conversation completeness: Complete, WITH\_DATA (63)]
  - [TCP Segment Len: 0]
  - Sequence Number: 0 (relative sequence number)
  - Sequence Number (raw): 1471426949
  - [Next Sequence Number: 1 (relative sequence number)]
  - Acknowledgment Number: 0
  - Acknowledgment number (raw): 0
  - 1000 .... = Header Length: 32 bytes (8)

Analizzando più nello specifico la riga 140 (SYN) possiamo esaminare gli indirizzi di sorgente e destinazione:

- MAC address sorgente: 08:00:27:71:73:e8
- IP sorgente: 192.168.32.101
- MAC address destinazione: 08:00:27:6e:13:6e
- IP destinazione: 192.168.32.100

Le immagini sottostanti mostrano che il MAC address sorgente appartiene a Windows e quello di destinazione a Kali.



```
Prompt dei comandi
Microsoft Windows [Versione 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tutti i diritti riservati.

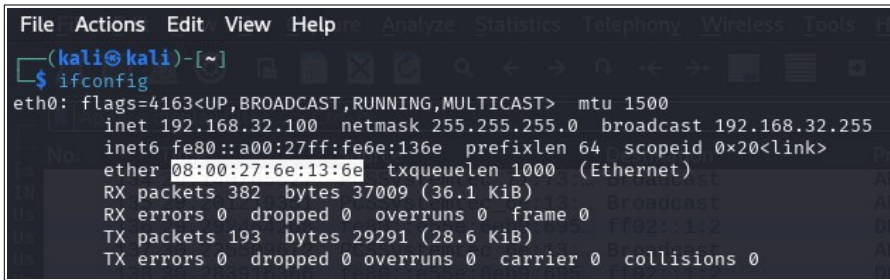
C:\Users\Sara>ipconfig/all

Configurazione IP di Windows

Nome host . . . . . : Sara-PC
Suffisso DNS primario . . . . . : 
Tipo nodo . . . . . : Ibrido
Routing IP abilitato . . . . . : No
Proxy WINS abilitato . . . . . : No

Scheda Ethernet Connessione alla rete locale (LAN):

Suffisso DNS specifico per connessione:
Descrizione . . . . . : Scheda desktop Intel(R) PRO/1000 MT
Indirizzo fisico . . . . . : 08-00-27-71-73-E8
DHCP abilitato . . . . . : No
Configurazione automatica abilitata . . . . . : Sì
Indirizzo IPv6 locale rispetto al collegamento . : fe80::e55e:6eb9:695a:1d19%11(Preferenziale)
Indirizzo IPv4 . . . . . : 192.168.32.101(Preferenziale)
Subnet mask . . . . . : 255.255.255.0
Gateway predefinito . . . . . : 192.168.32.1
```



```
File Actions Edit View Help
(kali@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.32.100 netmask 255.255.255.0 broadcast 192.168.32.255
    inet6 fe80::a00:27ff:fe6e:136e prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:6e:13:6e txqueuelen 1000 (Ethernet)
    RX packets 382 bytes 37009 (36.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 193 bytes 29291 (28.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

La riga 141 contiene la risposta di Kali (SYN + ACK), quindi notiamo che gli indirizzi sorgente e destinazione si scambiano.

Infine nella riga 142 abbiamo l'invio dell'ACK da Windows verso Kali.

Nel caso del protocollo HTTPS non potremo però analizzare il contenuto dei pacchetti in quanto si tratta di un protocollo cifrato.

Le informazioni saranno quindi criptate ed inaccessibili grazie alla TLS handshake, che assicura la protezione delle informazioni trasmesse criptandone il contenuto, (righe 143, 145 e 146).

143	31.007807239	192.168.32.101	192.168.32.100	TLSv1	215 Client Hello (SNI=epicode.internal)
144	31.007818743	192.168.32.100	192.168.32.101	TCP	54 443 → 49166 [ACK] Seq=1 Ack=162 Win=64128 Len=0
145	31.049158757	192.168.32.100	192.168.32.101	TLSv1	1373 Server Hello, Certificate, Server Key Exchange, Server Hello Done
146	31.054036179	192.168.32.101	192.168.32.100	TLSv1	188 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

The image shows a Wireshark packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The main window is divided into three panes: Packet List, Packet Details, and Packet Bytes.

**Packet List:**

Time	Source	Destination	Protocol	Length	Info
142	31.007078594	192.168.32.101	TCP	60	49166 → 443 [ACK] Seq=1 A
143	31.007807239	192.168.32.101	TLSv1	215	Client Hello (SNI=epicode
144	31.007818743	192.168.32.100	TCP	54	443 → 49166 [ACK] Seq=1 A
145	31.049158757	192.168.32.100	TLSv1	1373	Server Hello, Certificate
146	31.054036179	192.168.32.101	TLSv1	188	Client Key Exchange, Chan
147	31.054284804	192.168.32.100	TCP	54	443 → 49166 [ACK] Seq=132
148	31.055611870	192.168.32.100	TLSv1	113	Change Cipher Spec, Encry
149	31.059077732	192.168.32.101	TCP	60	49166 → 443 [FIN, ACK] Se
150	31.059078450	192.168.32.101	TCP	66	49167 → 443 [SYN] Seq=0 W
151	31.059444970	192.168.32.101	TCP	66	443 → 49167 [SYN, ACK] Se
152	31.060800248	192.168.32.101	TCP	60	49167 → 443 [ACK] Seq=1 A
153	31.061715303	192.168.32.101	TLSv1	215	Client Hello (SNI=epicode
154	31.061734167	192.168.32.100	TCP	54	443 → 49167 [ACK] Seq=1 A
155	31.066508057	192.168.32.100	TLSv1	91	Encrypted Alert
156	31.067215963	192.168.32.101	TCP	60	49166 → 443 [RST, ACK] Se
157	31.067215963	192.168.32.100	TLSv1	1373	Server Hello, Certificate

**Packet Details:**

- Acknowledgment number (raw): 1935748103
- 0101 .... = Header Length: 20 bytes (5)
- Flags: 0x018 (PSH, ACK)
- Window: 16095
- [Calculated window size: 64380]
- [Window size scaling factor: 4]
- Checksum: 0x3d1d [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- [Timestamps]
- [SEQ/ACK analysis]
- TCP payload (134 bytes)
- Transport Layer Security
  - TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
  - TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  - TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message

## Analisi pacchetti HTTP

Procediamo all'analisi dei pacchetti della connessione con HTTP.

Non essendo un protocollo criptato, possiamo analizzare il contenuto in chiaro del pacchetto.

Come possiamo vedere nelle immagini sottostanti, notiamo il protocollo usato per la trasmissione (TCP), la three way handshake (righe 7, 8 e 13), la porta in ascolto (80), gli IP e i MAC address sorgente e destinazione, infine la richiesta in chiaro GET (riga 16) e la risposta positiva dal server 200 ok (riga 19).

Time	Source	Destination	Protocol	Length	Info
7 0.006902674	192.168.32.101	192.168.32.100	TCP	66	49168 → 80 [SYN] Seq=0 Win=8192 Len=6
8 0.007054831	192.168.32.100	192.168.32.101	TCP	66	80 → 49168 [SYN, ACK] Seq=0 Ack=1 Win
13 0.008517488	192.168.32.101	192.168.32.100	TCP	60	49168 → 80 [ACK] Seq=1 Ack=1 Win=6570

The image shows a Wireshark packet capture on interface eth0. The packet list pane displays several packets, with packet 16 selected. The packet details pane shows the structure of the selected packet, and the packet bytes pane shows the raw data.

**Packet List:**

No.	Time	Source	Destination	Protocol	Length	Info
13	0.008517488	192.168.32.101	192.168.32.100	TCP	60	49168 → 80 [ACK] Seq=1
14	0.008517564	fe80::e55e:6eb9:695...	ff02::1:3	LLMNR	87	Standard query 0x8693 A
15	0.008517639	192.168.32.101	224.0.0.252	LLMNR	67	Standard query 0x8693 A
16	0.008517717	192.168.32.101	192.168.32.100	HTTP	472	GET / HTTP/1.1
17	0.008676386	192.168.32.100	192.168.32.101	TCP	54	80 → 49168 [ACK] Seq=1
18	0.078444014	192.168.32.100	192.168.32.101	TCP	204	80 → 49168 [PSH, ACK] S
19	0.085003089	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/
20	0.087064694	192.168.32.101	192.168.32.100	TCP	60	49168 → 80 [ACK] Seq=41
21	0.087065184	192.168.32.101	192.168.32.100	TCP	60	49168 → 80 [FIN, ACK] S
22	0.087192319	192.168.32.100	192.168.32.101	TCP	54	80 → 49168 [ACK] Seq=41
23	0.107721234	fe80::e55e:6eb9:695...	ff02::1:3	LLMNR	87	Standard query 0x8693 A
24	0.108768160	192.168.32.101	224.0.0.252	LLMNR	67	Standard query 0x8693 A
25	0.297080244	192.168.32.101	224.0.0.22	IGMPv3	60	Membership Report / Joi
26	0.297081086	fe80::e55e:6eb9:695...	ff02::1:6	ICMPv6	90	Multicast Listener Repo
27	5.240485063	PCSSystemtec_6e:13:...	PCSSystemtec_71:73:...	ARP	42	Who has 192.168.32.101?

**Packet Details (Frame 16):**

- Frame 16: 472 bytes on wire (3776 bits), 472 bytes captured (3776 bits) on interface eth0, id 0
- Ethernet II, Src: PCSSystemtec\_71:73:e8 (08:00:27:71:73:e8), Dst: PCSSystemtec\_6e:13:6e (08:00:27:6e:13:6e)
- Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100
- Transmission Control Protocol, Src Port: 49168, Dst Port: 80, Seq: 1, Ack: 1, Len: 418
- Hypertext Transfer Protocol
  - GET / HTTP/1.1\r\n
  - Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, appl
  - Accept-Language: it-IT\r\n
  - User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2
  - Accept-Encoding: gzip, deflate\r\n
  - Host: epicode.internal\r\n
  - Connection: Keep-Alive\r\n
  - \r\n
  - [Response in frame: 19]
  - [Full request URI: http://epicode.internal/]

**Packet Bytes:**

wireshark\_eth0Q1UE32.pcapng

Packets: 28 · Dropped: 0 (0.0%)

Profile: Default

## Considerazioni finali

Tramite l'analisi con Wireshark si nota che per quanto riguarda il protocollo HTTPS, successivamente alla three way handshake si verifica la TLS handshake che assicura la trasmissione cifrata e protetta dei dati stabilendo l'algoritmo crittografico e scambiando le chiavi di cifratura.

I dati trasmessi tramite protocollo HTTP, sono invece in chiaro e abbiamo quindi potuto analizzare il contenuto sia della richiesta che della risposta.

Il numero di pacchetti scambiati con HTTPS sarà maggiore rispetto a quelli di HTTP, per via della cifratura prevista dal protocollo.

In entrambi i protocolli i MAC address e gli IP saranno visibili.