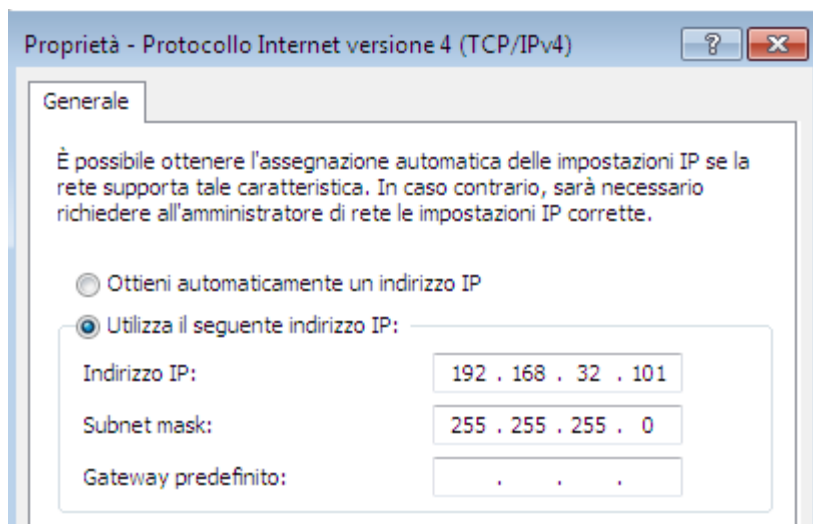


Esercizio Finale M1 – Galli Alessandro

Iniziamo assegnando un indirizzo IP statico a entrambe le macchine virtuali (Windows 7 e Kali Linux). Accediamo alla scheda di rete di Windows 7 tramite pannello di controllo e assegniamo l'IP.



Accediamo alla scheda di rete di Kali utilizzando il comando da terminale

```
sudo nano /etc/network/interfaces
```

Lanciamo poi i seguenti comandi e assegniamo l'IP:

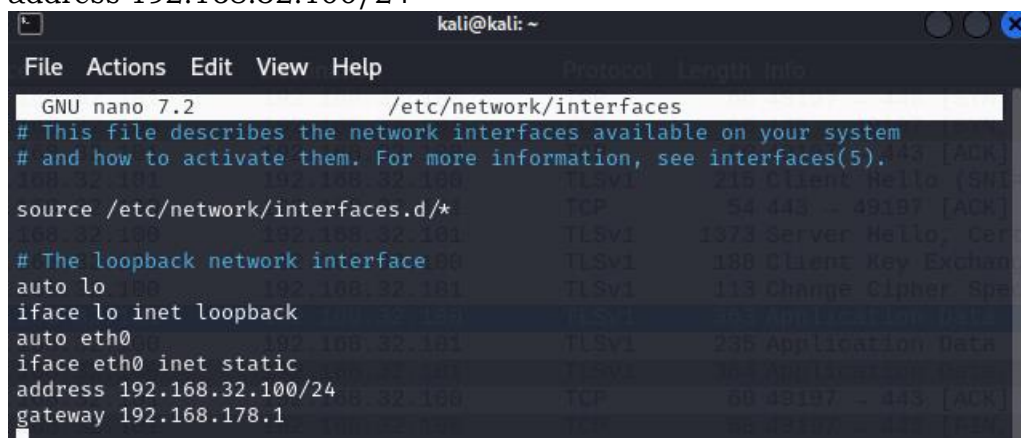
```
auto lo
```

```
iface lo inet loopback
```

```
auto eth0
```

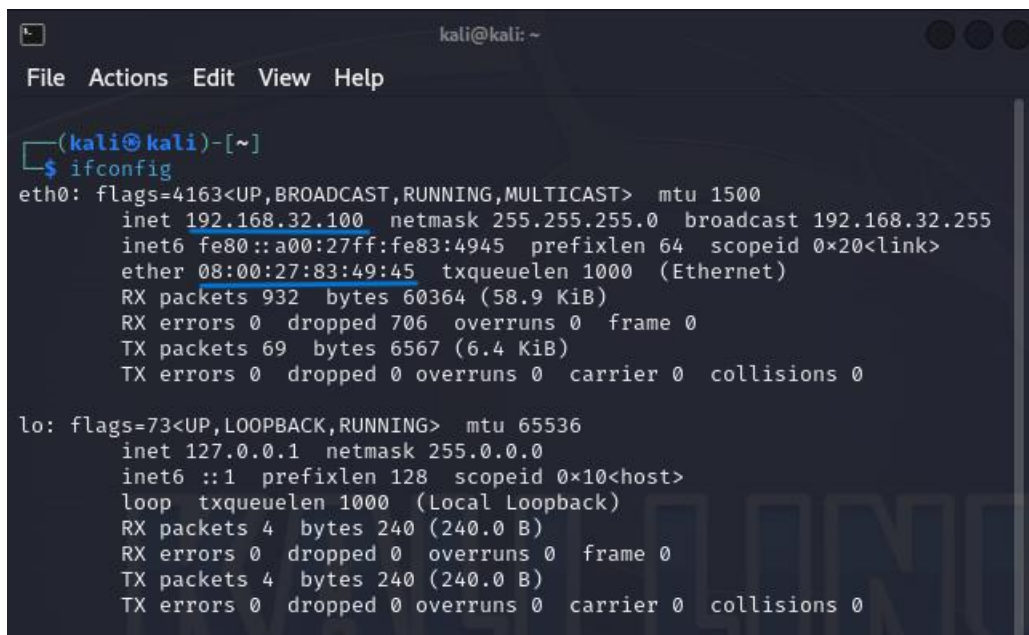
```
iface eth0 inet static
```

```
address 192.168.32.100/24
```



Non sarà necessario impostare un gateway in quanto non abbiamo bisogno di una reale connessione a Internet per questo esercizio.

Lanciamo **ifconfig** per verificare che sia tutto impostato correttamente

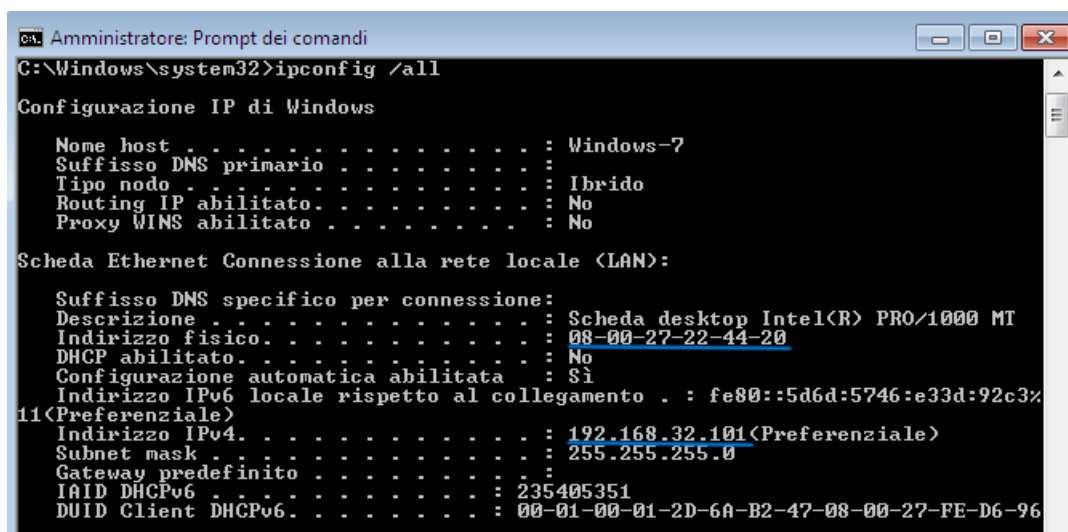


```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.32.100 netmask 255.255.255.0 broadcast 192.168.32.255  
    inet6 fe80::a00:27ff:fe83:4945 prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:83:49:45 txqueuelen 1000 (Ethernet)  
    RX packets 932 bytes 60364 (58.9 KiB)  
    RX errors 0 dropped 706 overruns 0 frame 0  
    TX packets 69 bytes 6567 (6.4 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>  
    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
```

IP Kali = 192.168.32.100

MAC Address Kali = 08:00:27:83:49:45

Lanciamo **ipconfig /all** per verificare che sia tutto impostato correttamente



```
Amministratore: Prompt dei comandi  
C:\Windows\system32>ipconfig /all  
  
Configurazione IP di Windows  
  
Nome host . . . . . : Windows-7  
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. . . . . : 08-00-27-22-44-20  
DHCP abilitato. . . . . : No  
Configurazione automatica abilitata : Si  
Indirizzo IPv6 locale rispetto al collegamento . : fe80::5d6d:5746:e33d:92c3%11(Preferenziale)  
Indirizzo IPv4. . . . . : 192.168.32.101(Preferenziale)  
Subnet mask . . . . . : 255.255.255.0  
Gateway predefinito . . . . . :  
IAID DHCPv6 . . . . . : 235405351  
DUID Client DHCPv6. . . . . : 00-01-00-01-2D-6A-B2-47-08-00-27-FE-D6-96
```

IP Win 7 = 192.168.32.101

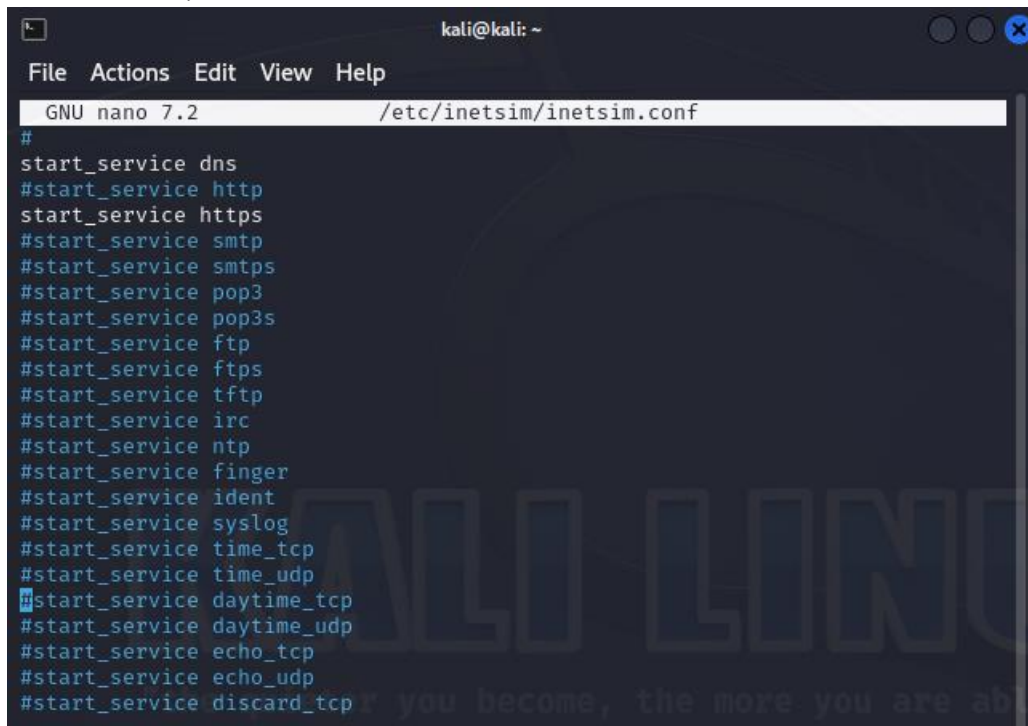
MAC Address Win 7 = 08:00:27:22:44:20

Dalle immagini vediamo sottolineati anche i MAC Address delle due VM, che torneranno utili più avanti nell'esercizio.

Successivamente, entriamo nel pannello di configurazione InetSim per generare traffico fasullo di pacchetti di rete, utilizzando i seguenti passaggi. Cominciamo utilizzando il server HTTPS.

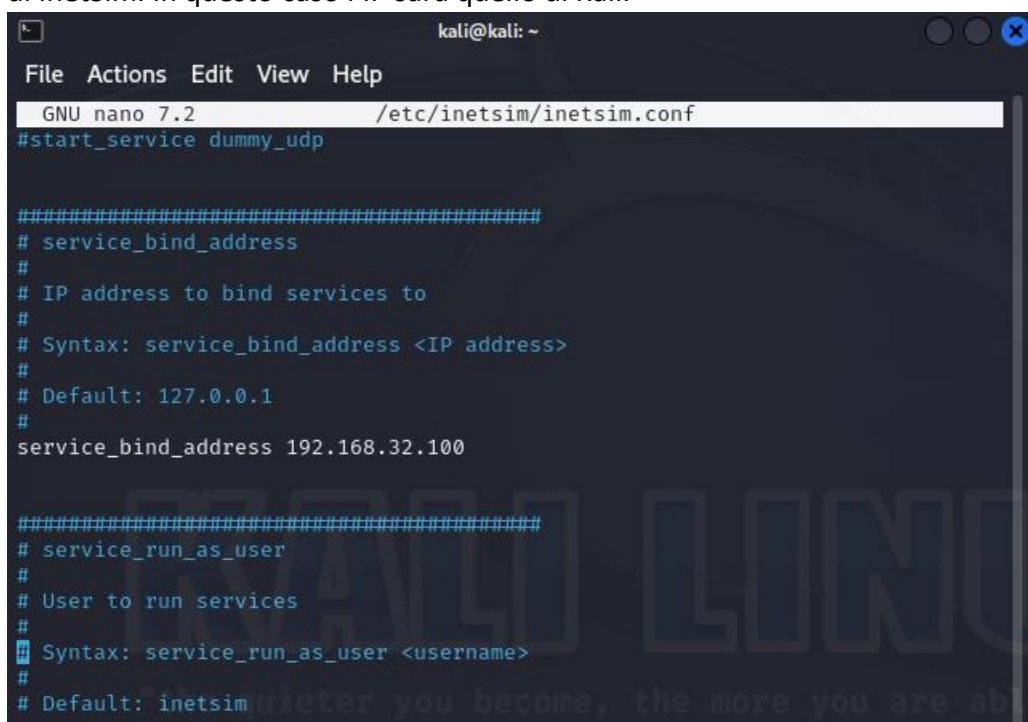
1) Lanciamo il comando `sudo nano /etc/inetsim/inetsim.conf`

2) Scrivendo il cancelletto #, andremo a disattivare tutti i servizi generati da InetSim che non ci interessano, lasciando attivi solo i servizi DNS e HTTPS.



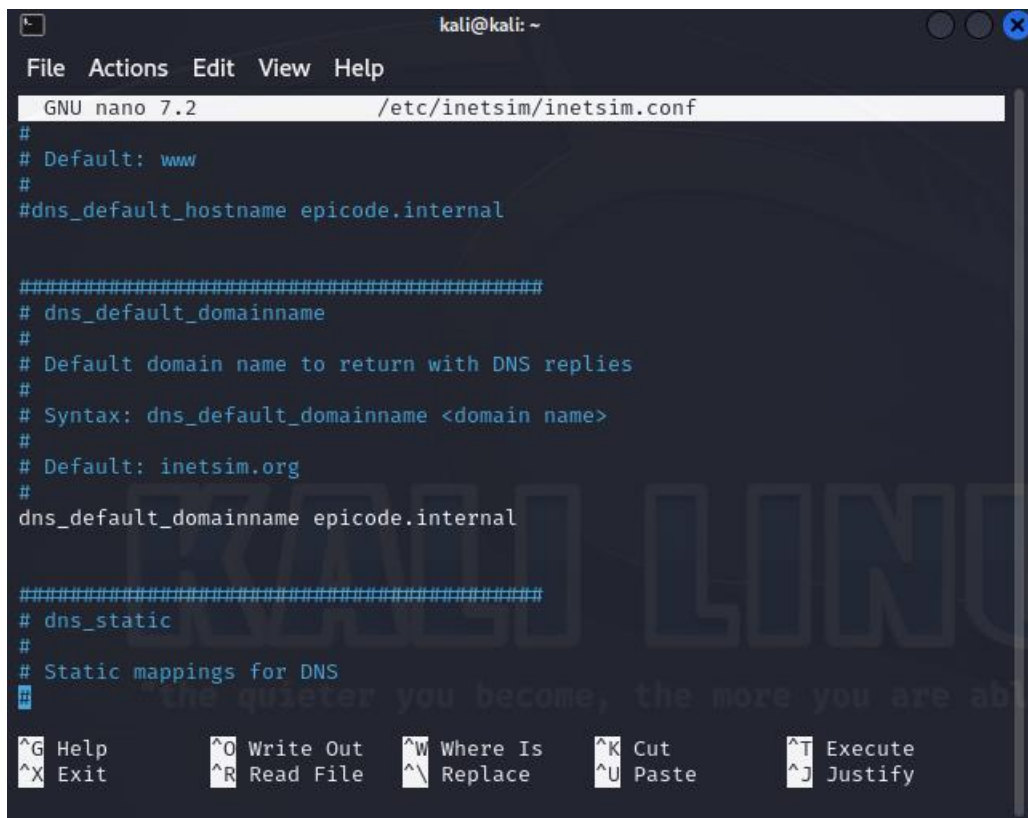
```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetsim/inetsim.conf  
#  
start_service dns  
#start_service http  
start_service https  
#start_service smtp  
#start_service smtps  
#start_service pop3  
#start_service pop3s  
#start_service ftp  
#start_service ftps  
#start_service tftp  
#start_service irc  
#start_service ntp  
#start_service finger  
#start_service ident  
#start_service syslog  
#start_service time_tcp  
#start_service time_udp  
#start_service daytime_tcp  
#start_service daytime_udp  
#start_service echo_tcp  
#start_service echo_udp  
#start_service discard_tcp
```

3) Abilitiamo “Service Bind Address”, che permette di inserire un IP a cui assegnare i servizi di InetSim. In questo caso l’IP sarà quello di Kali.



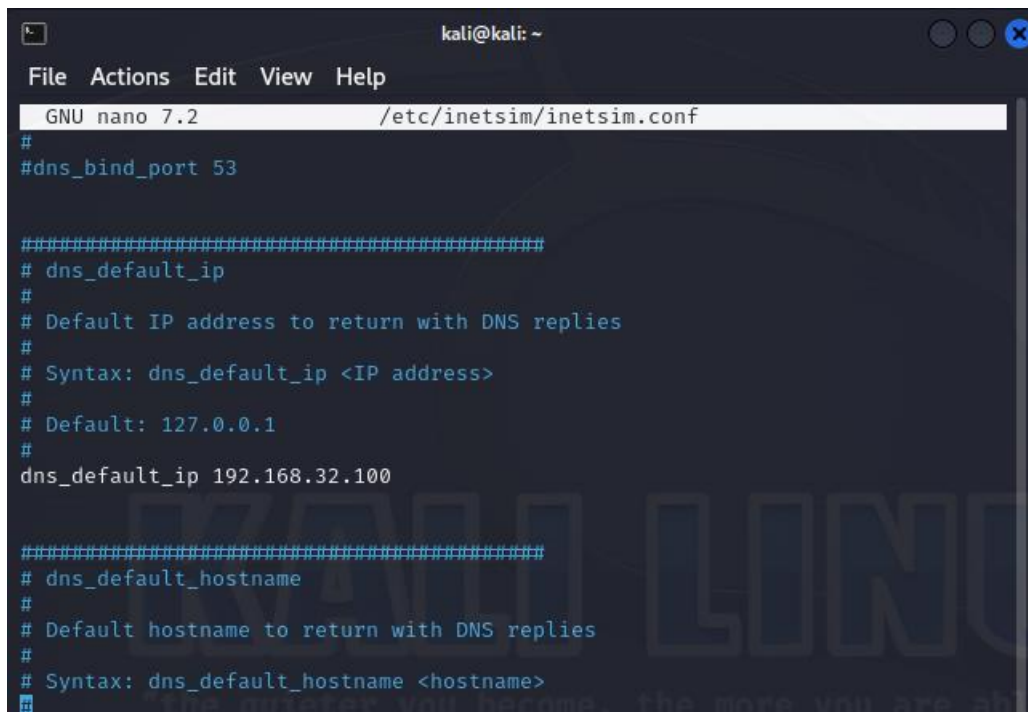
```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetsim/inetsim.conf  
#start_service dummy_udp  
  
#####  
# service_bind_address  
#  
# IP address to bind services to  
#  
# Syntax: service_bind_address <IP address>  
#  
# Default: 127.0.0.1  
#  
service_bind_address 192.168.32.100  
  
#####  
# service_run_as_user  
#  
# User to run services  
#  
# Syntax: service_run_as_user <username>  
#  
# Default: inetsim
```

4) Assegniamo il nome di dominio "epicode.internal" al server DNS di InetSim



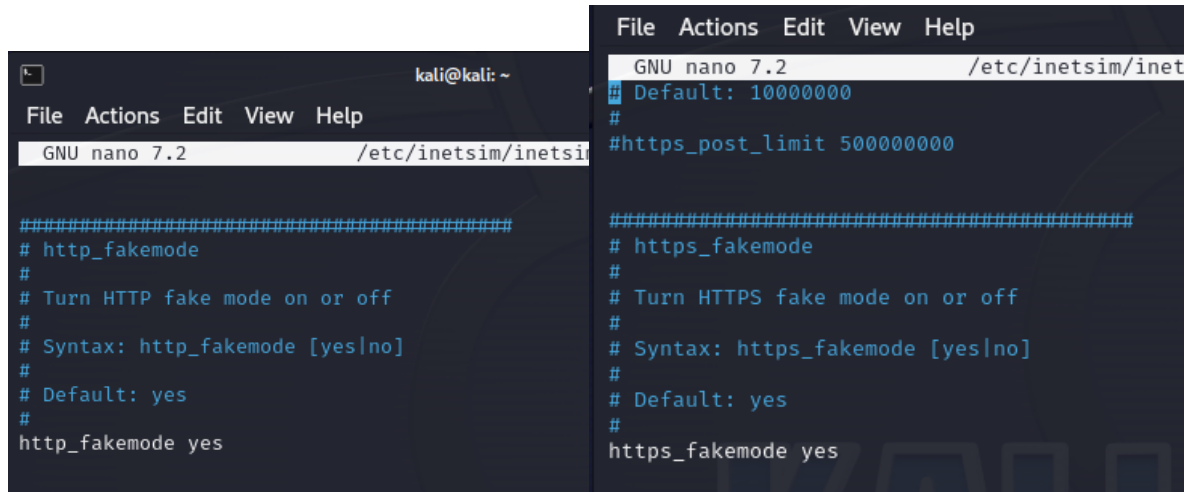
```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetSim/inetSim.conf  
#  
# Default: www  
#  
#dns_default_hostname epicode.internal  
  
#####  
# dns_default_domainname  
#  
# Default domain name to return with DNS replies  
#  
# Syntax: dns_default_domainname <domain name>  
#  
# Default: inetSim.org  
#  
dns_default_domainname epicode.internal  
  
#####  
# dns_static  
#  
# Static mappings for DNS  
#
```

5) Assegniamo l'indirizzo IP di Kali al DNS, che sarà l'IP del server HTTPS in cui verrà tradotto/risolto il nome di dominio inserito prima (epicode.internal). Usiamo l'IP di Kali dato che il servizio HTTPS è generato da InetSim, i cui servizi sono assegnati all' IP di Kali.



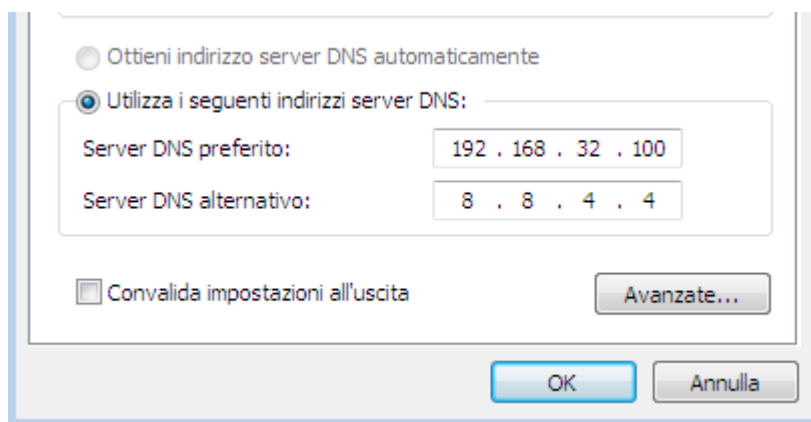
```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetSim/inetSim.conf  
#  
#dns_bind_port 53  
  
#####  
# dns_default_ip  
#  
# Default IP address to return with DNS replies  
#  
# Syntax: dns_default_ip <IP address>  
#  
# Default: 127.0.0.1  
#  
dns_default_ip 192.168.32.100  
  
#####  
# dns_default_hostname  
#  
# Default hostname to return with DNS replies  
#  
# Syntax: dns_default_hostname <hostname>  
#
```

6) Attiviamo la fake mode di HTTPS e HTTP dando il comando “yes”, per permettere ad InetSim di generare un file HTML il quale dovrà uscire a schermo scrivendo “epicode.internal” sul Web Browser di Windows 7, confermandoci il funzionamento dei due server.

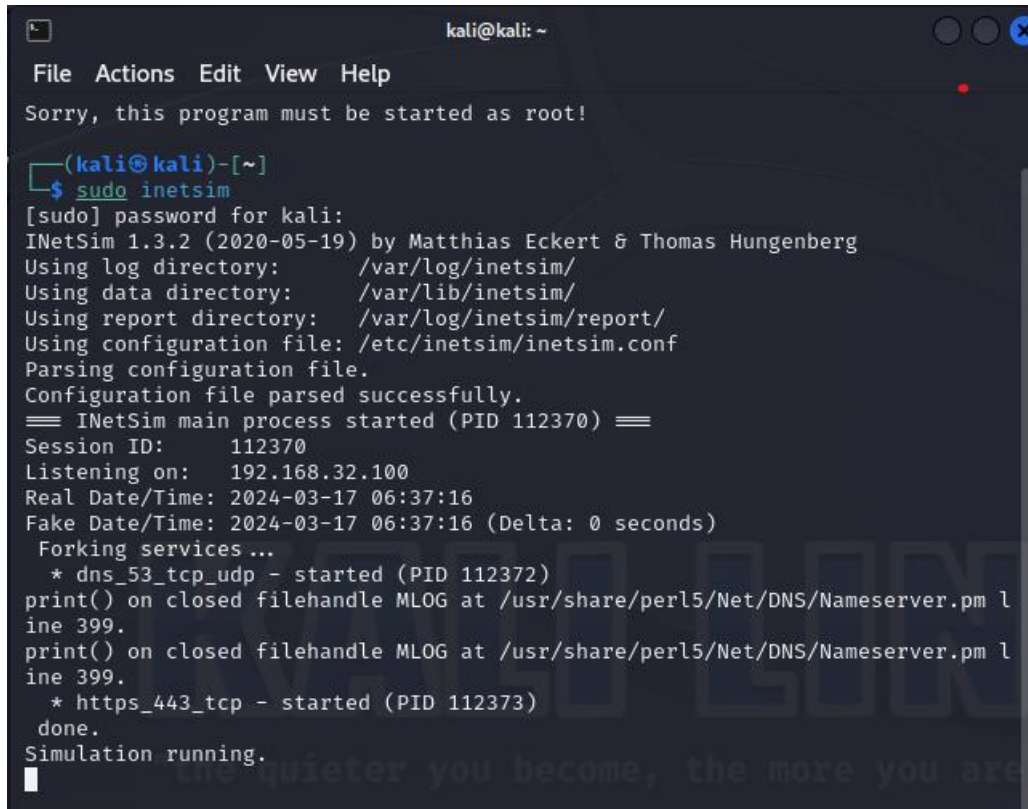


```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetsim/inetsi  
  
#####  
# http_fakemode  
#  
# Turn HTTP fake mode on or off  
#  
# Syntax: http_fakemode [yes|no]  
#  
# Default: yes  
#  
http_fakemode yes  
  
File Actions Edit View Help  
GNU nano 7.2 /etc/inetsim/inet  
# Default: 10000000  
#  
#https_post_limit 500000000  
  
#####  
# https_fakemode  
#  
# Turn HTTPS fake mode on or off  
#  
# Syntax: https_fakemode [yes|no]  
#  
# Default: yes  
#  
https_fakemode yes
```

La configurazione di InetSim è terminata. Ora Impostiamo come server DNS su Windows 7 l’indirizzo IP di Kali, dato che il servizio DNS è generato da InetSim che assegna i servizi all’IP Kali. Possiamo farlo accedendo alle impostazioni della scheda di rete a cui abbiamo assegnato l’indirizzo IP statico in precedenza.



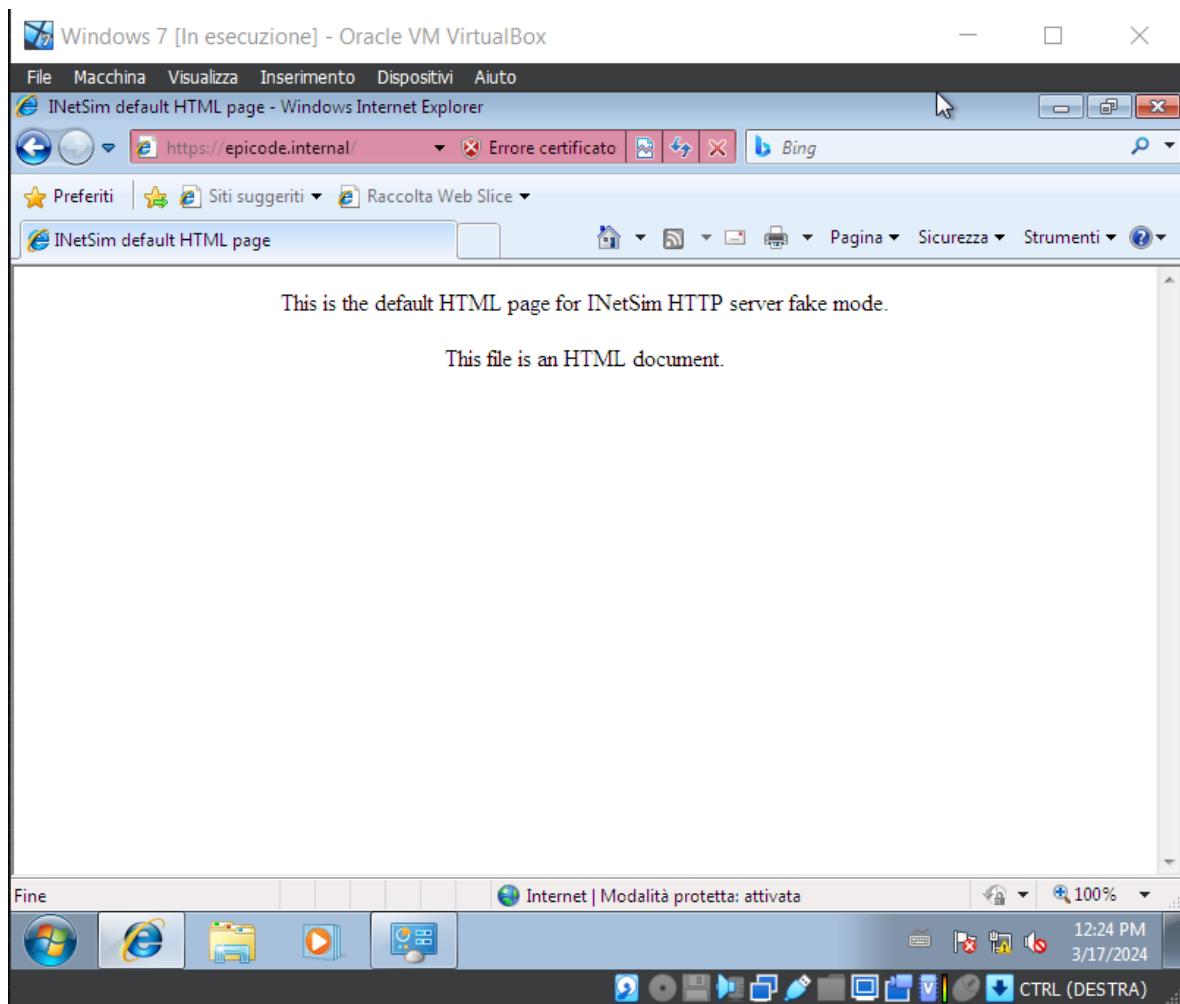
Ora possiamo eseguire InetSim su Kali per poter utilizzare i suoi servizi. Lanciamo il comando **sudo inetsim**



```
kali@kali: ~  
File Actions Edit View Help  
Sorry, this program must be started as root!  
  
(kali@kali)-[~]  
$ sudo inetsim  
[sudo] password for kali:  
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 112370) ==  
Session ID: 112370  
Listening on: 192.168.32.100  
Real Date/Time: 2024-03-17 06:37:16  
Fake Date/Time: 2024-03-17 06:37:16 (Delta: 0 seconds)  
Forking services ...  
* dns_53_tcp_udp - started (PID 112372)  
print() on closed filehandle MLOG at /usr/share/perl5/Net/DNS/Nameserver.pm line 399.  
print() on closed filehandle MLOG at /usr/share/perl5/Net/DNS/Nameserver.pm line 399.  
* https_443_tcp - started (PID 112373)  
done.  
Simulation running.  
█
```

Come vediamo dallo screenshot, sono in esecuzione i servizi DNS e HTTPS. (Continua alla pagina successiva)

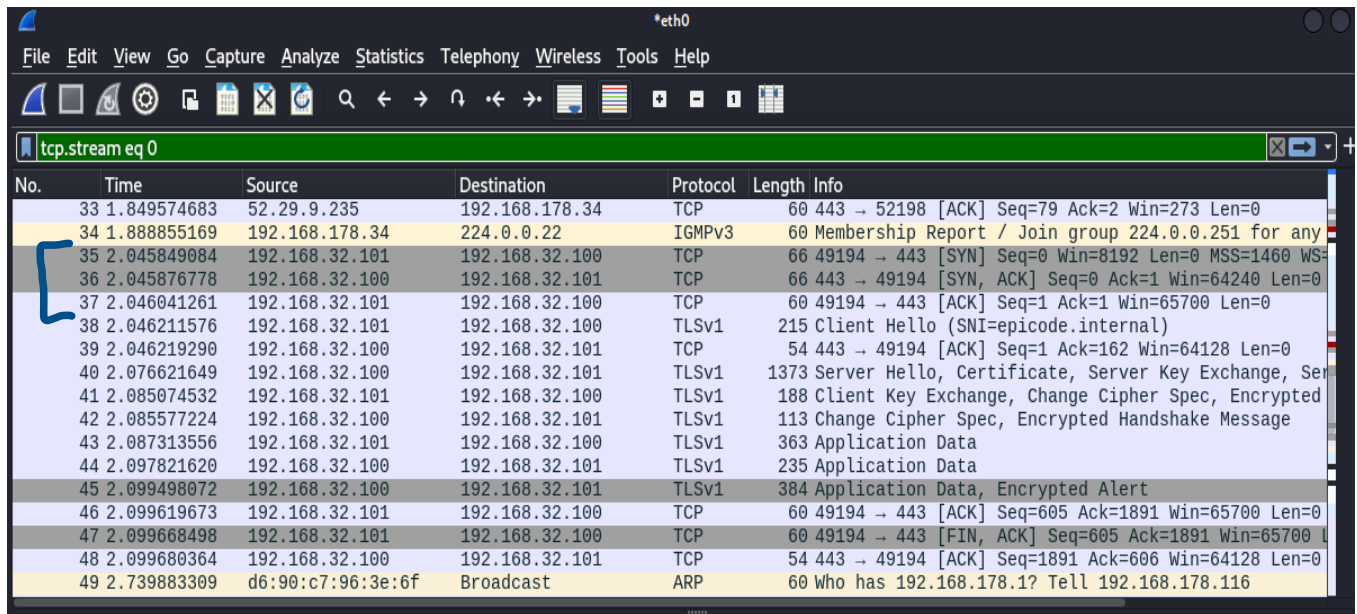
Andiamo su Internet Explorer da Windows 7 e digitiamo sulla barra di ricerca “epicode.internal”, per controllare che DNS e HTTPS siano configurati correttamente.



Il contenuto della richiesta HTTPS definito dal file HTML su InetSim (vedi immagine sotto) risulta visualizzato a schermo su Internet Explorer di Windows 7.

```
#####  
# https_default_fakefile  
#  
# The default fake file returned in fake mode if the file extension  
# in the HTTPS request does not match any of the extensions  
# defined above.  
#  
# The default fake file must be placed in <data-dir>/http/fakefiles  
#  
# Syntax: https_default_fakefile <filename> <mime-type>  
#  
# Default: none  
#  
https_default_fakefile sample.html text/html
```

Procediamo intercettando con WireShark i pacchetti instradati da InetSim, tramite la scheda di rete *eth0*. Come vediamo dallo screenshot sotto, vediamo gli IP del mittente e destinatario che ci fanno capire che è stato sniffato traffico di rete fra Kali (192.168.32.100) e Windows 7 (192.168.32.100). Inoltre, possiamo vedere che l'HTTPS utilizza il protocollo TLS (Transport Layer Security), che è utile a cifrare il messaggio. Vediamo anche che il three-way-handshake è andato a buon fine, guardando i valori di **Seq** e **Ack** ai pacchetti 35, 36 e 37. (**Seq** è assegnato a un valore casuale, mentre **Ack** corrisponde a **Seq** del pacchetto precedente + 1).



No.	Time	Source	Destination	Protocol	Length	Info
33	1.849574683	52.29.9.235	192.168.178.34	TCP	60	443 → 52198 [ACK] Seq=79 Ack=2 Win=273 Len=0
34	1.888855169	192.168.178.34	224.0.0.22	IGMPv3	60	Membership Report / Join group 224.0.0.251 for any
35	2.045849084	192.168.32.101	192.168.32.100	TCP	66	49194 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=0
36	2.045876778	192.168.32.100	192.168.32.101	TCP	66	443 → 49194 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
37	2.046041261	192.168.32.101	192.168.32.100	TCP	60	49194 → 443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
38	2.046211576	192.168.32.101	192.168.32.100	TLSv1	215	Client Hello (SNI=epicode.internal)
39	2.046219290	192.168.32.100	192.168.32.101	TCP	54	443 → 49194 [ACK] Seq=1 Ack=162 Win=64128 Len=0
40	2.076621649	192.168.32.100	192.168.32.101	TLSv1	1373	Server Hello, Certificate, Server Key Exchange, Ser
41	2.085074532	192.168.32.101	192.168.32.100	TLSv1	188	Client Key Exchange, Change Cipher Spec, Encrypted
42	2.085577224	192.168.32.100	192.168.32.101	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
43	2.087313556	192.168.32.101	192.168.32.100	TLSv1	363	Application Data
44	2.097821620	192.168.32.100	192.168.32.101	TLSv1	235	Application Data
45	2.099498072	192.168.32.100	192.168.32.101	TLSv1	384	Application Data, Encrypted Alert
46	2.099619673	192.168.32.101	192.168.32.100	TCP	60	49194 → 443 [ACK] Seq=605 Ack=1891 Win=65700 Len=0
47	2.099668498	192.168.32.101	192.168.32.100	TCP	60	49194 → 443 [FIN, ACK] Seq=605 Ack=1891 Win=65700 L
48	2.099680364	192.168.32.100	192.168.32.101	TCP	54	443 → 49194 [ACK] Seq=1891 Ack=606 Win=64128 Len=0
49	2.739883309	d6:90:c7:96:3e:6f	Broadcast	ARP	60	Who has 192.168.178.1? Tell 192.168.178.116

Come vediamo dalle frecce rosse nel secondo screenshot, il MAC Address di destinazione coincide con quello mostrato dal prompt di Windows (**vedi screenshot a pagina 2**), in maniera analoga al MAC Address “mittente” mostrato dal terminale di Kali.

39	2.046219290	192.168.32.100	192.168.32.101	TCP
40	2.076621649	192.168.32.100	192.168.32.101	TLSv1
41	2.085074532	192.168.32.101	192.168.32.100	TLSv1
42	2.085577224	192.168.32.100	192.168.32.101	TLSv1
43	2.087313556	192.168.32.101	192.168.32.100	TLSv1
44	2.097821620	192.168.32.100	192.168.32.101	TLSv1
45	2.099498072	192.168.32.100	192.168.32.101	TLSv1
46	2.099619673	192.168.32.101	192.168.32.100	TCP
47	2.099668498	192.168.32.101	192.168.32.100	TCP
48	2.099680364	192.168.32.100	192.168.32.101	TCP
49	2.739883309	d6:90:c7:96:3e:6f	Broadcast	ARP


```

> Frame 40: 1373 bytes on wire (10984 bits), 1373 bytes captured (10984 b
> Ethernet II, Src: PCSSystemtec_83:49:45 (08:00:27:83:49:45), Dst: PCSSy
  > Destination: PCSSystemtec_22:44:20 (08:00:27:22:44:20) ←
  > Source: PCSSystemtec_83:49:45 (08:00:27:83:49:45) ←
    Type: IPv4 (0x0800)
  > Internet Protocol Version 4, Src: 192.168.32.100, Dst: 192.168.32.101
  > Transmission Control Protocol, Src Port: 443, Dst Port: 49194, Seq: 1,
  > Transport Layer Security
  
```


Possiamo notare che succede esattamente l'inverso per i pacchetti inviati da Windows:

The image shows a Wireshark packet capture. The packet list on the left highlights frame 41, which is a TLSv1 packet. The details pane on the right shows the structure of this packet: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Transport Layer Security. Red arrows point to the destination IP (192.168.32.101) and the destination port (443) in the TCP details.

No.	Time	Source	Destination	Protocol
40	2.076621649	192.168.32.100	192.168.32.101	TLSv1
41	2.085074532	192.168.32.101	192.168.32.100	TLSv1
42	2.085577224	192.168.32.100	192.168.32.101	TLSv1
43	2.087313556	192.168.32.101	192.168.32.100	TLSv1
44	2.097821620	192.168.32.100	192.168.32.101	TLSv1
45	2.099498072	192.168.32.100	192.168.32.101	TLSv1
46	2.099619673	192.168.32.101	192.168.32.100	TCP
47	2.099668498	192.168.32.101	192.168.32.100	TCP
48	2.099680364	192.168.32.100	192.168.32.101	TCP
49	2.739883309	d6:90:c7:96:3e:6f	Broadcast	ARP

Frame 41: 188 bytes on wire (1504 bits), 188 bytes captured (1504 bits) on interface 0
Ethernet II, Src: PCSSystemtec_22:44:20 (08:00:27:22:44:20), Dst: PCSSystemtec_83:49:45 (08:00:27:83:49:45)
Destination: PCSSystemtec_83:49:45 (08:00:27:83:49:45)
Source: PCSSystemtec_22:44:20 (08:00:27:22:44:20)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100
Transmission Control Protocol, Src Port: 49194, Dst Port: 443, Seq: 162
Transport Layer Security

Per mostrare il contenuto della richiesta HTTPS, selezioniamo uno dei pacchetti del traffico fra Win 7 e Kali, poi facciamo **click destro > Follow > Follow TCP** e vediamo che il contenuto del pacchetto è cifrato, visto che vediamo a schermo numeri, lettere e segni che sembrano non avere alcun senso ma che in realtà simboleggiano una cifratura, rispettando così lo scopo del servizio HTTPS:

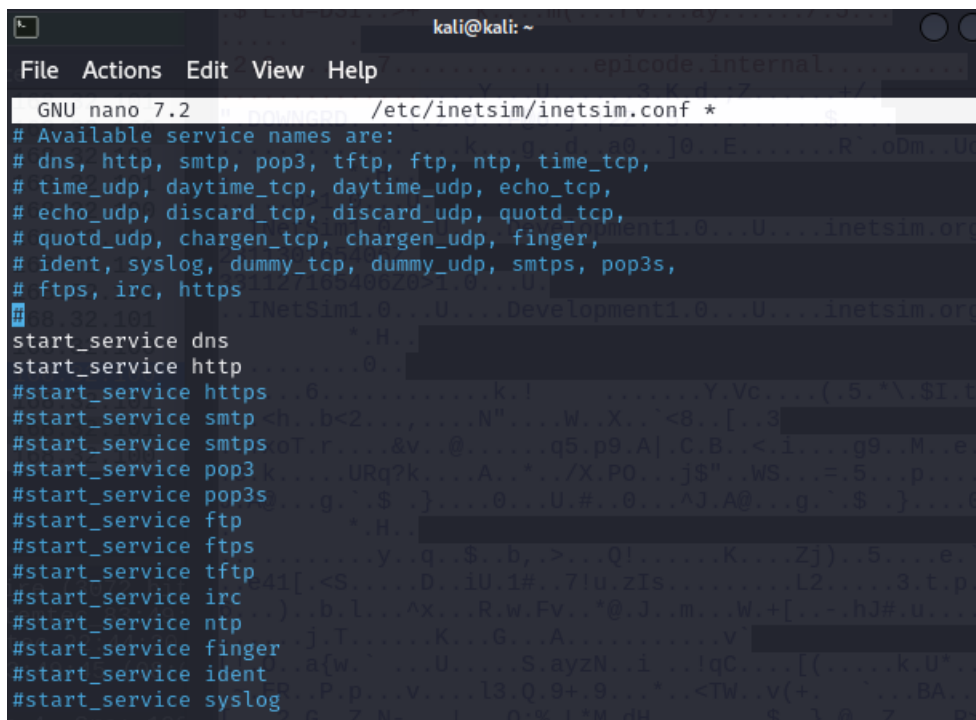
The image shows the 'Follow TCP Stream' view in Wireshark. The data is displayed as a series of hexadecimal characters, indicating that the content is encrypted. A red arrow points to the start of the data stream.

Wireshark - Follow TCP Stream (tcp.stream eq 2) - eth0

.....e...r...F...C...)....0.#....{.Z
\$.L.d=D3I...>+ k...m(...:rV...ay.../..5...
.....
2.8.....7.....epicode.internal.....
.....Y...U.....3.K.d;Z.....+/.
".DOWNGRD...{.z.o.F@6..}|2Z..5.....\$.
.....k...g.d..a0..]0..E.....R`oDm..Uo.=Cq..A..u0
.....*..H...
.....0>1.0...U...
..INetSim1.0...U...Development1.0...U...inetsim.org0..
231130165406Z..
331127165406Z0>1.0...U...
..INetSim1.0...U...Development1.0...U...inetsim.org0.."0
.....*..H...
.....0..
.....6.....k!.....Y.Vc....(.5.*\\$.I.t.t.%0+...k...sk.s
...<h..b<2....N"...W..X..`<8..[.3
..YxoT.r...&v...@...q5.p9.A|.C.B.<.i....g9..M.e.=..h)...FJ..|8.....pI..k..]
5.k....URq?k....A..*/X.PO...j\$".WS...=.5...p.....c.@.....?..S0Q0...U.....^
J.A@...g...\$..}....0...U..#...0...^J.A@...g...\$..}....0...U.....0...0
.....*..H...
.....y.q...\$.b..>..Q!.....K...Zj)..5....e...s....wt...:N...!72...5.....
..e41[.<S....D..iU.1#..7!u.zIs.....L2....3.t.p.C..8....2.....@.,y...`L..N.s^a
6....).b.l...^x...R.w.Fv..*@.J.m...W.+[...hJ#.u...F...:..ifv.-<"r.%.....;g...f.
.....j..T.....K...R...A.....V`
L!.O..a{w...`...U....S.ayzN..i...!qC...[(...k.U*..C....v..v.P.K.#eT
~.ER..P.p...v...l3.Q.9+.9...*...<TW..v(+...`...BA...R>cV..Ld+7H..y.J...B.4,
[...2.G..Z.N-...!..Q;%L*M.dH...\$.}.@.Z...R>w.....m|.CT.....B\$X
.....r.P...A.....R..7Ac...g...lq.....)....@p..l.%32.y.1.4.[W...)
.....F...BA.x...GK...3.%..[>0.<m[\$...7p...9{c...?..J...Z.E.....{.....
.....0WAX...dc..(.0...X.x.M...W*a..s..6.56..
YW,@Y.....0...n...d...+...K.8.....[.z.'..w....W2...0...z..F...0..
A.q1r..
..P...7.>..>..G.....zs...<..y.\.a.G.....Y..Mc.[.'..p7..B66..6..b.....B...q.....l..
A>th.U...F...s8.Yo...I..f..U0...(!..i...P..F.E...*...y..bH@)3.....^
3 client pkts, 4 server pkts, 5 turns.

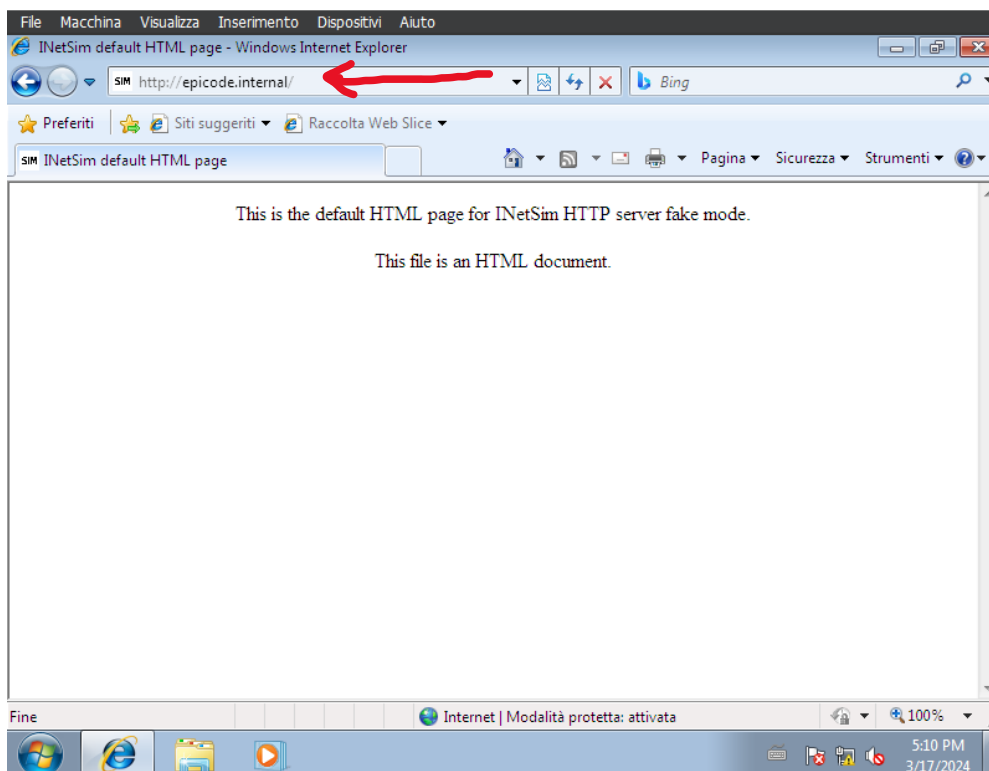
Entire conversation (2,493 bytes) Show data as ASCII Stream 2

Passiamo ora al protocollo HTTP tornando alla configurazione di InetSim. Lasciamo tutto invariato tranne i servizi, disabilitando l'HTTPS e abilitando l'HTTP. Dopodiché facciamo ripartire InetSim:



```
kali@kali: ~  
File Actions Edit View Help .....epicode.internal.....  
GNU nano 7.2 /etc/inetsim/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
```

Procediamo, come prima, a scrivere “epicode.internal” sul web browser di Windows 7 ma modificando in HTTP anziché HTTPS. Vediamo che il risultato che ci appare a schermo è lo stesso file HTML di default utilizzato anche dal protocollo HTTPS:



Tornando su Wireshark, intercettiamo nuovamente i pacchetti di rete tramite scheda *eth0*. Come visto prima, gli indirizzi IP mittente e destinatario coincidono, così come i MAC in entrambe le casistiche (**vedi screenshot a pagina 2**). Anche il three-way-handshake è andato a buon fine ai pacchetti 260-261-262, utilizzando lo stesso procedimento dell'HTTPS descritto in precedenza.

No.	Time	Source	Destination	Protocol	Length	Info
257	17.689122533	AVMAudiovisu_49:6e:...	Broadcast	0x8912	60	Ethernet II
258	18.010129688	PCSSystemtec_22:44:...	Broadcast	ARP	60	Who has 192.168.32.100? Tell 192.168.32.101
259	18.010143185	PCSSystemtec_83:49:...	PCSSystemtec_22:44:...	ARP	42	192.168.32.100 is at 08:00:27:83:49:45
260	18.010244516	192.168.32.101	192.168.32.100	TCP	66	49228 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4
261	18.010263345	192.168.32.100	192.168.32.101	TCP	66	80 → 49228 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
262	18.010402345	192.168.32.101	192.168.32.100	TCP	60	49228 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
263	18.010452140	192.168.32.101	192.168.32.100	HTTP	333	GET / HTTP/1.1
264	18.010459303	192.168.32.100	192.168.32.101	TCP	54	80 → 49228 [ACK] Seq=1 Ack=280 Win=64128 Len=0
265	18.023446682	192.168.32.100	192.168.32.101	TCP	204	80 → 49228 [PSH, ACK] Seq=1 Ack=280 Win=64128 Len=1
266	18.025017810	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
267	18.025197431	192.168.32.101	192.168.32.100	TCP	60	49228 → 80 [ACK] Seq=280 Ack=410 Win=65292 Len=0
268	18.025265075	192.168.32.101	192.168.32.100	TCP	60	49228 → 80 [FIN, ACK] Seq=280 Ack=410 Win=65292 Len=0
269	18.025274249	192.168.32.100	192.168.32.101	TCP	54	80 → 49228 [ACK] Seq=410 Ack=281 Win=64128 Len=0
270	19.691881919	AVMAudiovisu_49:6e:...	AtherosCommu_00:00:...	HomePL...	60	Qualcomm Atheros, OP_ATTR.REQ (Get Device Attribute)
271	19.692545912	AVMAudiovisu_49:6e:...	Broadcast	HomePL...	60	Qualcomm Atheros, GET_SW.REQ (Get Device/SW Version)
272	19.693018641	AVMAudiovisu_49:6e:...	Broadcast	0x8912	60	Ethernet II
273	20.027367269	192.168.178.117	192.168.178.255	UDP	416	53401 → 40777 Len=374

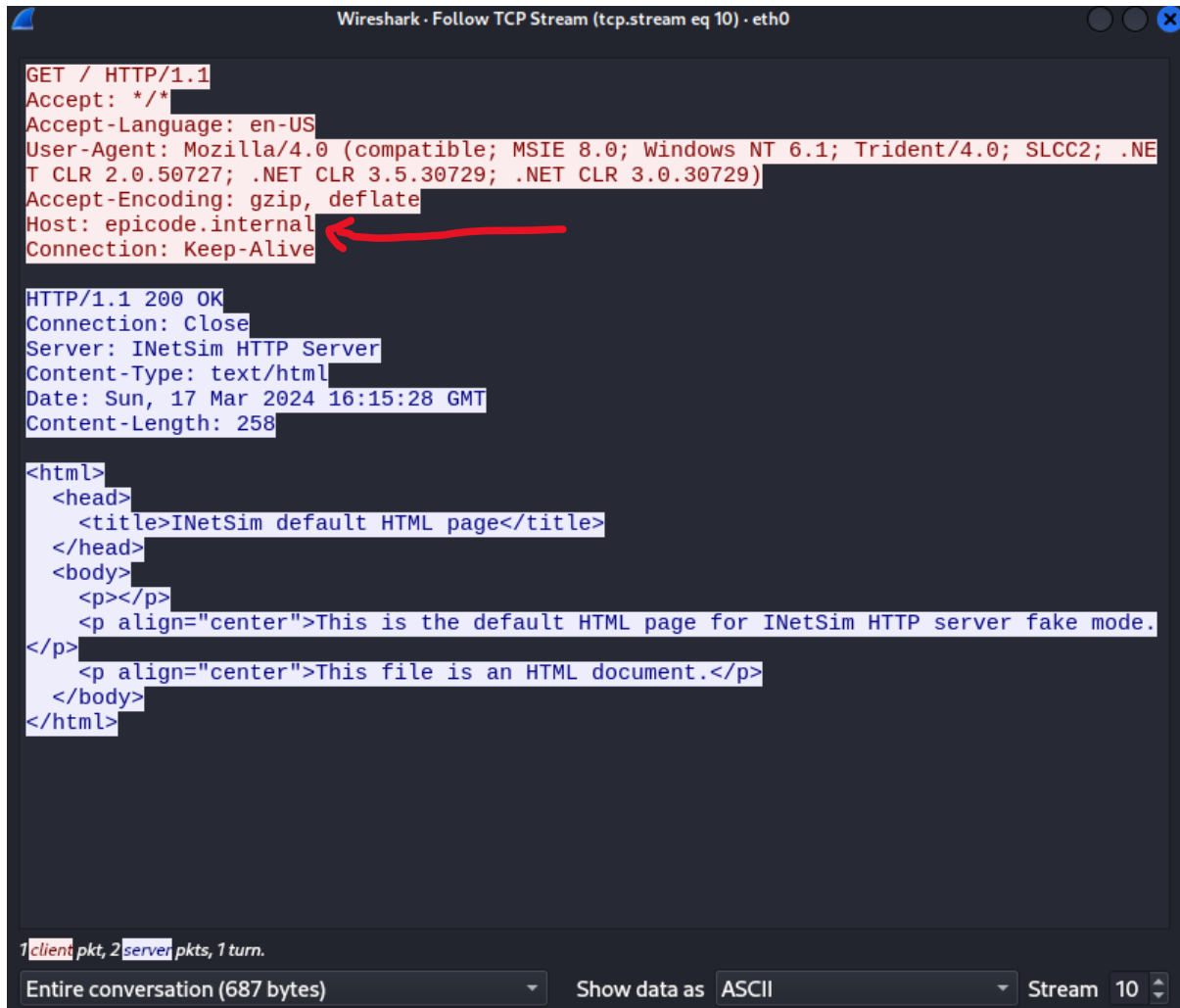
267	18.025197431	192.168.32.101	192.168.32.100	TCP
268	18.025265075	192.168.32.101	192.168.32.100	TCP
269	18.025274249	192.168.32.100	192.168.32.101	TCP
270	19.691881919	AVMAudiovisu_49:6e:...	AtherosCommu_00:00:...	HomePL...
271	19.692545912	AVMAudiovisu_49:6e:...	Broadcast	HomePL...
272	19.693018641	AVMAudiovisu_49:6e:...	Broadcast	0x8912
273	20.027367269	192.168.178.117	192.168.178.255	UDP

▶ Frame 268: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) or
 ▼ Ethernet II, Src: PCSSystemtec_22:44:20 (08:00:27:22:44:20), Dst: PCSSy
 ▶ Destination: PCSSystemtec_83:49:45 (08:00:27:83:49:45)
 ▶ Source: PCSSystemtec_22:44:20 (08:00:27:22:44:20)

267	18.025197431	192.168.32.101	192.168.32.100	TCP
268	18.025265075	192.168.32.101	192.168.32.100	TCP
269	18.025274249	192.168.32.100	192.168.32.101	TCP
270	19.691881919	AVMAudiovisu_49:6e:...	AtherosCommu_00:00:...	HomePL...
271	19.692545912	AVMAudiovisu_49:6e:...	Broadcast	HomePL...
272	19.693018641	AVMAudiovisu_49:6e:...	Broadcast	0x8912
273	20.027367269	192.168.178.117	192.168.178.255	UDP

▶ Frame 269: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) or
 ▼ Ethernet II, Src: PCSSystemtec_83:49:45 (08:00:27:83:49:45), Dst: PCSSy
 ▶ Destination: PCSSystemtec_22:44:20 (08:00:27:22:44:20)
 ▶ Source: PCSSystemtec_83:49:45 (08:00:27:83:49:45)

Per mostrare il contenuto della richiesta HTTP, selezioniamo uno dei pacchetti del traffico fra Win 7 e Kali, poi facciamo **clic destro > Follow > Follow TCP**. Possiamo notare che il pacchetto non è stato cifrato a differenza dell'HTTPS, visto che riusciamo a vedere l'intero contenuto, che contiene informazioni come data, ora e soprattutto lo script HTML che manda a schermo la frase sul web browser di Windows 7 (è lo stesso script fornito dal servizio HTTPS, che abbiamo visto in precedenza).



The image shows a Wireshark window titled "Wireshark - Follow TCP Stream (tcp.stream eq 10) - eth0". The window displays the details of an HTTP transaction. The request is a GET / HTTP/1.1 with various headers including Accept, Accept-Language, User-Agent, Accept-Encoding, Host (epicode.internal), and Connection: Keep-Alive. A red arrow points to the Host header. The response is an HTTP/1.1 200 OK from INetSim HTTP Server, with headers for Connection: Close, Content-Type: text/html, Date, and Content-Length: 258. The body of the response is an HTML document with a title "INetSim default HTML page" and two paragraphs of text. The status bar at the bottom indicates "1 client pkt, 2 server pkts, 1 turn." and shows the data as ASCII in a stream view.

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-US
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)
Accept-Encoding: gzip, deflate
Host: epicode.internal
Connection: Keep-Alive

HTTP/1.1 200 OK
Connection: Close
Server: INetSim HTTP Server
Content-Type: text/html
Date: Sun, 17 Mar 2024 16:15:28 GMT
Content-Length: 258

<html>
  <head>
    <title>INetSim default HTML page</title>
  </head>
  <body>
    <p></p>
    <p align="center">This is the default HTML page for INetSim HTTP server fake mode.</p>
    <p align="center">This file is an HTML document.</p>
  </body>
</html>
```

Riassumiamo le differenze fra HTTP e HTTPS:

- I pacchetti vengono intercettati da WireShark in entrambi i casi, ma nel caso dell'HTTP è possibile vederne il contenuto in chiaro, a differenza dell'HTTPS che lo cifra, come dimostrato dai precedenti screenshot.
- L'HTTPS utilizza il protocollo TLS (Transport Layer Security) per cifrare i pacchetti scambiati fra client e server, come visto dai precedenti screenshot.

L'HTTP risulta quindi molto rischioso se si vogliono inviare dati sensibili.