

PROGETTO FINALE M1

Il progetto finale del primo modulo ha avuto come obiettivo quello di consolidare le conoscenze acquisite durante questo primo mese.

L'obiettivo è stato simulare in un ambiente virtuale un'architettura client-server che permettesse al client (in questo caso la macchina Windows) di richiedere una risorsa web all'hostname `epicode.internal` che rispondesse all'indirizzo del server (in questo caso della macchina Kali).

Infine intercettare la comunicazione con Wireshark, evidenziando i MAC address di sorgente e destinazione ed il contenuto delle richieste HTTPS/HTTP ed evidenziare e motivare le differenze.

Il primo step che ho effettuato è stato quello di configurare l'indirizzo DNS sulla macchina windows, che sarebbe servito per sapere a che indirizzo IP avrebbe trovato il server che faceva l'associazione tra hostname e indirizzo IP.

Successivamente sono andato sulla macchina Kali e ho aperto la configurazione di Inetsim digitando sul terminale:

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

Dopo aver commentato con '#' i servizi che non mi servivano (lasciando per ora solo DNS e HTTPS), ho messo come `service_bind_address`: `192.168.50.100` (cioè l'indirizzo della macchina Kali), come `dns_default_ip` sempre lo stesso ip: `192.168.50.100`, come `default_hostname` `epicode` e come `default_domainname` `internal` (visto che dopo un paio di prove con `hostname epicode.internal` mi dava errore di sintassi).

Dopo aver lanciato il comando da root:

```
inetsim
```

Sono iniziati i problemi con il DNS.

Il servizio mi restituiva un errore che consisteva nel non trovare il 'main_loop' dentro al file `DNS.pm`:

```
[sudo] password for raul:
root@192:~# 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 5277) ===
Session ID: 5277
Listening on: 192.168.50.100
Real Date/Time: 2025-10-17 16:31:23
Fake Date/Time: 2025-10-17 16:31:23 (Delta: 0 seconds)
Forking services...
* dns_53_tcp_udp - started (PID 5279)
Can't locate object method "main_loop" via package "Net::DNS::Nameserver" at /usr/share/perl5/INetSim/DNS.pm line 69.
```

Raul Pastor

Dopo essermi addentrato dentro al file con il comando:

```
nano /usr/share/perl5/INetSim/DNS.pm
```

ho notato che effettivamente veniva richiamato una sola volta in tutto il file, come se fosse la rimanenza di un loop principale che doveva far girare il servizio DNS ma che ora non c'è più.

```
GNU nano 8.6 /usr/share/perl5/INetSim/DNS.pm *
my $uid = getpwnam($runasuser);
my $gid = getgrnam($runasgroup);
POSIX::setgid($gid);
my $newgid = POSIX::getgid();
if ($newgid != $gid) {
    INetSim::Log::MainLog("failed! (Cannot switch group)", INetSim::Config::getConfigParameter("DNS_ServiceName"));
    exit 0;
}

POSIX::setuid($uid);
if ($< != $uid || $> != $uid) {
    $< = $> = $uid; # try again - reportedly needed by some Perl 5.8.0 Linux systems
    if ($< != $uid) {
        INetSim::Log::MainLog("failed! (Cannot switch user)", INetSim::Config::getConfigParameter("DNS_ServiceName"));
        exit 0;
    }
}

$0 = 'inetsim_'; INetSim::Config::getConfigParameter("DNS_ServiceName");
INetSim::Log::MainLog("started (PID $CPID)", INetSim::Config::getConfigParameter("DNS_ServiceName"));
$server->run();
INetSim::Log::MainLog("stopped (PID $CPID)", INetSim::Config::getConfigParameter("DNS_ServiceName"));
exit 0;
```

(nell'immagine ho sostituito '\$server->main_loop;' con '\$server->run();' l'errore era scomparso ma il servizio cadeva subito nell''exit 0' c'era bisogno di un loop per tenere il servizio attivo).

dopo ore di ricerche sul web ho constatato che la soluzione migliore fosse quella di installare una versione obsoleta del modulo Perl che prevedesse il funzionamento del servizio e quindi la presenza del vecchio loop.

quindi dopo aver rimesso la rete in modalità bridged (per connettermi al router locale) su kali ho lanciato i comandi per disinstallare il modulo Perl tramite cpan per poi installarne una versione piu obsoleta:

```
sudo cpanm --uninstall Net::DNS
sudo cpanm Net::DNS@1.37
```

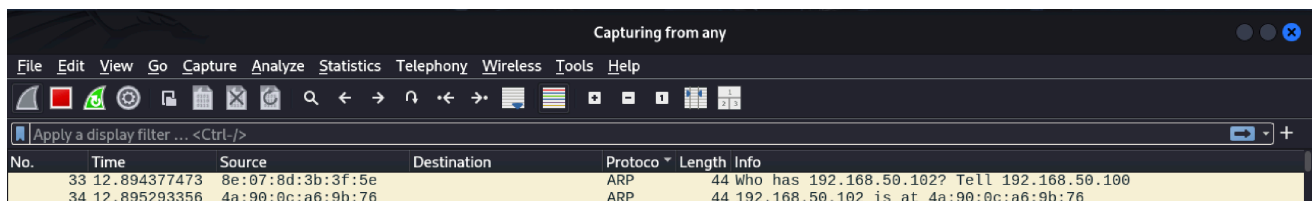
Infine dopo aver rilanciato Inetsim il problema era risolto (vedere immagine successiva).

```
raul@192: ~  
(raul@192)-[~]  
$ 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 5807) ===  
Session ID: 5807  
Listening on: 192.168.50.100  
Real Date/Time: 2025-10-19 09:46:40  
Fake Date/Time: 2025-10-19 09:46:40 (Delta: 0 seconds)  
Forking services...  
* dns_53_tcp_udp - started (PID 5809)  
* http_80_tcp - started (PID 5810)  
* https_443_tcp - started (PID 5811)  
done.  
Simulation running.  
^C * https_443_tcp - stopped (PID 5811)  
* http_80_tcp - stopped (PID 5810)  
* dns_53_tcp_udp - stopped (PID 5809)
```

Tornando alla consegna dell'esercizio, dopo aver impostato la rete delle macchine su internal, ho aperto Wireshark e ho simulato una richiesta web (HTTPS e HTTP) dalla macchina windows con hostname epicode.internal.

Le informazioni che possiamo ricavare da Wireshark sono tante, tra queste ci sono:

I Mac address dei due dispositivi attraverso il protocollo ARP:



The image shows a Wireshark packet capture window titled "Capturing from any". The packet list shows two ARP packets. The first packet is an ARP request from 8e:07:8d:3b:3f:5e to 4a:90:0c:a6:9b:76. The second packet is an ARP response from 4a:90:0c:a6:9b:76 to 8e:07:8d:3b:3f:5e.

No.	Time	Source	Destination	Protocol	Length	Info
33	12.894377473	8e:07:8d:3b:3f:5e	4a:90:0c:a6:9b:76	ARP	44	Who has 192.168.50.102? Tell 192.168.50.100
34	12.895293356	4a:90:0c:a6:9b:76	8e:07:8d:3b:3f:5e	ARP	44	192.168.50.102 is at 4a:90:0c:a6:9b:76

MAC address 4a:90:0c:a6:9b:76 -> IP 192.168.50.102

MAC address 8e:07:8d:3b:3f:5e -> IP 192.168.50.100

Oppure si può verificare sempre su Wireshark cliccando su un qualsiasi protocollo con un determinato IP sorgente per vedere le informazioni del Layer Data del modello ISO/OSI e visualizzare il campo 'Source' che rappresenta il Mac address sorgente. Il contenuto del protocollo HTTPS non contiene nessuna informazione sul contenuto a livello applicazione perché il payload è cifrato

```
▼ TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol  
  Opaque Type: Application Data (23)  
  Version: TLS 1.2 (0x0303)  
  Length: 19  
  Encrypted Application Data: 446c43552a3226f7e91546ca88273cae1ca2cb  
  [Application Data Protocol: Hypertext Transfer Protocol]
```

ma contiene principalmente informazioni sul collegamento a livello DATA (Mac address), a livello RETE (con indirizzi IP), a livello TRASPORTO (sempre su porta 443) e a livello

APPLICAZIONE (con sessione TLS sicura, cifratura pre-applicazione e HTTP cifrato all'interno della sessione TLS)

```
▶ Frame 1290: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on 0
▼ Linux cooked capture v1
  Packet type: Unicast to us (0)
  Link-layer address type: Ethernet (1)
  Link-layer address length: 6
  Source: 4a:90:0c:a6:9b:76 (4a:90:0c:a6:9b:76)
  Unused: 0000
  Protocol: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 70
  Identification: 0x9420 (37920)
  ▶ 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: TCP (6)
  Header Checksum: 0x8076 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.50.102
  Destination Address: 192.168.50.100
  [Stream index: 1]
▼ Transmission Control Protocol, Src Port: 51328, Dst Port: 443, Seq: 518
  Source Port: 51328
  Destination Port: 443
  [Stream index: 90]
  [Stream Packet Number: 8]
  ▶ [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 30]
  Sequence Number: 518 (relative sequence number)
  Sequence Number (raw): 572176636
  [Next Sequence Number: 548 (relative sequence number)]
  Acknowledgment Number: 1444 (relative ack number)
```

Il contenuto del protocollo HTTP contiene, a livello di applicazione, tutte le informazioni sulla richiesta del contenuto perché il payload non è cifrato (per esempio in questa richiesta GET):

```
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
  Host: epicode.internal\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9\r\n
  \r\n
  [Response in frame: 531]
  [Full request URI: http://epicode.internal/]
```

Per concludere le principali differenze tra protocollo HTTP e HTTPS per i layer ISO/OSI sono:

- Layer Fisico: Non ci sono differenze tra i due protocolli.
- Layer Data: I Mac address sorgente e destinazione sono sempre visibili.
- Layer Rete: Gli indirizzi IP sorgente e destinazione sono sempre visibili.
- Layer Trasporto: Per il protocollo HTTP viene usata la porta 80 mentre per l'HTTPS sempre la 443.
- Layer Sessione: Per il layer sessione il protocollo HTTPS utilizza la sessione TLS (prima chiamata SSL) per creare una sessione sicura, HTTP no.
- Layer Presentazione: Nel layer presentazione il protocollo HTTPS introduce la crittografia prima del layer applicazione, HTTP no.
- Layer Applicazione: a livello applicazione le richieste HTTP sono perfettamente leggibili in rete mentre le richieste HTTPS sono cifrate e protette da una sessione TLS sicura.

San Giovanni in Marignano, 19/10/2025

Raul Pastor