


## ESERCITAZIONE WEEK4 DAYS



Esercizio  
Traccia e requisiti

Nell'esercizio di oggi metteremo insieme le competenze acquisite finora.  
Lo studente verrà valutato sulla base della risoluzione al problema seguente.

**Requisiti e servizi:**

- Kali Linux ☐ IP 192.168.32.100
- Windows 7 ☐ IP 192.168.32.101
- HTTPS server: attivo
- Servizio DNS per risoluzione nomi di dominio: attivo

**Traccia:**

Simulare, in ambiente di laboratorio virtuale, un'architettura client server in cui un client con indirizzo 192.168.32.101 (Windows 7) richiede tramite web browser una risorsa all'hostname `epicode.internal` che risponde all'indirizzo 192.168.32.100 (Kali).

Si intercetti poi la comunicazione con Wireshark, evidenziando i MAC address di sorgente e destinazione ed il contenuto della richiesta HTTPS.

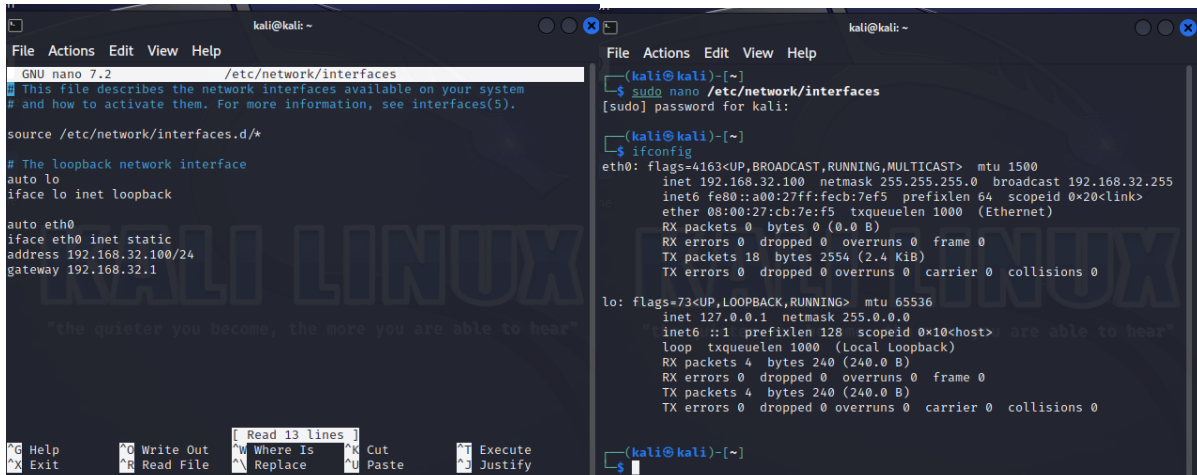
Ripetere l'esercizio, sostituendo il server HTTPS, con un server HTTP. Si intercetti nuovamente il traffico, evidenziando le eventuali differenze tra il traffico appena catturato in HTTP ed il traffico precedente in HTTPS. Spiegare, motivandole, le principali differenze se presenti.

2

Per lo svolgimento di questo esercizio, occorre installare su VirtualBox le macchine virtuali Kali Linux e Windows 7 che verranno impostate nella modalità di rete *internal* così da permettere la loro comunicazione reciproca, ma non con l'ambiente esterno.

Per prima cosa si configura la rete e gli indirizzi IP delle macchine Kali Linux e Windows 7.

Per farlo, su Kali Linux da terminale si invia il comando **sudo nano /etc/network/interfaces** in modo da poter modificare il file nel seguente modo:



```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/network/interfaces  
# This file describes the network interfaces available on your system  
# and how to activate them. For more information, see interfaces(5).  
  
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  
  
"the quieter you become, the more you are able to hear"
```

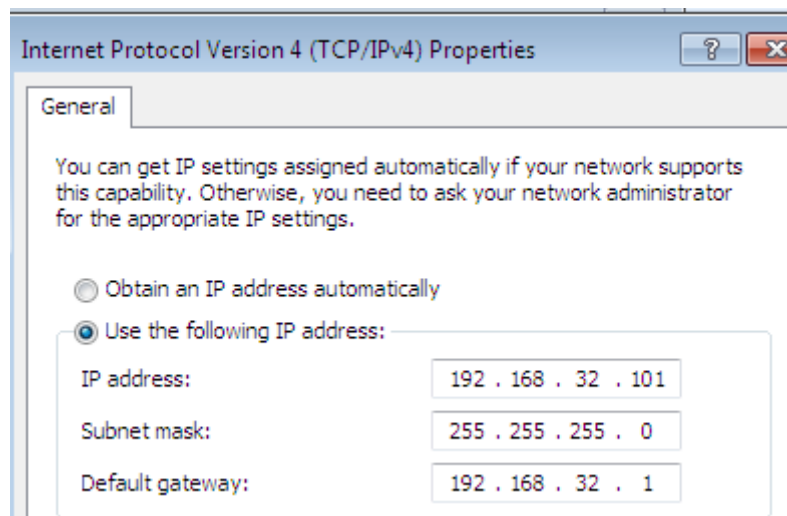
```
kali@kali: ~  
File Actions Edit View Help  
$ sudo nano /etc/network/interfaces  
[sudo] password for kali:  
  
$ ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
inet 192.168.32.100 netmask 255.255.255.0 broadcast 192.168.32.255  
inet6 fe80::a00:27ff:feeb:7ef5 prefixlen 64 scopeid 0x20<link>  
ether 08:00:27:cb:7e:f5 txqueuelen 1000 (Ethernet)  
RX packets 0 bytes 0 (0.0 B)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 18 bytes 2554 (2.4 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> are able to hear"  
loop txqueuelen 1000 (Local Loopback)  
RX packets 4 bytes 240 (240.0 B)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 4 bytes 240 (240.0 B)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Si imposta dunque l'IP della macchina Kali Linux a 192.168.32.100, l'indirizzo di gateway pari a 192.168.32.1 come da convenzione. Dopo aver salvato, si manda il comando **ifconfig** per controllare che le modifiche siano state apportate. Se non risultano apportate, si riavvia la macchina e si rimanda tale comando.

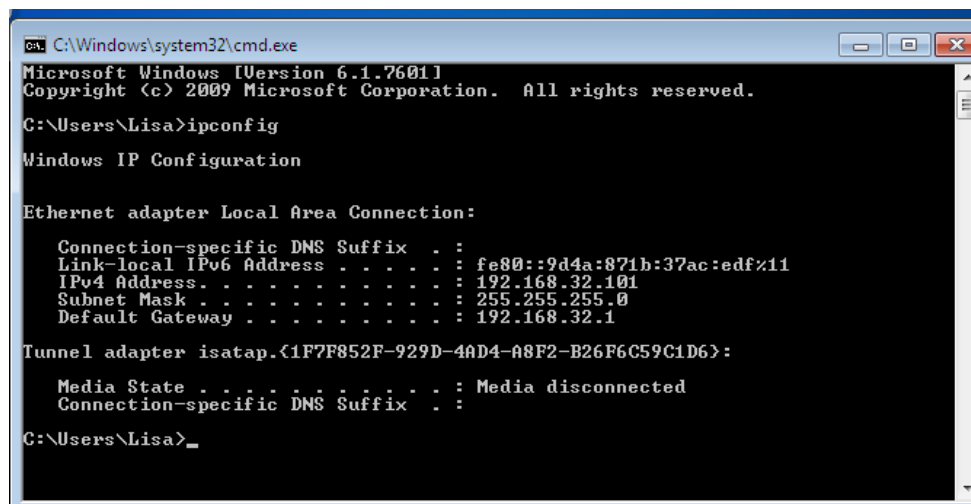
Dalla risposta ottenuta possiamo notare anche che la netmask è impostata a 255.255.255.0 e l'IP riservato per il broadcast è come da convenzione 192.168.32.255.

L'IP di rete sarà 192.168.32.0.

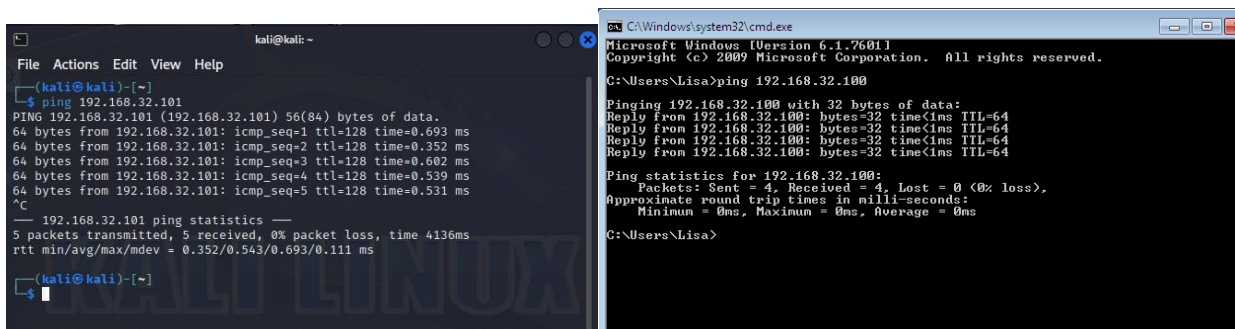
Per Windows 7, queste configurazioni si fanno da interfaccia grafica, andando a configurare il protocollo IPv4:



Anche in questo caso, da terminale si invia il comando **ipconfig** per verificare che le modifiche siano state correttamente salvate:



Inoltre, per verificare che effettivamente ci sia comunicazione tra le macchine, si invia dai terminali il comando **ping** con cui si inviano dei pacchetti verso la macchina destinataria e si verifica la loro corretta ricezione:

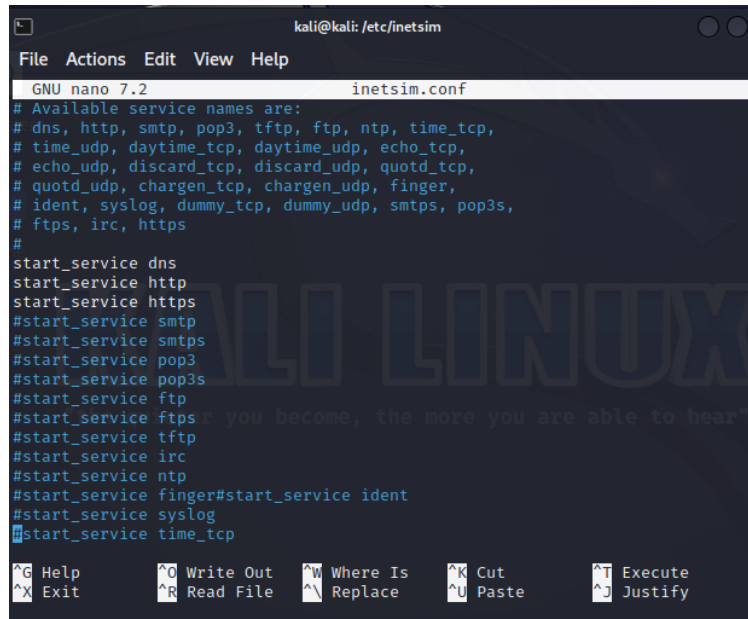


Si precisa che per Windows 7 è stata creata una nuova policy del Firewall per permettere le richieste di ping in ingresso (inbound rules).

Una volta ottenuta la corretta comunicazione tra le due macchine, si procede ad attivare i servizi interessati dal caso in esame: HTTP, HTTPS, DNS.

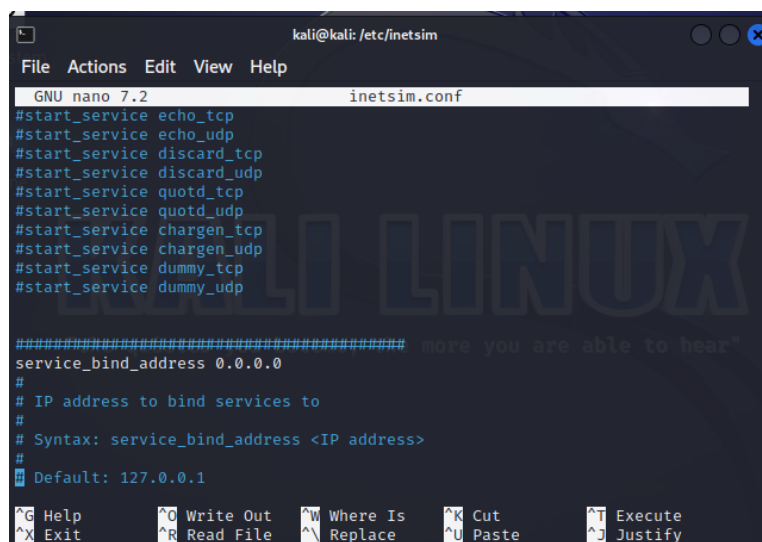
Per farlo, ci si serve di un software messo a disposizione da Kali Linux per simulare i principali e comuni servizi internet in ambiente virtuale, chiamato *inetsim*. Da terminale, ci si sposta nella directory dove è contenuto il file che si intende modificare, attraverso il comando **cd /etc/inetsim** e si invia il comando **sudo nano inetsim.conf**. In tal modo possiamo modificare il file *inetsim.conf* in base alle nostre necessità.

Si scommentano dunque i servizi di nostro interesse:



```
kali@kali: /etc/inetsim
File Actions Edit View Help
GNU nano 7.2 inetsim.conf
# Available service names are:
# dns, http, smtp, pop3, tftp, ftp, ntp, time_tcp,
# time_udp, daytime_tcp, daytime_udp, echo_tcp,
# echo_udp, discard_tcp, discard_udp, quotd_tcp,
# quotd_udp, chargen_tcp, chargen_udp, finger,
# ident, syslog, dummy_tcp, dummy_udp, smtps, pop3s,
# ftps, irc, https
#
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
#start_service tftp
#start_service irc
#start_service ntp
#start_service finger
#start_service ident
#start_service syslog
#start_service time_tcp
```

Si imposta il *service bind address* pari a 0.0.0.0 in modo che il server si metta in ascolto su qualsiasi interfaccia di rete.



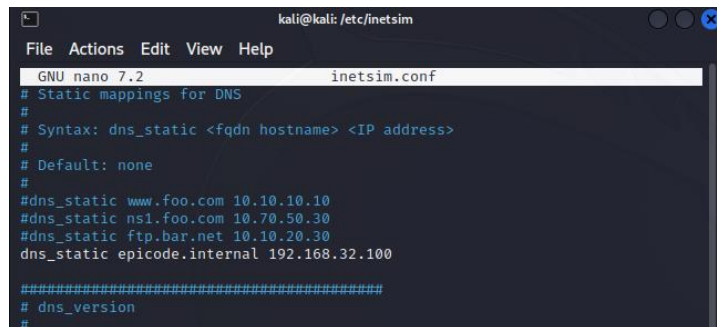
```
kali@kali: /etc/inetsim
File Actions Edit View Help
GNU nano 7.2 inetsim.conf
#start_service echo_tcp
#start_service echo_udp
#start_service discard_tcp
#start_service discard_udp
#start_service quotd_tcp
#start_service quotd_udp
#start_service chargen_tcp
#start_service chargen_udp
#start_service dummy_tcp
#start_service dummy_udp

##### more you are able to hear"
service_bind_address 0.0.0.0
#
# IP address to bind services to
#
# Syntax: service_bind_address <IP address>
#
# Default: 127.0.0.1
```

Questa modifica è necessaria perché nel file di default il *service bind address* è impostato a 127.0.0.1 per cui inetsim ascolterebbe solo sull'interfaccia di loopback e quindi solo le richieste dei programmi sulla stessa macchina. Altre macchine sulla stessa rete non sarebbero così in grado di raggiungere inetsim.

Il servizio DNS permette di tradurre nomi di dominio in indirizzi IP che i browser usano per caricare le risorse Internet. In questo esercizio si richiede la risoluzione del dominio *epicode.internal* che deve rispondere all'indirizzo della macchina Kali Linux.

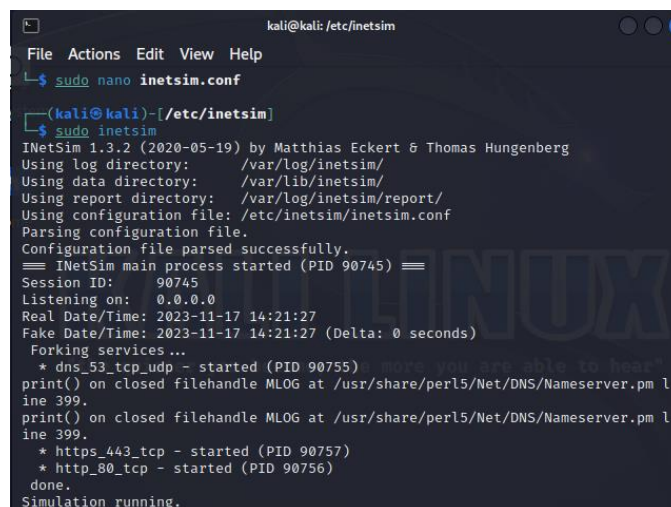
Dunque si modifica il dns statico:



```
kali@kali: /etc/inetsim
File Actions Edit View Help
GNU nano 7.2 inetsim.conf
# Static mappings for DNS
#
# Syntax: dns_static <fqdn hostname> <IP address>
#
# Default: none
#
#dns_static www.foo.com 10.10.10.10
#dns_static ns1.foo.com 10.70.50.30
#dns_static ftp.bar.net 10.10.20.30
#dns_static epicode.internal 192.168.32.100
#####
# dns_version
#
```

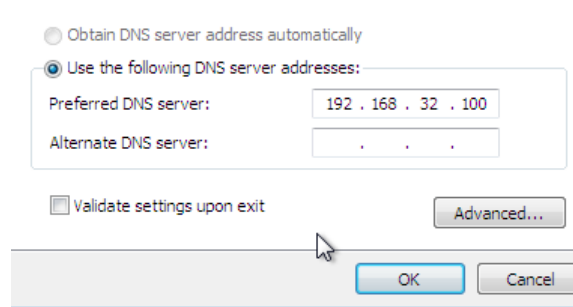
La porta di default associata al DNS è la 53, al servizio HTTP è la 80 e al servizio HTTPS è la 443.

Il file deve poi essere salvato e da terminale si può lanciare inetsim con il comando **sudo inetsim**:



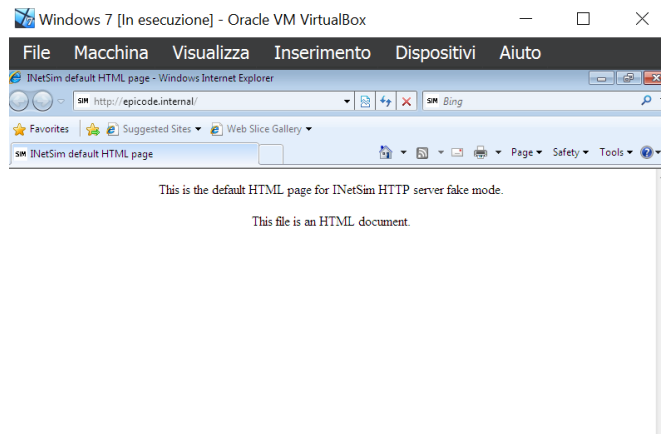
```
kali@kali: /etc/inetsim
File Actions Edit View Help
$ sudo nano inetsim.conf
(kali@kali)-[/etc/inetsim]
$ sudo inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory: /var/log/inetsim/
Using data directory: /var/lib/inetsim/
Using report directory: /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
== INetSim main process started (PID 90745) ==
Session ID: 90745
Listening on: 0.0.0.0
Real Date/Time: 2023-11-17 14:21:27
Fake Date/Time: 2023-11-17 14:21:27 (Delta: 0 seconds)
Forking services ...
* dns_53_tcp_udp - started (PID 90755)
print() on closed filehandle MLOG at /usr/share/perl5/Net/DNS/Nameserver.pm l
ine 399.
print() on closed filehandle MLOG at /usr/share/perl5/Net/DNS/Nameserver.pm l
ine 399.
* https_443_tcp - started (PID 90757)
* http_80_tcp - started (PID 90756)
done.
Simulation running.
```

Sulle impostazioni di configurazione di Windows 7 si specifica l'IP del DNS come preferito:

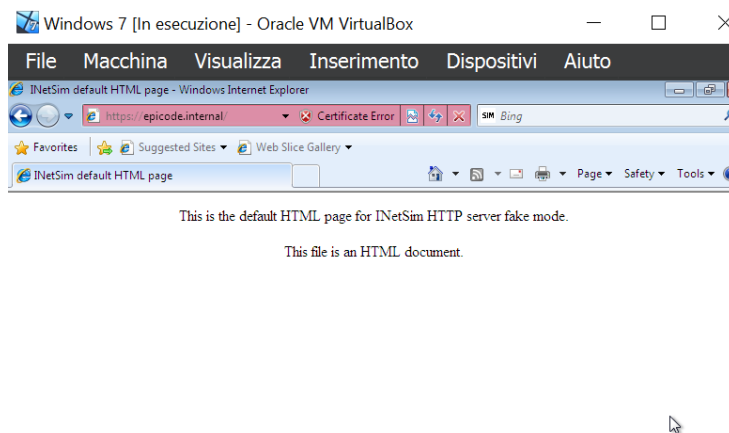


Una volta che inetsim è avviato, da Windows 7 si può aprire il browser e navigare su:

1. <http://epicode.internal/>



2. <https://epicode.internal/>



In entrambi i casi si dimostra la corretta risoluzione IP-dominio da parte del DNS.

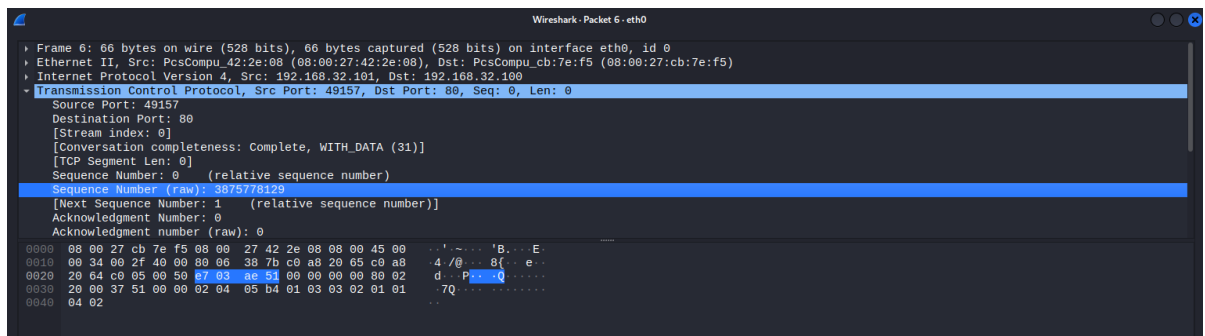
A questo punto si intercetta il traffico con Wireshark e si analizzano i pacchetti in transito. Si inizia con il server HTTP:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_42:2e:08	Broadcast	ARP	60	who has 192.168.32.17 Tell 192.168.32.101
2	0.863471666	PcsCompu_42:2e:08	Broadcast	ARP	60	who has 192.168.32.17 Tell 192.168.32.101
3	1.862489237	PcsCompu_42:2e:08	Broadcast	ARP	60	who has 192.168.32.17 Tell 192.168.32.101
4	38.298824657	PcsCompu_42:2e:08	Broadcast	ARP	60	who has 192.168.32.100? Tell 192.168.32.101
5	38.298853251	PcsCompu_cb:7e:f5	PcsCompu_42:2e:08	ARP	42	192.168.32.100 is at 08:00:27:cb:7e:f5
6	38.299935818	192.168.32.101	192.168.32.100	TCP	60	49157 → 80 [SYN] Seq=0 Win=0 Len=0 MSS=1460 WS=4 SACK_PERM
7	38.299965219	192.168.32.100	192.168.32.101	TCP	60	80 → 49157 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
8	38.308122381	192.168.32.101	192.168.32.100	TCP	60	49157 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
9	38.308263191	192.168.32.101	192.168.32.100	HTTP	465	GET / HTTP/1.1
10	38.308271846	192.168.32.100	192.168.32.101	TCP	54	80 → 49157 [ACK] Seq=1 Ack=412 Win=64128 Len=0
11	38.315190373	192.168.32.100	192.168.32.101	TCP	204	80 → 49157 [PSH, ACK] Seq=1 Ack=412 Win=64128 Len=150 [TCP segment of a reassembled PDU]
12	38.316678342	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
13	38.316933406	192.168.32.101	192.168.32.100	TCP	60	49157 → 80 [ACK] Seq=412 Ack=410 Win=65292 Len=0
14	38.318053006	192.168.32.101	192.168.32.100	TCP	60	49157 → 80 [FIN, ACK] Seq=412 Ack=410 Win=65292 Len=0
15	38.318062342	192.168.32.100	192.168.32.101	TCP	54	80 → 49157 [ACK] Seq=410 Ack=413 Win=64128 Len=0
16	38.355609291	192.168.32.101	192.168.32.100	TCP	60	49158 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
17	38.355763098	192.168.32.100	192.168.32.101	TCP	60	80 → 49158 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
18	38.356042401	192.168.32.101	192.168.32.100	TCP	60	49158 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
19	38.356042557	192.168.32.101	192.168.32.100	HTTP	341	GET /favicon.ico HTTP/1.1
20	38.356076829	192.168.32.100	192.168.32.101	TCP	54	80 → 49158 [ACK] Seq=1 Ack=288 Win=64128 Len=0

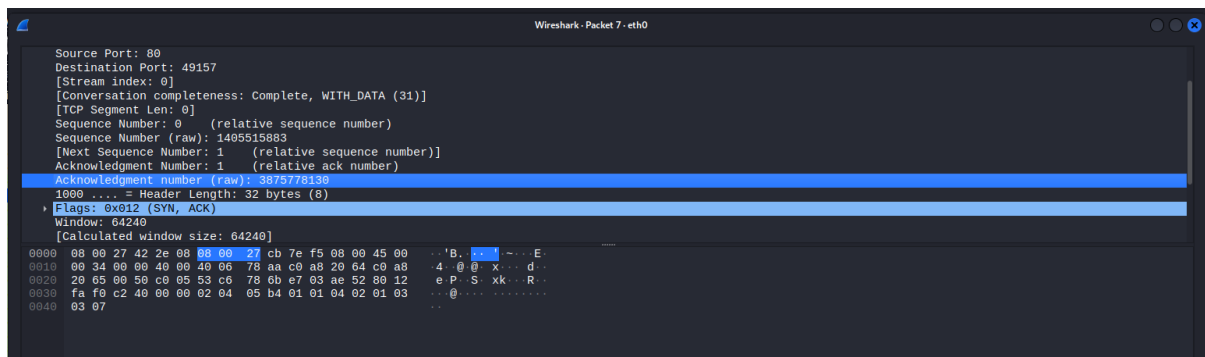
La prima cosa che si nota è lo scambio di informazioni tramite protocollo ARP: la sorgente con MAC 08:00:27:42:2e:08 ovvero la macchina di Windows 7 sta facendo una richiesta broadcast per capire chi è il dispositivo che possiede nella rete l'IP 192.168.32.100 (pacchetto numero 4).

Il dispositivo proprietario di quell'indirizzo IP, ovvero Kali Linux, con MAC 08:00:27:cb:7e:f5 risponde al MAC 08:00:27:42:2e:08 (W7) che si tratta di se stesso. Può dunque iniziare la comunicazione tra i due indirizzi IP tramite protocollo TCP che garantisce la corretta trasmissione e ricezione dei pacchetti, reinviando eventuali pacchetti persi. Quindi inizia il cosiddetto "three-way handshake":

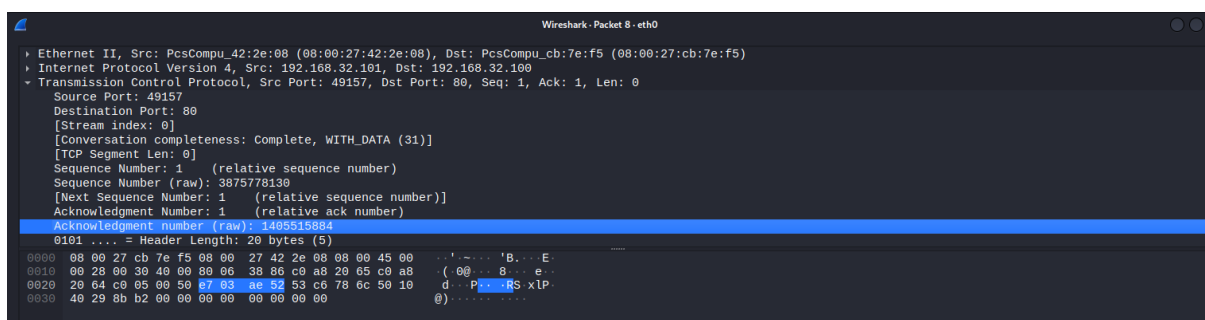
- Il client Windows 7 invia dalla sua porta 49157 (assegnata randomicamente) verso la porta HTTP 80 una richiesta con il flag SYN attivo e sequence number 3875778129;



- Il server Kali Linux risponde dalla porta HTTP 80 alla porta del client 49157 con un pacchetto in cui ci sono il flag SYN e ACK attivi, un sequence number randomico 1405515883 e un acknowledge number pari a 3875778130, ovvero il sequence number inviato dal client in precedenza più 1;



- Il client risponde dalla porta 49157 verso la 80 con un altro pacchetto con il flag ACK attivo con sequence number pari a 3875778130 e acknowledge number pari al sequence number precedente + 1, ovvero 1405515884.





Una precisazione: i MAC address sorgente e destinatario visibili da Wireshark sono confrontabili con quelli delle macchine in questione da terminale, in modo da verificare la correttezza:

```
Frame 0: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
Ethernet II, Src: PcsCompu_42:2e:08 (08:00:27:42:2e:08), Dst: PcsCompu_cb:7e:f5 (08:00:27:cb:7e:f5)
Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100

27 eth0: <BROADCAST,MULTICAST,UP>
mode DEFAULT group default qlen 10
link/ether 08:00:27:cb:7e:f5

Description: Intel(R) PRO/1000 MT
Physical Address: 08-00-27-42-2E-08
DHCP Enabled: No
Autoconfiguration Enabled: Yes
Link-local IPv6 Address: fe80::9d4a:871b:32ac:
```

Da qui inizia la comunicazione mediante protocollo HTTP. Il client (Src. 192.168.32.101) fa dalla porta 49157 una richiesta tramite il metodo GET alla destinazione 192.168.32.100, porta 80 e poiché non siamo in un canale con scambio cifrato, si riesce a vedere il contenuto di questa richiesta tramite Wireshark:

```
Wireshark - Packet 9 - eth0
Frame 9: 465 bytes on wire (3720 bits), 465 bytes captured (3720 bits) on interface eth0, id 0
Ethernet II, Src: PcsCompu_42:2e:08 (08:00:27:42:2e:08), Dst: PcsCompu_cb:7e:f5 (08:00:27:cb:7e:f5)
Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100
Transmission Control Protocol, Src Port: 49157, Dst Port: 80, Seq: 1, Ack: 1, Len: 411
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg, application/x-ms-xbap, */*\r\n
Accept-Language: en-GB\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC ...
Accept-Encoding: gzip, deflate\r\n
Host: epicode.internal\r\n
Connection: Keep-Alive\r\n
\r\n
[Full request URI: http://epicode.internal/]
0000 08 00 27 cb 7e f5 08 00 27 42 2e 08 00 00 45 09  .. 10 .. 9
0010 01 c3 00 31 40 00 00 00 36 ea c0 a9 20 05 c9 a8  .. d .. P .. RS xLP
0020 20 04 c0 05 00 50 e7 03 ae 52 53 c6 78 6c 50 18  .. 0) & .. GE T / HTTP
0030 40 29 26 88 00 00 47 45 54 20 2f 20 48 54 54 50  /1.1 ..Ac cept: im
0040 2f 31 2e 31 0d 0a 41 63 63 65 70 74 3a 20 69 6d  age/jpeg, applic
0050 61 67 65 2f 6a 70 65 67 2c 20 61 70 70 6c 69 63  ation/x- ms-appli
0060 61 74 69 6f 6e 2f 78 2d 6d 73 2d 61 70 70 6c 69  cation, image/gi
0070 63 61 74 69 6f 6e 2c 20 69 6d 61 67 65 2f 67 69  f, applic ation/x
0080 66 2c 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78  aml+xml, image/p
0090 61 6d 6c 2b 78 6d 6c 2c 20 69 6d 61 67 65 2f 70  jpeg, ap plicatio
00a0 6a 70 65 67 2c 20 61 70 70 6c 69 63 61 74 69 6f  n/x-ms-x bap, */*
00b0 6e 2f 78 2d 6d 73 2d 78 62 61 70 2c 20 2a 2f 2a  / Accept -Languag
00c0 0d 0a 41 63 65 70 74 2d 4e 61 6e 67 75 61 67  e: en-GB ..User-A
00d0 65 3a 20 65 6e 2d 47 42 0d 0a 55 73 65 72 2d 41  gent: Mo zilla/4.
00e0 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 34 2e  0 (compa tible; M
00f0 30 20 28 63 6f 6d 70 61 74 69 62 6c 65 3b 20 4d  SIE 8.0; Windows
0100 53 49 45 20 38 2e 30 3b 20 57 69 6e 64 6f 77 73
```

Dalla figura sopra riportata si evince che il metodo utilizzato è il metodo GET, su protocollo HTTP, versione 1.1, dal client Mozilla versione 4.0, Host epicode.internal, mantenendo attiva la connessione dopo la request in modalità *keep-alive*.

È inoltre visibile la risposta del server al client con il contenuto in formato testuale:

The image shows a Wireshark packet capture of an HTTP response. The packet list on the left shows Frame 12: 312 bytes on wire (2496 bits), 312 bytes captured (2496 bits) on interface eth0, id 0. The packet details pane on the right shows the following structure:

- Frame 12: 312 bytes on wire (2496 bits), 312 bytes captured (2496 bits) on interface eth0, id 0
- Ethernet II, Src: PcsCompu\_cb:7e:f5 (08:00:27:cb:7e:f5), Dst: PcsCompu\_42:2e:08 (08:00:27:42:2e:08)
- Internet Protocol Version 4, Src: 192.168.32.100, Dst: 192.168.32.101
- Transmission Control Protocol, Src Port: 80, Dst Port: 49157, Seq: 151, Ack: 412, Len: 258
- [2 Reassembled TCP Segments (408 bytes): #11(150), #12(258)]
- Hypertext Transfer Protocol
- HTTP/1.1 200 OK\r\n
- Server: InetSim HTTP Server\r\n
- Content-Type: text/html\r\n
- Connection: Close\r\n
- Content-Length: 258\r\n
- Date: Sat, 18 Nov 2023 13:03:15 GMT\r\n
- \r\n
- [HTTP response 1/1]

The packet bytes pane on the right shows the raw data of the response, which is an HTML document. The first part of the document is:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>InetSim HTTP Server</title>
</head>
<body>
<p>This is the default HTML page for InetSim HTTP server fake mode.</p>
<p>This file is an HTML document.</p>
</body>
</html>
```

La risposta contiene l'informazione che il protocollo usato è HTTP, versione 1.1, che la request e la response sono andate a buon fine (200 OK), che il server usato è InetSim HTTP Server e che il contenuto della request è testuale con lunghezza totale di 258 bytes.

Nel caso invece di una richiesta fatta con server HTTPS (da porta 49160 verso la porta 443), lo scambio dei pacchetti è cifrato usando il protocollo TLSv1, per cui non è visibile in chiaro il contenuto della comunicazione.

The image shows a Wireshark packet capture of a TLS handshake and a retransmitted packet. The packet list on the left shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::9d4a:871b:37a...	ff02::1:2	DHCPv6	149	Solicit XID: 0x38f83c CID: 000100012cce4b31080027422e08
2	0.992487564	fe80::9d4a:871b:37a...	ff02::1:2	DHCPv6	149	Solicit XID: 0x38f83c CID: 000100012cce4b31080027422e08
3	2.992412613	fe80::9d4a:871b:37a...	ff02::1:2	DHCPv6	149	Solicit XID: 0x38f83c CID: 000100012cce4b31080027422e08
4	4.086611667	PcsCompu_42:2e:08	Broadcast	ARP	60	Who has 192.168.32.100? Tell 192.168.32.101
5	4.086629248	PcsCompu_cb:7e:f5	PcsCompu_42:2e:08	ARP	42	192.168.32.100 is at 08:00:27:cb:7e:f5
6	4.086736506	192.168.32.101	192.168.32.100	TCP	66	49160 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
7	4.086762624	192.168.32.100	192.168.32.101	TCP	66	443 → 49160 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
8	4.086853594	192.168.32.101	192.168.32.100	TCP	60	49160 → 443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
9	4.089216708	192.168.32.101	192.168.32.100	TLSv1	183	Client Hello
10	4.089246940	192.168.32.101	192.168.32.100	TCP	54	443 → 49160 [ACK] Seq=1 Ack=130 Win=64128 Len=0
11	4.123153532	192.168.32.100	192.168.32.101	TLSv1	1373	Server Hello, Certificate, Server Key Exchange, Server Hello Done
12	4.138940270	192.168.32.101	192.168.32.100	TLSv1	188	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
13	4.139567941	192.168.32.100	192.168.32.101	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
14	4.1570971854	PcsCompu_42:2e:08	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101
15	4.348105664	192.168.32.100	192.168.32.101	TCP	113	[TCP Retransmission] 443 → 49160 [PSH, ACK] Seq=1320 Ack=264 Win=64
16	4.348547624	192.168.32.101	192.168.32.100	TCP	66	49160 → 443 [ACK] Seq=264 Ack=1379 Win=64320 Len=0 SLE=1320 SRE=137
17	4.961715957	PcsCompu_42:2e:08	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101
18	5.960876615	PcsCompu_42:2e:08	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101

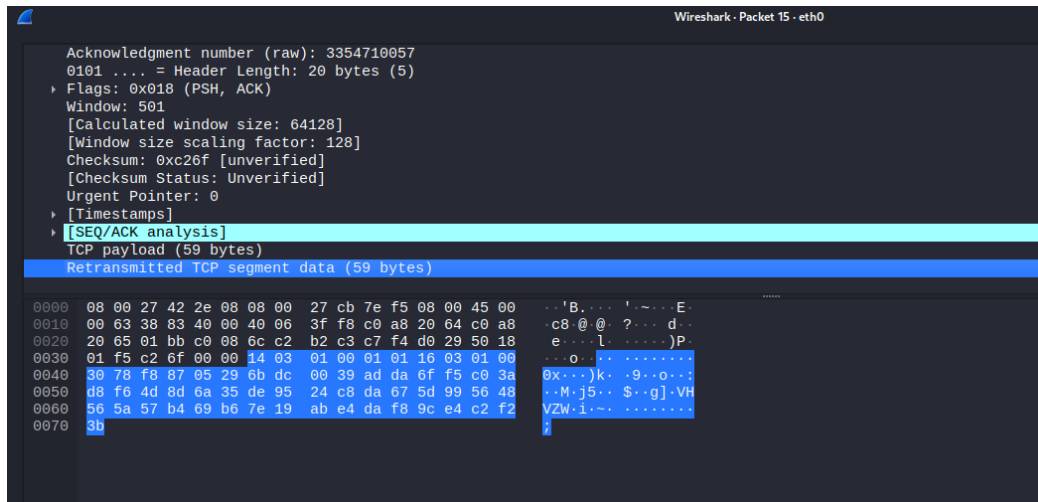
The image shows a Wireshark packet capture of a TLS handshake and a retransmitted packet. The packet list on the left shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
34	13.320957238	192.168.32.101	192.168.32.255	NBNS	92	Name query NB WPAD<00>
35	14.070329186	192.168.32.101	192.168.32.255	NBNS	92	Name query NB WPAD<00>
36	14.821436896	192.168.32.101	192.168.32.255	NBNS	92	Name query NB WPAD<00>
37	14.99282072	fe80::9d4a:871b:37a...	ff02::1:2	DHCPv6	149	Solicit XID: 0x38f83c CID: 000100012cce4b31080027422e08
38	15.596546971	192.168.32.101	192.168.32.100	TCP	60	49160 → 443 [FIN, ACK] Seq=264 Ack=1379 Win=64320 Len=0
39	15.596678377	192.168.32.100	192.168.32.101	TLSv1	91	Encrypted Alert
40	15.596806038	192.168.32.101	192.168.32.100	TCP	60	49160 → 443 [RST, ACK] Seq=265 Ack=1416 Win=0 Len=0
41	18.273416884	192.168.32.101	192.168.32.100	TCP	66	49161 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
42	18.273449784	192.168.32.100	192.168.32.101	TCP	66	443 → 49161 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
43	18.273633734	192.168.32.101	192.168.32.100	TCP	60	49161 → 443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
44	18.274036401	192.168.32.101	192.168.32.100	TLSv1	215	Client Hello
45	18.274045964	192.168.32.100	192.168.32.101	TCP	54	443 → 49161 [ACK] Seq=1 Ack=162 Win=64128 Len=0
46	18.302465722	192.168.32.100	192.168.32.101	TLSv1	1373	Server Hello, Certificate, Server Key Exchange, Server Hello Done
47	18.311957986	192.168.32.101	192.168.32.100	TLSv1	188	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
48	18.312470821	192.168.32.100	192.168.32.101	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
49	18.328140132	192.168.32.101	192.168.32.100	TCP	60	49161 → 443 [FIN, ACK] Seq=296 Ack=1379 Win=64320 Len=0
50	18.328295862	192.168.32.100	192.168.32.101	TLSv1	91	Encrypted Alert
51	18.328488908	192.168.32.101	192.168.32.100	TCP	60	49161 → 443 [RST, ACK] Seq=297 Ack=1416 Win=0 Len=0



Quello che è osservabile è che il client manda un messaggio di *Client Hello* al server che risponde con un messaggio di *Server Hello* e invia il suo certificato. Si scambiano in maniera cifrata la chiave e inizia la comunicazione che da Wireshark non è visibile in chiaro.

Inoltre nella prima figura si evince anche che è stato necessario ritrasmettere un segmento di 59 bytes a dimostrazione di come il protocollo TCP si occupi di far avvenire la corretta trasmissione e ricezione dei pacchetti, reinviandoli in caso di problematiche.



The image shows a Wireshark packet capture window titled "Wireshark - Packet 15 - eth0". The packet list on the left shows packet 15 selected. The packet details pane on the right shows the following structure:

- Acknowledgment number (raw): 3354710057
- 0101 .... = Header Length: 20 bytes (5)
- Flags: 0x018 (PSH, ACK)
- Window: 501
- [Calculated window size: 64128]
- [Window size scaling factor: 128]
- Checksum: 0xc26f [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- [Timestamps]
- [SEQ/ACK analysis]
- TCP payload (59 bytes)
- Retransmitted TCP segment data (59 bytes)

The packet bytes pane at the bottom shows the raw data of the retransmitted segment, starting with 0000 08 00 27 42 2e 08 08 00 27 cb 7e f5 08 00 45 00 and ending with 0070 3b. The corresponding ASCII representation on the right shows the start of the data as "...B... '... E..." and ends with "...VH...".