

Progetto finale Modulo 1 (W4D4) - Alessandro Criscuoli

In questa esercitazione, eseguita su laboratorio virtuale VirtualBox, su Internal Network ho impostato gli IP delle macchine Kali Linux e Windows nei seguenti modi:

```
(kali@kali)~$ cat /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
    netmask 255.255.255.0
    gateway 192.168.32.1
    dns-nameservers 192.168.32.100

(kali@kali)~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
    link/ether 08:00:27:1f:b7:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.32.100/24 brd 192.168.32.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe1f:b723/64 scope link proto kernel_ll
        valid_lft forever preferred_lft forever
```

```
Scheda Ethernet Ethernet:

Suffisso DNS specifico per connessione:
Descrizione . . . . . : Intel(R) PRO/1000 MT Desktop Adapter
Indirizzo fisico. . . . . : 08-00-27-1A-45-C5
DHCP abilitato. . . . . : No
Configurazione automatica abilitata : Si
Indirizzo IPv4. . . . . : 192.168.32.101(Preferenziale)
Subnet mask . . . . . : 255.255.255.0
Gateway predefinito . . . . . : 192.168.32.1
Server DNS . . . . . : 192.168.32.100
NetBIOS su TCP/IP . . . . . : Attivato
```

Come si può notare dalle immagini ho settato il DNS server con IP della KaliLinux su entrambe le vm. Per verificare che le vm comunicassero correttamente ho fatto un test di ping reciproco, inizialmente ciò avveniva con esito positivo solo per il ping da Windows a Kali ma

non da Kali a Windows, questo dato dalla protezione del Windows Firewall, una volta inserita la policy con action Allow, il ping da Kali a Windows è stato possibile da testare con esito positivo.

Regole connessioni in entrata

Nome	Profilo	Abilitata	Indirizzo locale	Operazione	Protocollo
Consenti ICMP da KaliLinux	Tutti	Sì	Qualsiasi	Consenti	ICMPv4

Una volta configurata la rete locale l'obiettivo era attivare i servizi DNS, HTTP e HTTPS in modo da renderli disponibili all'interno della rete, per poter simulare una architettura client-server.

In questo report si descrivono i passaggi per configurare come server la vm Kali e come client la vm Windows, come obiettivo finale il raggiungimento della risorsa tramite hostname

epicode.internal

Ho optato per l'utilizzo di **Inetsim** per la simulazione dei servizi, iniziando dal DNS questo non poteva essere simulato per dei problemi di stabilità della versione di Net::DNS, un modulo in utilizzo da Inetsim, ho dovuto così eseguire un upgrade del modulo alla versione 1.37, recuperato dalla seguente repository: <https://cpan.metacpan.org/authors/id/N/NL/NLNETLABS/Net-DNS-1.37.tar.gz>

comandi utilizzati (da terminale)

wget *link*

tar -xzf Net-DNS-1.37.tar.gz #comando di estrazione, xzf sono le opzioni di estrazione

cd *path cartella estratta NetDNS*

perl Makefile.PL #controlla le dipendenze e genera il "Makefile" (file per l'installazione del modulo)

make #compilazione del modulo tramite il Makefile

make test #test del modulo

sudo make install #installa il modulo con permessi admin

Nota: Makefile è un file di testo che contiene istruzioni per automatizzare la compilazione e l'installazione di software.

Fatti i passaggi per il FIX dei problemi di DNS service ho continuato con la configurazione di Inetsim.

Configurazione Inetsim

entrando in modifica nel file di configurazione di Inetsim tramite comando:

`sudo nano /etc/inetsim/inetsim.conf`

```
GNU nano 8.6
#####
# InetSim configuration file
#
#####
# Main configuration
#####
#####
# start_service
#
# The services to start
#
# Syntax: start_service <service name>
#
# Default: none
#
# 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
#start_service time_udp
#start_service daytime_tcp
#start_service daytime_udp
#start_service echo_tcp
#start_service echo_udp
#####
^G Help      ^O Write Out  ^F Where Is   ^X Cut
^X Exit      ^R Read File  ^_ Replace    ^U Paste
```

Qui ho aggiunto il carattere # per commentare lo start_service dei servizi non essenziali per la simulazione, tutti eccetto dns, http e https. Successivamente ho specificato l'IP address della macchina su cui verranno esposti i servizi simulati, ovvero la KaliLinux [192.168.32.100].

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

Service DNS

```
#####
# dns_static
#
# 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
```

Per aggiungere il record DNS “epicode.internal” ho aggiunto sotto dns_static una riga in più, non commentata, dove specifico hostname seguito dall'IP address della vm KaliLinux.

I servizi HTTP/HTTPS non necessitano di nessuna configurazione, sono già impostati di default sulle rispettive porte 80 e 443, mentre il loro contenuto non è importante ai fini della simulazione.

Start dei servizi

Per effettuare lo start dei servizi ho digitato da riga di comando “`sudo inetsim`” in modo da eseguire l'applicativo con permessi elevati.

```
(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 47828) ==
Session ID:      47828
Listening on:    192.168.32.100
Real Date/Time:  2025-10-19 22:31:14
Fake Date/Time: 2025-10-19 22:31:14 (Delta: 0 seconds)
Forking services ...
* dns_53_tcp_udp - started (PID 47838)
* http_80_tcp    - started (PID 47839)
* https_443_tcp  - started (PID 47840)
done.
Simulation running.
█
```

Avendo effettuato tutto correttamente si possono vedere i PID (Process ID) del processo principale Inetsim e dei servizi a cui abbiamo lasciato lo `start_service` in file di configurazione, assieme allo stato “simulation running”.

Verifica da client

Da vm Windows sono andato ad effettuare le verifiche dei servizi:

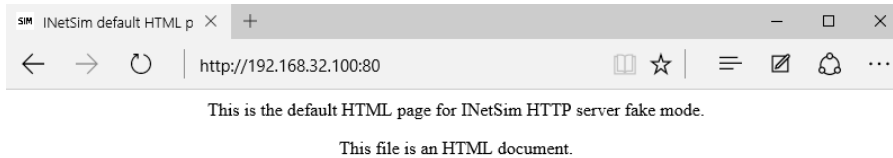
```
C:\Users\user>ping epicode.internal

Esecuzione di Ping epicode.internal [192.168.32.100] con 32 byte di dati:
Risposta da 192.168.32.100: byte=32 durata<1ms TTL=64
Risposta da 192.168.32.100: byte=32 durata<1ms TTL=64
Risposta da 192.168.32.100: byte=32 durata=1ms TTL=64
Risposta da 192.168.32.100: byte=32 durata<1ms TTL=64

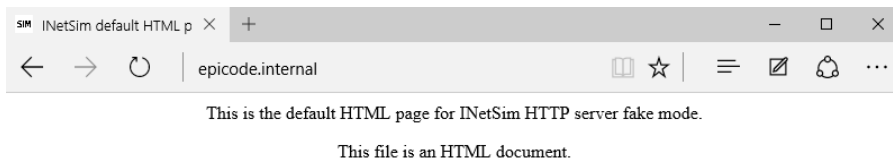
Statistiche Ping per 192.168.32.100:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 0ms, Massimo = 1ms, Medio = 0ms
```

Pingando l'hostname *epicode.internal* verrà tradotto l'IP 192.168.32.100 [Kali]
Questo dimostra che la simulazione del server DNS sta funzionando sulla internal network.

Visita da Web Browser (http/https)



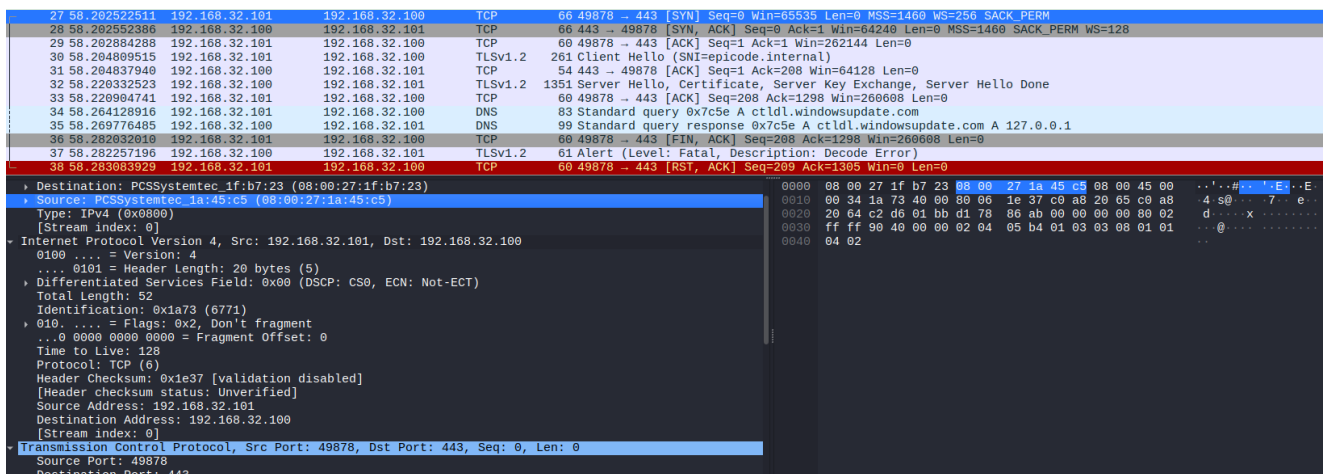
Immettendo l'indirizzo IP della KaliLinux sono stato indirizzato alla pagina http esposta sulla porta 80.



Immettendo l'hostname epicode.internal sono stato indirizzato alla pagina https esposta sulla porta 443.

Sniff pacchetti con Wireshark (HTTPS)

A questo punto per analizzare i pacchetti e vedere le request e response ho iniziato lo "sniffaggio" dell'interfaccia eth0 su Kali:

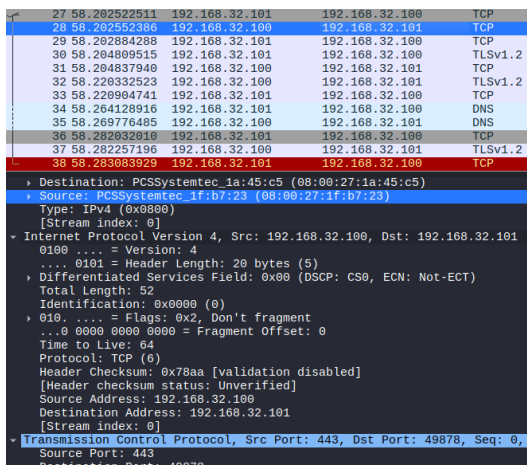


Le fasi catturate sono del momento in cui ho richiesto dal browser client di visitare l'indirizzo <https://epicode.internal:443>, si possono verificare i MAC address di source e destination, le porte di entrambi e le informazioni di ogni pacchetto. Nella prima request c'è:

Source MAC address [08-00-27-1A-45-C5] -> Windows / Source port -> 49878

Destination MAC address [08:00:27:1f:b7:23] -> Kali / Destination port -> 443

Subito dopo abbiamo la prima response con Source e Destination invertiti.



Nota: In questi screenshot si può notare un Error riguardante il protocollo TLS, dovuto al fatto che la pagina HTTPS non riesce a restituire nessun certificato SSL/TLS.

Sniff pacchetti con Wireshark (HTTP)

Ho ripetuto il procedimento mentre visitavo la pagina in http, questo è ciò che ne emerge:

4	12.929954912	192.168.32.101	192.168.32.100	TCP	66 49905 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
5	12.929997611	192.168.32.100	192.168.32.101	TCP	66 80 → 49905 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
6	12.930324996	192.168.32.101	192.168.32.100	TCP	60 49905 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
7	12.930846482	192.168.32.101	192.168.32.100	HTTP	405 GET / HTTP/1.1
8	12.930861155	192.168.32.100	192.168.32.101	TCP	54 80 → 49905 [ACK] Seq=1 Ack=352 Win=64128 Len=0
9	12.945777487	192.168.32.100	192.168.32.101	TCP	204 80 → 49905 [PSH, ACK] Seq=1 Ack=352 Win=64128 Len=150 [TCP PDU reassembled in 11]
10	12.946806698	192.168.32.101	192.168.32.100	TCP	60 49905 → 80 [ACK] Seq=352 Ack=151 Win=261888 Len=0
11	12.946829346	192.168.32.100	192.168.32.101	HTTP	312 HTTP/1.1 200 OK (text/html)
12	12.947511692	192.168.32.101	192.168.32.100	TCP	60 49905 → 80 [ACK] Seq=352 Ack=409 Win=261632 Len=0
13	12.948957172	192.168.32.100	192.168.32.101	TCP	54 80 → 49905 [FIN, ACK] Seq=409 Ack=352 Win=64128 Len=0
14	12.949425036	192.168.32.101	192.168.32.100	TCP	60 49905 → 80 [ACK] Seq=352 Ack=410 Win=261632 Len=0
15	12.951151435	192.168.32.101	192.168.32.100	TCP	60 49905 → 80 [FIN, ACK] Seq=352 Ack=410 Win=261632 Len=0
16	12.951174853	192.168.32.100	192.168.32.101	TCP	54 80 → 49905 [ACK] Seq=410 Ack=353 Win=64128 Len=0
17	13.018972212	192.168.32.101	192.168.32.100	DNS	77 Standard query 0x10af A urs.microsoft.com
18	13.017204417	192.168.32.100	192.168.32.101	DNS	93 Standard query response 0x10af A urs.microsoft.com A 127.0.0.1
19	14.992907648	PCSSystemtec_1a:45:...	Broadcast	ARP	60 Who has 192.168.32.1? Tell 192.168.32.101
20	15.726757255	PCSSystemtec_1a:45:...	Broadcast	ARP	60 Who has 192.168.32.1? Tell 192.168.32.101
21	16.724798114	PCSSystemtec_1a:45:...	Broadcast	ARP	60 Who has 192.168.32.1? Tell 192.168.32.101

Destination: PCSSystemtec_1f:b7:23 (08:00:27:1f:b7:23)	0000 08 00 27 1f b7 23 08 00 27 1a 45 c5 08 00 45 00 ..'..#...E..E
Source: PCSSystemtec_1a:45:c5 (08:00:27:1a:45:c5)	0010 00 34 1a a2 40 00 00 06 1e 08 c0 a8 20 65 c0 a8 4..@.....e..
Type: IPv4 (0x0800)	0020 20 64 c2 f1 00 50 46 35 cb 4e 00 00 00 00 02 d..PF5.N.....
[Stream index: 1]	0030 ff ff d8 30 00 00 02 04 05 b4 01 03 03 08 01 01 ..0.....
Internet Protocol Version 4, Src: 192.168.32.101, Dst: 192.168.32.100	0040 64 02
0100 = Version: 4	
... 0101 = Header Length: 20 bytes (5)	
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)	
Total Length: 52	
Identification: 0x1aa2 (6818)	
010. = Flags: 0x2, Don't Fragment	
...0 0000 0000 0000 = Fragment Offset: 0	
Time to Live: 128	
Protocol: TCP (6)	
Header Checksum: 0x1e08 [validation disabled]	
[Header checksum status: Unverified]	
Source Address: 192.168.32.101	
Destination Address: 192.168.32.100	
[Stream index: 0]	
Transmission Control Protocol, Src Port: 49905, Dst Port: 80, Seq: 0, Len: 0	
Source Port: 49905	
Destination Port: 80	

Le fasi catturate sono del momento in cui ho richiesto dal browser client di visitare l'indirizzo <http://192.168.32.100:80>, oltre a notare gli stessi MAC address in Source e Destination si può notare un cambio delle porte, ora la porta in uso dal server è la 80, la default del servizio http. Oltre a questo mentre prima (in https) non si riusciva a vedere i contenuti delle request/response in chiaro ora questo è possibile.

Conclusione

Tramite l'utilizzo di Inetsim sono riuscito a simulare una architettura client-server in cui ho esposto ed utilizzato i servizi DNS, HTTP e HTTPS. Con Wireshark invece ho potuto analizzare lo scambio di pacchetti che avviene tra client e server durante la simulazione dei servizi.

Ho potuto constatare personalmente che la difficoltà maggiore della simulazione consisteva nel trovare una valida alternativa per la simulazione del DNS dato che quella proposta da Inetsim ha problemi di stabilità, dopo varie ore di troubleshooting ho trovato la soluzione aggiornando il Modulo Perl utilizzato da Inetsim per il DNS.