

REPORT PROGETTO

In questo report spiegherò come creare in un ambiente virtuale su VirtualBox un architettura client server tra una macchina virtuale linux (kali), che sarà il nostro server e windows 7 (che sarà il nostro client).

Per prima cosa si cambia l'indirizzo ip sulla macchina linux:

```

L$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.50.100 netmask 255.255.255.0 broadcast 192.168.50.255
    FileS inet6 fe80::a00:27ff:fec7:e136 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c7:e1:36 txqueuelen 1000 (Ethernet)
    RX packets 76 bytes 8730 (8.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 18 bytes 2564 (2.5 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

```

Con il comando **sudo nano /etc/network/interfaces** sul cmd cambiamo l'indirizzo ip che ci viene richiesto:

```

# 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

```

Cambiamo anche l'indirizzo ip su windows 7 andando a inserire anche l'ip del DNS statico:

...

Andiamo a settare InetSim su Kali. InetSim che è tool open source per la simulazione di servizi Internet standard, come DNS, HTTP / HTTPS e così via. Anche se noi utilizzeremo principalmente questi tre nel progetto.

Usiamo il comando **sudo nano /etc/inetsim/inetsim.config** per andare a configurare il nostro server. Andiamo quindi a cambiare queste due righe di comando con l'ip che abbiamo associato a kali:

```
#####  
# 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
```

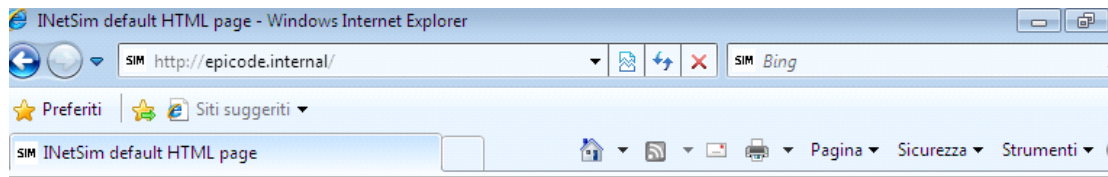
```
#####  
# dns_default_ip  
#  
# Default IP address to return with DNS replies  
#  
# Syntax: dns_default_ip <IP address>  
#  
# Default: 127.0.0.1  
#  
dns_default_ip 192.168.32.100
```

Succesivamente andiamo ad associare il nome **epicode.internal** al dns statico associato all'ip del server cioè **193.168.32.100**:

```
#####  
# 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
```

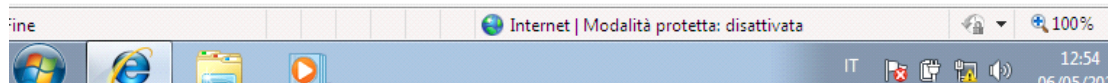
Ora che abbiamo completato l'operazione di configurazione del nostro server dns facciamo eseguire il programma InetSim con il comando: **sudo inetsim** .

Passiamo quindi a windows e apriamo il nostro browser web e cerchiamo il nostro server a cui chiederemo un servizio:



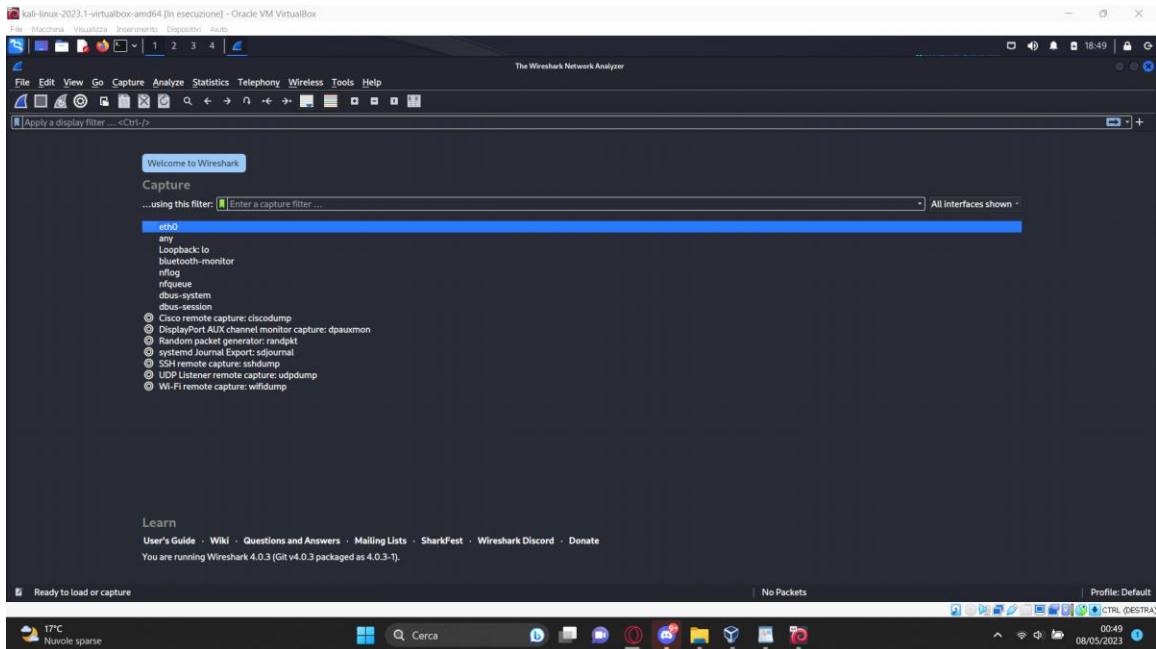
This is the default HTML page for INetSim HTTP server fake mode.

This file is an HTML document.

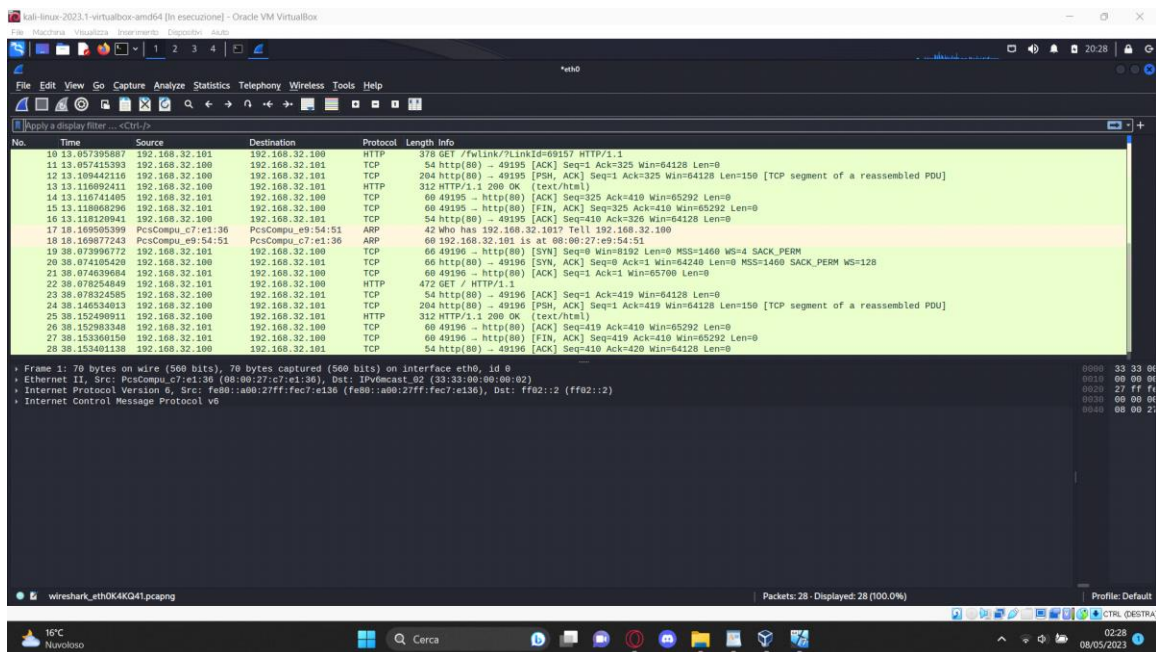


Questo nostro server ci restituisce una pagina in **HTML**.

Ora andremmo ad osservare il traffico di dati tra il client e il server tramite un applicazione che si occupa dello **sniffing** : **Wireshark** .



Prima di cercare di cercare il server clicchiamo su **Start capturing packets** per iniziare la cattura dei pacchetti. Dopo la ricerca sul browser otterremo la seguente schermata:



Ora nella riga di ricerca andremmo a cercare nello specifico i pacchetti del protocollo **HTTP** immettendo il seguente codice:

```
ip.addr == 192.168.32.101 && tcp.port == 80
```

Ci restituirà la seguente schermata:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.32.101	192.168.32.100	TCP	60	49215 → http(80) [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM
2	0.000045790	192.168.32.100	192.168.32.101	TCP	60	http(80) → 49215 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 SACK_PERM WS=128
3	0.000303478	192.168.32.101	192.168.32.100	TCP	60	49215 → http(80) [ACK] Seq=1 Ack=1 Win=0 Len=0
4	0.000312022	192.168.32.101	192.168.32.100	HTTP	204	GET / HTTP/1.1
5	0.000329667	192.168.32.100	192.168.32.101	TCP	54	http(80) → 49215 [ACK] Seq=1 Ack=308 Win=0 Len=0
6	0.020210558	192.168.32.100	192.168.32.101	TCP	204	http(80) → 49215 [PSH, ACK] Seq=1 Ack=308 Win=0 Len=0 [TCP segment of a reassembled PDU]
7	0.022918205	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
8	0.023216076	192.168.32.101	192.168.32.100	TCP	60	49215 → http(80) [ACK] Seq=308 Ack=410 Win=0 Len=0
9	0.023392319	192.168.32.101	192.168.32.100	TCP	60	49215 → http(80) [FIN, ACK] Seq=308 Ack=410 Win=0 Len=0
10	0.023411415	192.168.32.100	192.168.32.101	TCP	54	http(80) → 49215 [ACK] Seq=410 Ack=309 Win=0 Len=0
44	39.945506357	192.168.32.101	192.168.32.100	TCP	60	49217 → http(80) [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM WS=128
45	39.945541604	192.168.32.100	192.168.32.101	TCP	60	http(80) → 49217 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 SACK_PERM WS=128
46	39.945713467	192.168.32.101	192.168.32.100	HTTP	271	GET /msdownload/update/v3/static/trustedroot/authtl.cab HTTP/1.1
47	39.945877662	192.168.32.101	192.168.32.100	HTTP	204	http(80) → 49217 [PSH, ACK] Seq=1 Ack=218 Win=0 Len=0
48	39.945891331	192.168.32.100	192.168.32.101	TCP	54	http(80) → 49217 [ACK] Seq=1 Ack=218 Win=0 Len=0
49	39.969541900	192.168.32.100	192.168.32.101	TCP	204	http(80) → 49217 [PSH, ACK] Seq=1 Ack=218 Win=0 Len=0
50	39.973244223	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
51	39.973560859	192.168.32.101	192.168.32.100	TCP	60	49217 → http(80) [ACK] Seq=218 Ack=410 Win=0 Len=0
52	39.974997879	192.168.32.101	192.168.32.100	TCP	60	49217 → http(80) [FIN, ACK] Seq=218 Ack=410 Win=0 Len=0

Questi sono tutti i processi del protocollo http.

Per vedere il protocollo **HTTPS** dovremmo prima andare nel browser web e cambiare da http a https sulla barra di ricerca.

Mentre per vedere i pacchetti del controllo **HTTPS** andremmo a scrivere il codice:

```
ip.addr == 192.168.32.101 && tcp.port == 443
```

Ci restituirà la seguente schermata:

No.	Time	Source	Destination	Protocol	Length	Info
13	28.503122959	192.168.32.101	192.168.32.100	TCP	60	49216 → https(443) [SYN] Seq=0 Win=0 Len=0 MSS=1460 WS=4 SACK_PERM
14	28.503172426	192.168.32.100	192.168.32.101	TCP	60	https(443) → 49216 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 SACK_PERM WS=128
15	28.503526313	192.168.32.101	192.168.32.100	TCP	60	49216 → https(443) [ACK] Seq=1 Ack=1 Win=0 Len=0
16	28.503756190	192.168.32.101	192.168.32.100	TLSv1	215	Client Hello
17	28.503807540	192.168.32.100	192.168.32.101	TCP	54	https(443) → 49216 [ACK] Seq=1 Ack=162 Win=0 Len=0
18	28.503421335	192.168.32.100	192.168.32.101	TLSv1	1373	Server Hello, Certificate, Server Key Exchange, Server Hello Done
19	28.502218306	192.168.32.101	192.168.32.100	TLSv1	168	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
20	28.502270889	192.168.32.100	192.168.32.101	TCP	54	https(443) → 49216 [ACK] Seq=1320 Ack=296 Win=0 Len=0
21	28.503543189	192.168.32.100	192.168.32.101	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
22	28.77210321	192.168.32.100	192.168.32.101	TCP	113	TCP Retransmission https(443) → 49216 [PSH, ACK] Seq=1320 Ack=296 Win=0 Len=0
24	28.772506209	192.168.32.101	192.168.32.100	TCP	60	49216 → https(443) [ACK] Seq=296 Ack=1379 Win=0 Len=0 SLE=1320 SRE=1379
114	62.837455428	192.168.32.101	192.168.32.100	TCP	60	49216 → https(443) [FIN, ACK] Seq=296 Ack=1379 Win=0 Len=0
115	62.837663567	192.168.32.100	192.168.32.101	TLSv1	91	Encrypted Alert
116	62.837850450	192.168.32.101	192.168.32.100	TCP	60	49216 → https(443) [RST, ACK] Seq=297 Ack=1416 Win=0 Len=0

La differenza tra i due protocolli è la **cifratura** delle informazioni del pacchetto:

HTTP:

```
..TQ..6..E
*d @. @. . . . d .
e.P.? $B VO.Q..P.
[.]6..<html>. <
head>. <title
>INetSim default
HTML page</titl
e>. </head>. <
body>. <p></p>
>. <p align="
center"> This is
the defa ult HTML
page fo r INetSi
m HTTP s erver fa
ke mode. </p>.
<p alig n="cente
r">This file is
an HTML document
.</p>. </body>.
</html>.
```

HTTPS:

```

. . . TQ . . . 6 . . E .
. . O D @ . . UK . . d .
. . e . . @ [ N X M v . P .
. . [ . . . Y . . U .
. . - . . . ( h . . F . .
. . # . . . . 2DOWNGRD
. . . . . W . . H . . A
. . ( . * . (YE . . .
rI . . . . .
. . . . . k . . g . d
. . a0 . . ]0 . . E . .
. . . 1 / . . ^ . 5R . .
. . . ME0 . . . * . H . .
. . . . 0 > 1 . 0 . . U .
. . INetSi m1 0 . . U
. . . Deve lopment1
. . 0 . . U . . inetsi
m.org0 . . 2303101
35115Z . . 33030713
5115Z0 > 1 . 0 . . U .
. . INetSi m1 0 . . U
. . . Deve lopment1

```

Inoltre possiamo anche vedere l'indirizzamento dei pacchetti tramite l'indirizzo **MAC** che cambia ad ogni nodo e la sequenza **seq - ack** per l'operazione di handshake:

```

▼ Ethernet II, Src: PcsCompu_e9:54:51 (08:00:27:e9:54:51), Dst: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36)
  ► Destination: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36)
  ► Source: PcsCompu_e9:54:51 (08:00:27:e9:54:51)

```

```

▼ Ethernet II, Src: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36), Dst: PcsCompu_e9:54:51 (08:00:27:e9:54:51)
  ► Destination: PcsCompu_e9:54:51 (08:00:27:e9:54:51)
  ► Source: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36)
  Type: IPv4 (0x0800)

```

```

66 49215 → http(80) [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
66 http(80) → 49215 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
60 49215 → http(80) [ACK] Seq=1 Ack=1 Win=65700 Len=0
361 GET / HTTP/1.1
54 http(80) → 49215 [ACK] Seq=1 Ack=308 Win=64128 Len=0
204 http(80) → 49215 [PSH, ACK] Seq=1 Ack=308 Win=64128 Len=150 [TCP segment of a reassembled PDU]
312 HTTP/1.1 200 OK (text/html)
60 49215 → http(80) [ACK] Seq=308 Ack=410 Win=65292 Len=0
60 49215 → http(80) [FIN, ACK] Seq=308 Ack=410 Win=65292 Len=0
54 http(80) → 49215 [ACK] Seq=410 Ack=309 Win=64128 Len=0

```