

Simulazione rete complessa UNIT 1 WEEK 1

Assegnazione indirizzo IPv4 VM Kali Linux

Da terminale:

```
(kali㉿kali)-[~]  
$ sudo nano /etc/network/interfaces
```

- Elevo l'user account a privilegi root ed eseguo l'editor di testo nano

```
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
```

- Imposto l'indirizzo IPv4 statico, salvo e chiudo.

```
(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:fec7:e136 prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:c7:e1:36 txqueuelen 1000 (Ethernet)  
    RX packets 3840 bytes 419056 (409.2 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 2504 bytes 437453 (427.2 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- Riavvio la VM e controllo che la configurazione dell'interfaccia di rete corrisponda

Assegnazione indirizzo IPv4 VM Windows 7

Dal pannello di controllo seleziono:

- "Network and Internet"
- "Network and Sharing Center"
- "Change adapter settings"
- "Properties" della LAN
- "Internet Protocol Version 4 (TCP/IPv4)"
- "Properties" del protocollo selezionato

Use the following IP address:	
IP address:	192 . 168 . 32 . 101
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	192 . 168 . 32 . 1

- Imposto l'indirizzo IPv4 statico

```
C:\Users\admin>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::f470:72bf:bf3c:fbdc%11
    IPv4 Address. . . . . : 192.168.32.101
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.32.1
```

- Riavvio la VM e controllo che la configurazione dell'interfaccia di rete corrisponda nella command-line

```
(kali㉿kali)-[~]
$ ping 192.168.32.101
PING 192.168.32.101 (192.168.32.101) 56(84) bytes of data:
64 bytes from 192.168.32.101: icmp_seq=1 ttl=128 time=1.11 ms
64 bytes from 192.168.32.101: icmp_seq=2 ttl=128 time=0.666 ms
64 bytes from 192.168.32.101: icmp_seq=3 ttl=128 time=0.505 ms
64 bytes from 192.168.32.101: icmp_seq=4 ttl=128 time=0.696 ms
^C
— 192.168.32.101 ping statistics —
4 packets transmitted, 4 received, 0% packet loss, time 3424ms
rtt min/avg/max/mdev = 0.505/0.744/1.110/0.223 ms
```

```
Pinging 192.168.32.100 with 32 bytes of data:
Reply from 192.168.32.100: bytes=32 time<1ms TTL=64
Reply from 192.168.32.100: bytes=32 time<1ms TTL=64
Reply from 192.168.32.100: bytes=32 time<1ms TTL=64
Reply from 192.168.32.100: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.32.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

- Mi assicuro che le VM siano entrambe raggiungibili

Configurazione inetsim

Da terminale Kali:

```
(kali㉿kali)-[~]
$ sudo nano /etc/inetsim/inetsim.conf
```

- Elevo l'user account a privilegi root ed eseguo l'editor di testo nano

```
service_bind_address 192.168.32.100
```

- Imposto l'indirizzo IP che riceverà e distribuirà il traffico network

```
dns_default_ip 192.168.32.100
```

- Imposto l'indirizzo IP che riceverà le risposte DNS facendolo corrispondere all'indirizzo IPv4 della VM Kali Linux

```
dns_static_epicode.internal 192.168.32.100
```

- Imposto un host file remoti che corrisponda a epicode.internal
- Salvo e chiudo

```

(kali㉿kali)-[~]
└─$ sudo inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenber
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 254862) ===
Session ID:      254862
Listening on:    192.168.32.100
Real Date/Time:  2023-05-06 20:31:51
Fake Date/Time: 2023-05-06 20:31:51 (Delta: 0 seconds)
Forking services ...
* dns_53_tcp_udp - started (PID 254864)
* http_80_tcp - started (PID 254865)
* time_37_tcp - started (PID 254879)
* pop3s_995_tcp - started (PID 254870)
* discard_9_tcp - started (PID 254885)
* syslog_514_udp - started (PID 254878)
* smtp_25_tcp - started (PID 254867)
* daytime_13_tcp - started (PID 254881)
* irc_6667_tcp - started (PID 254874)
* chargen_19_tcp - started (PID 254889)
* ntp_123_udp - started (PID 254875)
* time_37_udp - started (PID 254880)
* https_443_tcp - started (PID 254866)
* ftp_21_tcp - started (PID 254871)
* smtps_465_tcp - started (PID 254868)
* discard_9_udp - started (PID 254886)
* daytime_13_udp - started (PID 254882)
* quotd_17_udp - started (PID 254888)
* chargen_19_udp - started (PID 254890)
* echo_7_tcp - started (PID 254883)
* dummy_1_tcp - started (PID 254891)
* quotd_17_tcp - started (PID 254887)
* echo_7_udp - started (PID 254884)
* dummy_1_udp - started (PID 254892)
* tftp_69_udp - started (PID 254873)
* ftps_990_tcp - started (PID 254872)
* pop3_110_tcp - started (PID 254869)
* ident_113_tcp - started (PID 254877)
* finger_79_tcp - started (PID 254876)
done.
Simulation running.

```

- Faccio partire la simulazione

Dalla VM di Windows e dal pannello di controllo seleziono:

- "Network and Internet"
- "Network and Sharing Center"
- "Change adapter settings"
- "Properties" della LAN
- "Internet Protocol Version 4 (TCP/IPv4)"
- "Properties" del protocollo selezionato

- Imposto l'indirizzo del server DNS

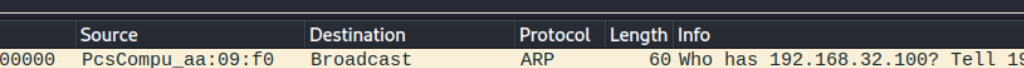
-
- The screenshot shows a web browser window. The title bar says "SIM INetSim default HTML page". The address bar shows "Non sicurc | epicode.internal/". The page content displays the text: "This is the default HTML page for INetSim HTTP server fake mode." and "This file is an HTML document."

Intercettazione comunicazione con Wireshark

- Catturo i pacchetti sull'https con Wireshark
- Imposto un filtro per il protocollo ARP(**arp**)

- | No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-------------------|-------------------|----------|--------|---|
| 1 | 0.000000000 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.100? Tell 192.168.32.101 |
| 2 | 0.000014587 | PcsCompu_c7:e1:36 | PcsCompu_aa:09:f0 | ARP | 42 | 192.168.32.100 is at 08:00:27:c7:e1:36 |
| 13 | 0.002257704 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
| 60 | 0.584697044 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
| 61 | 1.558526381 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
- ▶ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
 ▶ Ethernet II, Src: PcsCompu_aa:09:f0 (08:00:27:aa:09:f0), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
 ▶ Source: PcsCompu_aa:09:f0 (08:00:27:aa:09:f0)
 Type: ARP (0x0806)
 Padding: 00
 ▶ Address Resolution Protocol (request)

- Nel primo frame la richiesta ARP chiede a chi sia assegnato l'IP 192.168.32.100 e di comunicarlo a 192.168.32.101
- Analizzando il secondo livello la richiesta viene eseguita sfruttando il MAC Address impostandolo al suo valore più alto (ff:ff:ff:ff:ff:ff)
- L'origine della richiesta ARP è **PcsCompu_aa:09:f0**(08:00:27:aa:09:f0) mentre la destinazione è **Broadcast**(ff:ff:ff:ff:ff:ff)

- 
- The image shows a Wireshark packet capture of an ARP request. The top section displays a list of packets, with packet 2 selected. The packet details pane shows the Ethernet II header, the ARP request payload, and the Address Resolution Protocol (ARP) details. The ARP request is for the IP address 192.168.32.100, and the source MAC address is 08:00:27:c7:e1:36. The ARP details section shows the source MAC address and the type of request (ARP).
- | No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-------------------|-------------------|----------|--------|---|
| 1 | 0.000000000 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.100? Tell 192.168.32.101 |
| 2 | 0.000014587 | PcsCompu_c7:e1:36 | PcsCompu_aa:09:f0 | ARP | 42 | 192.168.32.100 is at 08:00:27:c7:e1:36 |
| 13 | 0.002257704 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
| 60 | 0.584697044 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
| 61 | 1.558526381 | PcsCompu_aa:09:f0 | Broadcast | ARP | 60 | Who has 192.168.32.1? Tell 192.168.32.101 |
- Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
- Ethernet II, Src: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36), Dst: PcsCompu_aa:09:f0 (08:00:27:aa:09:f0)
- Destination: PcsCompu_aa:09:f0 (08:00:27:aa:09:f0)
- Source: PcsCompu_c7:e1:36 (08:00:27:c7:e1:36)
- Type: ARP (0x0806)
- Address Resolution Protocol (reply)

- Nel secondo frame il protocollo ARP risponde alla richiesta precedente, comunicando che l'IP 192.168.32.100 è stato assegnato al MAC address 08:00:27:c7:e1:36
- Questa volta l'origine è **PcsCompu_c7:e1:36**(08:00:27:c7:e1:36) assegnato all'IP 192.168.32.100, mentre la destinazione è **PcsCompu_aa:09:f0**(08:00:27:aa:09:f0), rispondendo così alla richiesta ARP precedente.

Analisi HTTPS

- Faccio un clear del filtro precedente e ne metto uno per visualizzare il traffico HTTPS (**tcp.port==443**)

tcp.port==443						
No.	Time	Source	Destination	Protocol	Length	Info
3	0.001358270	192.168.32.101	192.168.32.100	TCP	66	55221 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
4	0.001376520	192.168.32.100	192.168.32.101	TCP	66	443 → 55221 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5	0.001942248	192.168.32.101	192.168.32.100	TCP	60	55221 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0

I frame mostrano l'apertura di una connessione tramite "**Three-way handshake**" TCP, un metodo utilizzato in una rete TCP/IP per creare una connessione tra un host/client locale e un server

- Nel terzo frame il pacchetto **SYN**(Seq=0) viene inviato dall'IP 192.168.32.101 (port 55221) al server di destinazione 192.168.32.100 (port 443) per chiedere al server se sia disponibile per nuove connessioni
- Nel quarto frame il server 192.168.32.100 riceve il pacchetto SYN dal nodo client, risponde e restituisce la ricevuta di conferma con **SYN,ACK**(Seq=0,Ack=1)
- Nel quinto frame il client 192.168.32.101 riceve il pacchetto SYN,ACK dal server 192.168.32.100 e risponde con un pacchetto **ACK**(Seq=1, Ack=1)

In questo modo viene creata la **connessione TCP**, l'host ed il server sono in grado di comunicare

tcp.port==443						
No.	Time	Source	Destination	Protocol	Length	Info
3	0.001358270	192.168.32.101	192.168.32.100	TCP	66	55221 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
4	0.001376520	192.168.32.100	192.168.32.101	TCP	66	443 → 55221 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5	0.001942248	192.168.32.101	192.168.32.100	TCP	60	55221 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
6	0.002723335	192.168.32.101	192.168.32.100	TLSv1.3	571	Client Hello
7	0.002732960	192.168.32.100	192.168.32.101	TCP	54	443 → 55221 [ACK] Seq=1 Ack=518 Win=64128 Len=0
9	0.027985261	192.168.32.100	192.168.32.101	TLSv1.3	1475	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
10	0.030195220	192.168.32.101	192.168.32.100	TLSv1.3	84	Change Cipher Spec, Application Data
11	0.030346889	192.168.32.101	192.168.32.100	TCP	60	55221 → 443 [FIN, ACK] Seq=548 Ack=1422 Win=64256 Len=0
12	0.035104160	192.168.32.100	192.168.32.101	TCP	54	443 → 55221 [FIN, ACK] Seq=1422 Ack=549 Win=64128 Len=0
13	0.035335531	192.168.32.101	192.168.32.100	TCP	60	55221 → 443 [ACK] Seq=549 Ack=1423 Win=64256 Len=0

Stabilita la connessione TCP possiamo notare come si consegua un handshake **TLSv1.3**

- Nell'handshake TLS 1.3 abbiamo un "**Client Hello**" dove il client 192.168.32.191 invia un messaggio "hello" al server
- Questo messaggio include la **versione TLS** supportata dal client, le suite di crittografia e una stringa di byte casuali generata dal client
- Abbiamo poi il messaggio "**Server Hello**" in risposta al client, dove il server invia un messaggio contenente il **certificato SSL** del server, la suite di cifratura scelta dal server e la stringa di byte casuali dal server
- Il client verifica il certificato SSL del server con l'autorità di certificazione che l'ha emesso, confermando che il server è chi dice di essere ed il client stia interagendo con il proprietario del dominio
- Successivamente il client invia un'altra stringa casuale di byte, il "**segreto premaster**", crittografato con chiave pubblica e che può essere decrittografato solo con la chiave privata del server
- Il server decrittifica il segreto premaster e, sia il client che il server generano chiavi di sessione dalla stringa casuale client, dalla stringa casuale server e dal segreto premaster
- Sia il client che il server inviano un messaggio "**terminato**" crittografato con una chiave di sessione
- In questo modo l'handshake è stato completato e la comunicazione continua utilizzando le chiavi di sessione

Analisi HTTP

- Catturo i pacchetti sull'http con Wireshark
- Imposto un filtro per visualizzare il traffico HTTP(**tc.port==80**)

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000357508	192.168.32.101	192.168.32.100	TCP	66	55274 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
4	0.000374755	192.168.32.100	192.168.32.101	TCP	66	80 → 55274 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5	0.001271963	192.168.32.101	192.168.32.100	TCP	60	55274 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0

I frame mostrano l'apertura di una connessione tramite "**Three-way handshake**" TCP, un metodo utilizzato in una rete TCP/IP per creare una connessione tra un host/client locale e un server

- Nel terzo frame il pacchetto **SYN**(Seq=0) viene inviato dall'IP 192.168.32.101 (port 55274) al server di destinazione 192.168.32.100 (port 80) per chiedere al server se sia disponibile per nuove connessioni
- Nel quarto frame il server 192.168.32.100 riceve il pacchetto SYN dal nodo client, risponde e restituisce la ricevuta di conferma con **SYN,ACK**(Seq=0,Ack=1)
- Nel quinto frame il client 192.168.32.101 riceve il pacchetto SYN,ACK dal server 192.168.32.100 e risponde con un pacchetto **ACK**(Seq=1, Ack=1)
- In questo modo viene creata la **connessione TCP**, l'host ed il server sono in grado di comunicare

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000357508	192.168.32.101	192.168.32.100	TCP	66	55274 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
4	0.000374755	192.168.32.100	192.168.32.101	TCP	66	80 → 55274 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
5	0.001271963	192.168.32.101	192.168.32.100	TCP	60	55274 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
6	0.001882195	192.168.32.101	192.168.32.100	HTTP	530	GET / HTTP/1.1
7	0.001891285	192.168.32.100	192.168.32.101	TCP	54	80 → 55274 [ACK] Seq=1 Ack=477 Win=64128 Len=0
9	0.011169970	192.168.32.100	192.168.32.101	TCP	204	80 → 55274 [PSH, ACK] Seq=1 Ack=477 Win=64128 Len=150 [TCP segment of a reassembled PDU]
10	0.012276384	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
11	0.012559147	192.168.32.101	192.168.32.100	TCP	60	55274 → 80 [ACK] Seq=477 Ack=410 Win=65280 Len=0
12	0.013325437	192.168.32.101	192.168.32.100	TCP	60	55274 → 80 [FIN, ACK] Seq=477 Ack=410 Win=65280 Len=0
13	0.013335750	192.168.32.100	192.168.32.101	TCP	54	80 → 55274 [ACK] Seq=410 Ack=478 Win=64128 Len=0

La comunicazione tra server e client **HTTP** è basata sullo scambio di messaggi testuali tipicamente indicati come **HTTP Request** e **HTTP Response**.

All'interno di questi due tipi di comandi abbiamo gli **HTTP headers** che sono delle informazioni aggiuntive che accompagnano sia le richieste HTTP che le risposte HTTP

- Il sesto frame contiene l'**HTTP Request GET** e la versione HTTP utilizzata
- Il decimo frame contiene l'**HTTP Response Header 200 OK**, la richiesta è stata ricevuta, elaborata e accettata.
- Dopo ciò ne conseguono i dati relativi alla pagina richiesta

Conclusioni

L'**HTTPS** è **HTTP** con crittografia e verifica.

La differenza tra i due protocolli è che **HTTPS** utilizza **TSL (SSL)** per crittografare le normali richieste e risposte **HTTP** e per firmare digitalmente tali richieste e risposte.

Di conseguenza, **HTTPS** è molto più sicuro di **HTTP**.