

# Progetto Finale M1 - Introduzione all'Hacking

## Brief

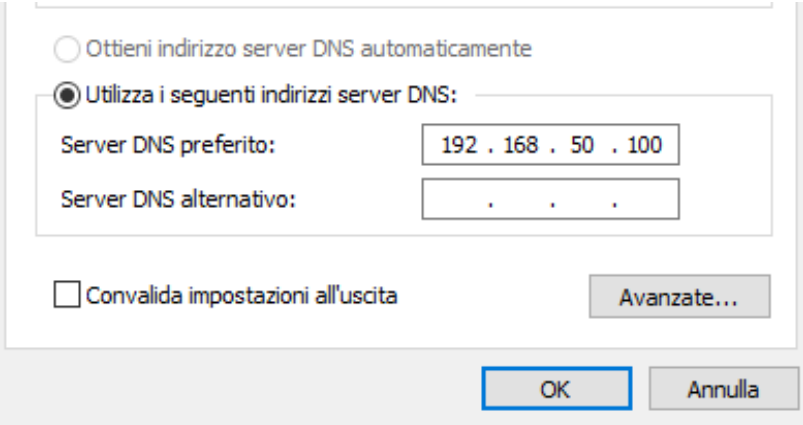
L'obiettivo dell'esercitazione è simulare, in un ambiente di laboratorio virtuale, un'architettura client-server nella quale un client effettua una richiesta web verso l'hostname **epicode.internal**. La comunicazione verrà poi intercettata con **Wireshark**, analizzando il contenuto della richiesta HTTP. Successivamente, l'esercizio sarà ripetuto sostituendo il server HTTP con un server HTTPS, al fine di confrontare i due tipi di traffico e spiegare le principali differenze osservate tra la comunicazione cifrata e quella non cifrata.

In questo scenario la macchina Kali Linux, con indirizzo IP **192.168.50.100**, svolge il ruolo di server web e server DNS, mentre la macchina Windows, con indirizzo IP **192.168.50.102**, funge da client e invia la richiesta tramite browser.

## Configurazioni preliminari

La prima cosa da fare è configurare il nostro ambiente di test. In questo esercizio la macchina Kali lavorerà sia come server web sia come server DNS, risolvendo le richieste della macchina Windows verso **epicode.internal** sul proprio indirizzo IP.

Per prima cosa, su Windows, modificheremo le configurazioni della scheda di rete per indirizzare tutte le **richieste DNS** in uscita verso la macchina Kali, affinché la risoluzione punti al suo indirizzo IP (**192.168.50.100**). Nella scheda **Proprietà** dell'interfaccia di rete, si seleziona **Protocollo Internet versione 4 (TCP/IPv4)** e, cliccando su **Proprietà**, si configura manualmente l'indirizzo IP del server DNS di riferimento. In questo modo, la macchina Windows utilizzerà l'indirizzo IP della macchina Kali come server DNS per la risoluzione dei nomi di dominio all'interno dell'ambiente di laboratorio.



The image shows a screenshot of the 'Internet Protocol Version 4 (TCP/IPv4) Properties' dialog box in Windows. The 'Use the following IP addresses' section is expanded, and the 'DNS settings' section is also expanded. Under 'DNS settings', the radio button 'Use the following DNS server addresses' is selected. The 'Preferred DNS server' field is set to '192.168.50.100'. The 'Alternate DNS server' field is empty, showing only the dots. At the bottom, there is a checkbox for 'Validate settings upon exit' which is unchecked, and buttons for 'OK', 'Annulla', and 'Avanzate...'. The 'OK' button is highlighted with a blue border.

A questo punto la macchina Windows è configurata per indirizzare tutte le richieste DNS verso la macchina Kali, ogni volta che sarà necessaria la risoluzione di un nome di dominio.

Il passo successivo consiste nella configurazione del server web e del server DNS su Kali. A questo scopo sono stati utilizzati due tool: **InetSim** per la gestione del server HTTP e **DNSMasq** per il servizio DNS.

*In un primo momento si è tentato di utilizzare esclusivamente **InetSim** per entrambi i servizi, ma sono sorti alcuni problemi con il modulo DNS a causa di dipendenze Perl non compatibili. Considerando il tempo a disposizione e volendo simulare un approccio realistico, come avviene spesso in contesti lavorativi in cui l'efficienza è fondamentale, si è scelto di adottare **DNSMasq** per la gestione delle richieste DNS.*

La configurazione di **InetSim** per il server web risulta piuttosto semplice. Utilizzando un editor di testo per modificare alcune impostazioni all'interno del file di configurazione `/etc/inetsim/inetsim.conf`, è sufficiente rimuovere il commento dalle voci `start_service http` e `start_service https` e impostare l'indirizzo IP del server che ospita il servizio web, ossia l'indirizzo della macchina Kali.

```
#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
#start_service discard_udp
#start_service quotd_tcp
#start_service quotd_udp
#start_service chargen_tcp
#start_service chargen_udp
#start_service dummy_tcp
#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 192.168.50.100
```

Per la configurazione del servizio **DNS**, è necessario intervenire sul file di configurazione **/etc/dnsmasq.conf**, aggiungendo un **record A** per associare l'hostname **epicode.internal** all'indirizzo IP **192.168.50.100**. In questo modo, ogni richiesta DNS relativa a **epicode.internal** verrà risolta correttamente dalla macchina Kali.

```
# Add domains which you want to force to an IP address here.
# The example below send any host in double-click.net to a local
# web-server.
#address=/double-click.net/127.0.0.1
address=/epicode.internal/192.168.50.100
```

A questo punto non resta che avviare i due servizi configurati. Per avviare **InetSim**, è sufficiente eseguire il seguente comando da terminale:

- **sudo inetsim**

Per quanto riguarda **DNSMasq**, sarà invece necessario avviare il relativo demone tramite il comando:

- **sudo systemctl start dnsmasq**

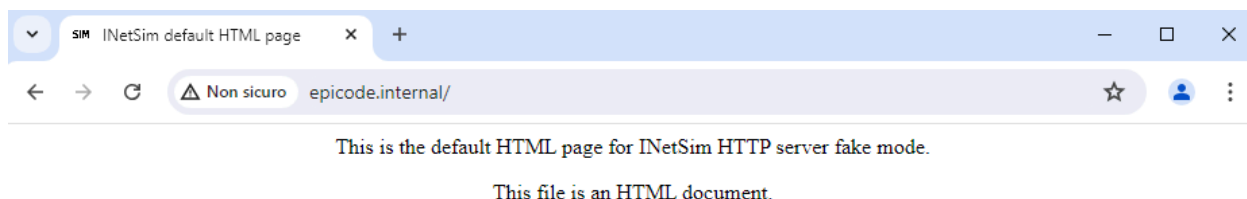
In questo modo entrambi i servizi risulteranno attivi e pronti a gestire le richieste provenienti dalla macchina Windows.

```
(kali@kali)-[~]
$ sudo systemctl status dnsmasq
[sudo] password for kali:
● dnsmasq.service - dnsmasq - A lightweight DHCP and caching DNS server
   Loaded: loaded (/usr/lib/systemd/system/dnsmasq.service; disabled; preset: disabled)
   Active: active (running) since Sat 2025-10-18 10:21:51 EDT; 15s ago
  Invocation: 4010ceb2eb12406c8777f6af7b0d8113
     Docs: man:dnsmasq(8)
   Process: 1580 ExecStartPre=/usr/share/dnsmasq/systemd-helper checkconfig (code=exited, status=0/SUCCESS)
   Process: 1586 ExecStart=/usr/share/dnsmasq/systemd-helper exec (code=exited, status=0/SUCCESS)
   Process: 1593 ExecStartPost=/usr/share/dnsmasq/systemd-helper start-resolvconf (code=exited, status=0/SUCCESS)
  Main PID: 1591 (dnsmasq)
    Tasks: 1 (limit: 4538)
   Memory: 2.2M (peak: 4M)
      CPU: 43ms
   CGroup: /system.slice/dnsmasq.service
           └─1591 /usr/sbin/dnsmasq -x /run/dnsmasq/dnsmasq.pid -u dnsmasq -7 /etc/dnsmasq.d/.dpkg-dist,.dpkg-old,

Oct 18 10:21:51 kali systemd[1]: Starting dnsmasq.service - dnsmasq - A lightweight DHCP and caching DNS server ...
Oct 18 10:21:51 kali dnsmasq[1591]: started, version 2.91 cachesize 150
Oct 18 10:21:51 kali dnsmasq[1591]: DNS service limited to local subnets
Oct 18 10:21:51 kali dnsmasq[1591]: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2 DHCP DHCPv6 no-Lua
Oct 18 10:21:51 kali dnsmasq[1591]: reading /etc/resolv.conf
Oct 18 10:21:51 kali dnsmasq[1591]: using nameserver 192.168.50.1#53
Oct 18 10:21:51 kali dnsmasq[1591]: read /etc/hosts - 7 names
Oct 18 10:21:51 kali systemd[1]: Started dnsmasq.service - dnsmasq - A lightweight DHCP and caching DNS server.

(kali@kali)-[~]
$ 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 2019) ==
Session ID: 2019
Listening on: 192.168.50.100
Real Date/Time: 2025-10-18 10:22:35
Fake Date/Time: 2025-10-18 10:22:35 (Delta: 0 seconds)
Forking services ...
* http_80_tcp - started (PID 2029)
done.
Simulation running.
```

Una volta che entrambi i servizi saranno “**up and running**”, sarà necessario verificare che la risoluzione DNS e la visualizzazione della pagina web avvengano correttamente. Aprendo un browser sulla macchina Windows e navigando verso **http://epicode.internal** (o **https://epicode.internal**), si può osservare che sia la risoluzione del nome sia il caricamento della pagina web avvengono senza problemi.



## Analisi del traffico con Wireshark

Adesso possiamo procedere ad analizzare il traffico di rete con Wireshark. Avviamo il tool sulla nostra macchina Kali e dal client Windows navighiamo verso **http://epicode.internal**.

No.	Time	Source	Destination	Protocol	Length	Info
2710	454.104528247	192.168.50.102	192.168.50.100	HTTP	532	GET / HTTP/1.1
2716	454.125088603	192.168.50.100	192.168.50.102	HTTP	312	HTTP/1.1 200 OK (text/html)
2722	454.164263449	192.168.50.102	192.168.50.100	HTTP	451	GET /favicon.ico HTTP/1.1
2725	454.173759720	192.168.50.100	192.168.50.102	HTTP	252	HTTP/1.1 200 OK (image/x-icon)

Filtrando il traffico **HTTP** con Wireshark, è possibile osservare che la richiesta viene inviata dal client al server, il quale risponde fornendo al client la pagina web richiesta. Analizzando più nel dettaglio lo stack **TCP/IP**, possiamo comprendere meglio cosa accade durante la comunicazione.

A livello **data-link (Layer 2)**, nella richiesta HTTP, è possibile identificare gli indirizzi MAC di sorgente (Windows), e di destinazione (Kali).

```
Frame 2710: 532 bytes on wire (4256 bits), 532 bytes captured (4256 bits) on interface eth0, id 0
  Ethernet II, Src: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1), Dst: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7:eb)
    Destination: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7:eb)
    Source: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1)
      type: IPv4 (0x0800)
      [Stream index: 3]
    Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
    Transmission Control Protocol, Src Port: 49468, Dst Port: 80, Seq: 1, Ack: 1, Len: 478
    Hypertext Transfer Protocol
```

A livello **tre (Network/Internet layer)**, invece, è possibile osservare tutte le dinamiche relative all'indirizzamento IP, ossia come i pacchetti vengano inviati dal client al server e come il server li instrada verso la destinazione corretta.

```

> Frame 2710: 532 bytes on wire (4256 bits), 532 bytes captured (4256 bits) on interface eth0, id 0
> Ethernet II, Src: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1), Dst: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7)
> 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: 518
        Identification: 0x441a (17434)
    > 010. .... = Flags: 0x2, Don't fragment
        ...0 0000 0000 0000 = Fragment Offset: 0
        Time to Live: 128
        Protocol: TCP (6)
        Header Checksum: 0xcebc [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 192.168.50.102
        Destination Address: 192.168.50.100
        [Stream index: 3]
> Transmission Control Protocol, Src Port: 49468, Dst Port: 80, Seq: 1, Ack: 1, Len: 478
> Hypertext Transfer Protocol
```

Salendo ancora nello stack, a livello **quattro (Transport layer)** possiamo osservare che il protocollo di trasporto utilizzato è **TCP**. La porta di origine è una porta **random** assegnata dal sistema operativo del client, mentre la porta di destinazione corrisponde alla porta standard del servizio HTTP, ossia la **porta 80**.

```

> Frame 2710: 532 bytes on wire (4256 bits), 532 bytes captured (4256 bits) on interface eth0, id 0
> Ethernet II, Src: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1), Dst: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7)
> Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
> Transmission Control Protocol, Src Port: 49468, Dst Port: 80, Seq: 1, Ack: 1, Len: 478
    Source Port: 49468
    Destination Port: 80
    [Stream index: 40]
    [Stream Packet Number: 4]
    > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 478]
    Sequence Number: 1 (relative sequence number)
    Sequence Number (raw): 83894214
    [Next Sequence Number: 479 (relative sequence number)]
    Acknowledgment Number: 1 (relative ack number)
    Acknowledgment number (raw): 4205411964
    0101 .... = Header Length: 20 bytes (5)
    > Flags: 0x018 (PSH, ACK)
    Window: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x3e23 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    > [Timestamps]
    > [SEQ/ACK analysis]
    TCP payload (478 bytes)
> Hypertext Transfer Protocol
```

Infine, al livello **applicativo (Application layer)** è possibile osservare l'effettivo contenuto della richiesta HTTP.

```

Frame 2710: 532 bytes on wire (4256 bits), 532 bytes captured (4256 bits) on interface eth0, id 0
Ethernet II, Src: PCSSystemec-63:28:11 (08:00:27:63:28:11), Dst: PCSSystemec-f4:c7:b8 (08:00:27:f4:c7:b8)
Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
Transmission Control Protocol, Src Port: 49468, Dst Port: 80, Seq: 1, Ack: 1, Len: 478
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
Host: epicode.intelna.vn\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\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/64.0.3408.136 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: it-IT;q=0.9,en-US;q=0.8,en;q=0.7\r\n
\r\n
[Response in frame: 2716]
Full request URL: http://epicode.intelna.vn/

```

```

Frame 2760: 312 bytes on wire (2496 bits), 312 bytes captured (2496 bits) on interface eth0, id 0
    Ethernet II, Src: PCSysstemec-f4c7eb (08:00:27:f4:c7:eb), Dst: PCSysstemec-03:28:a1 (08:00:27:03:28:a1), Internet Protocol Version 4, Src: 192.168.50.100, Dst: 192.168.50.102
    Hypertext Transfer Protocol, Src Port: 80, Dst Port: 43408, Seq: 1031, Ack: 479, Len: 258
        [2 Reassembled TCP Segments (488 bytes): #2215(150), #2216(258)]
        > HTTP/1.1 200 OK\r\n
          Server: InetSim\r\n
          Content-Type: text/html\r\n
          Connection: close\r\n
          Content-Length: 258\r\n
          Date: Sun, 19 Oct 2025 07:19:45 GMT\r\n
          \r\n
          [Request in frame: 2710]
          [time since request: 0.020560356 seconds]\r\n
          Request URI: /
          Full request URI: http://epicdoc.intelvald.com/
          File Data: 258 bytes
          Line-based text data: text/html (40 lines)
          <html>\n
            <title>InetSim default HTML page</title>\n
            </head>\n
            <body>\n
              <p>\n
                <align=center>This is the default HTML page for InetSim HTTP server file mode.</p>\n
                <p>\n
                  <align=center>This file is an HTML document.</p>\n
              \n
            </body>\n
            </html>\n

```

379	5.91955029	192.168.50.192	192.168.50.192	TLSv1.3	1876 Client Hello (SNI=epicode.internal)
383	3.35313421	192.168.50.192	192.168.50.192	TLSv1.3	1487 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data
384	5.33639430	192.168.50.192	192.168.50.192	TLSv1.3	134 Change Cipher Spec, Application Data
385	5.336772083	192.168.50.192	192.168.50.192	TLSv1.3	762 Application Data
386	5.541058879	192.168.50.192	192.168.50.192	TLSv1.3	309 Application Data
387	5.557421326	192.168.50.192	192.168.50.192	TLSv1.3	786 Application Data, Application Data, Application Data, Application Data

Per prima cosa, si osserva che, prima dello scambio effettivo delle informazioni, avviene tra le parti comunicanti un **TLS handshake**.



Durante questa fase, il client si presenta al server, il server risponde e propone al client un protocollo di cifratura. Successivamente avviene lo scambio delle chiavi, dopodiché la comunicazione conseguente risulterà **cifrata**.

A livello **2 (Data-Link)** e **3 (Network)** non si riscontrano particolari differenze rispetto al traffico HTTP, mentre a livello **4 (Transport)** si nota che la porta di destinazione della richiesta è la **443**, ossia la porta standard del protocollo applicativo HTTPS.

```
Frame 385: 762 bytes on wire (6096 bits), 762 bytes captured (6096 bits) on interface eth0, id 0
Ethernet II, Src: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1), Dst: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7:eb)
Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
Transmission Control Protocol, Src Port: 49454, Dst Port: 443, Seq: 1903, Ack: 1444, Len: 708
  Source Port: 49454
  Destination Port: 443
  [Stream index: 2]
  [Stream Packet Number: 8]
```

Al livello **Application**, è possibile dedurre che il tipo di comunicazione è HTTP (**Application Data Protocol: Hypertext Transfer Protocol**), ma il contenuto dei dati è **cifrato**. Di conseguenza, il traffico può essere intercettato, ma non interpretato. La crittografia garantisce la **confidenzialità** dei dati, uno dei tre principi fondamentali della **CIA Triad**.

```
Frame 385: 762 bytes on wire (6096 bits), 762 bytes captured (6096 bits) on interface eth0, id 0
Ethernet II, Src: PCSSystemtec_63:28:a1 (08:00:27:63:28:a1), Dst: PCSSystemtec_f4:c7:eb (08:00:27:f4:c7:eb)
Internet Protocol Version 4, Src: 192.168.50.102, Dst: 192.168.50.100
Transmission Control Protocol, Src Port: 49454, Dst Port: 443, Seq: 1903, Ack: 1444, Len: 708
Transport Layer Security
  TLSv1.2 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 703
    Encrypted Application Data [...] e8531b867369d2d1231bd5ca01c1335f45cbc10c36723154b63be8ae1df6b13782c
    [Application Data Protocol: Hypertext Transfer Protocol]
```

In conclusione, l'esercizio ha permesso di osservare direttamente le differenze tra comunicazioni HTTP e HTTPS, evidenziando come la cifratura garantisca la sicurezza e la confidenzialità dei dati scambiati.

Nicola Guidi, Campiglia Marittima (LI)  
19/10/2025