# Introduzione all'Hacking - Progetto Finale W4D4

**RICHIESTA**

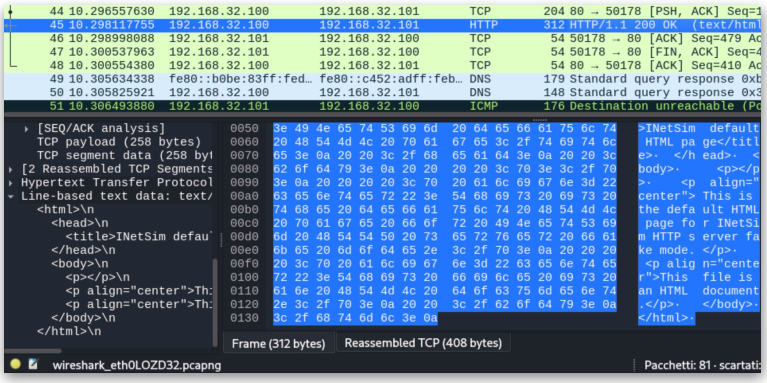
Il progetto finale del modulo W4D4 è incentrato sulla costruzione, nel laboratorio virtuale, di un'architettura client-server in cui un client con indirizzo 192.168.32.101 (VM Windows) richieda, tramite web browser, una risorsa all'hostname epicode.internal che risponde all'indirizzo 192.168.32.100 (VM Kali Linux).

Nella seconda fase bisogna, tramite il software Wireshark, intercettare il traffico ed estrapolare le informazioni relative al pacchetto. Bisogna inoltre mettere in evidenza le differenze fra la stessa richiesta su server HTTPS e su server HTTP

~~~

**SOLUZIONE**

Per la risoluzione dell'esercizio è necessario, prima di tutto, configurare in modo corretto le due macchine virtuali del laboratorio. Bisogna attivare poi un server HTTPS, un server HTTP e il servizio DNS. Tutte le richieste potrebbero essere evase utilizzando il software iNetSim, ma un bug del servizio DNS mi ha costretto a utilizzare un altro software (spiegazioni più dettagliate nel blocco Dettagli Tecnici).

Utilizzando Wireshark ho intercettato sia la richiesta HTTPS che quella HTTP, evidenziando alcune importanti differenze. Nello specifico, sono due i campi dove le differenze sono più nette:

- La visibilità dei dati, visto che la richiesta HTTPS è completamente crittografata mentre quella HTTP è in chiaro. Questo vuol dire che la richiesta HTTP è leggibile tramite Wireshark

- Le porte utilizzate: Tutto il traffico HTTPS passa dalla porta 443, mentre il traffico HTTP passa dalla porta 80

~~~

**IMMAGINE**

Nell'immagine allegata, possiamo vedere come la comunicazione HTTP sia leggibile tramite Wireshark. La vedremo meglio alla fine del blocco Dettagli Tecnici



**DETTAGLI TECNICI**

Il primo passo per la risoluzione dell'esercizio è quello di mettermi nella condizione di rispetto dei requisiti richiesti. Ho innanzitutto assegnato i giusti IP alle macchine virtuali. Nello specifico, ho assegnato alla VM con Kali Linux l'indirizzo IP **192.168.32.100/24** e alla VM con Windows l'indirizzo IP **192.168.32.101/24**. Ho assegnato poi per entrambe le macchine il giusto indirizzo di Gateway e, per la macchina Windows, ho assegnato come indirizzo DNS quello della VM Kali Linux.

Per avere la certezza di aver configurato tutto correttamente, ho lanciato una richiesta di Ping dalla VM Kali Linux verso quella Windows. La risposta della VM Windows certifica il fatto che le due VM sono connesse fra loro.

```
┌──(pagizza㉿kali)-[~]
└─$ pingping 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=1.54 ms
64 bytes from 192.168.32.101: icmp_seq=2 ttl=128 time=2.26 ms
64 bytes from 192.168.32.101: icmp_seq=3 ttl=128 time=2.09 ms
64 bytes from 192.168.32.101: icmp_seq=4 ttl=128 time=1.62 ms
64 bytes from 192.168.32.101: icmp_seq=5 ttl=128 time=0.602 ms
64 bytes from 192.168.32.101: icmp_seq=6 ttl=128 time=1.86 ms
64 bytes from 192.168.32.101: icmp_seq=7 ttl=128 time=2.03 ms
^C
── 192.168.32.101 ping statistics ──
7 packets transmitted, 7 received, 0% packet loss, time 6028ms
rtt min/avg/max/mdev = 0.602/1.713/2.259/0.511 ms
```

Gli altri requisiti richiesti sono l'attivazione di un server di un HTTPS Server, un servizio DNS per la risoluzione dei nomi di dominio e un HTTP Server, indispensabile nella seconda parte dell'esercitazione. A tal proposito, utilizzerò il software **iNetSim**, che consente di simulare facilmente servizi di rete in un ambiente isolato dedicato a test e analisi. iNetSim è già installato sulla VM con Kali Linux, quindi l'ho solo configurato per rispondere alle richieste dell'esercizio.

Nello specifico, ho innanzitutto attivato i servizi di rete DNS, HTTPS e HTTP dal file di configurazione **/etc/inetsim/inetsim.conf**

```
GNU nano 8.3                                    /etc/inetsim/inetsim.conf
##########################################
# 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
```

Successivamente c'è da modificare la parte relativa al dns_static per far sì che il servizio DNS "traduca" la richiesta di "epicode.internal" come una richiesta all'indirizzo IP **192.168.32.100**

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

Nonostante la configurazione sia corretta, la VM con Windows non riuscirà a risolvere la richiesta a epicode.internal. Questo è dato da un malfunzionamento della sezione DNS all'interno di iNetSim. Ho dovuto quindi trovare una via alternativa, passata dall'installazione del pacchetto **DNSMasq**

Ho quindi configurato correttamente DNSMasq facendo in modo che il servizio "traduca" la richiesta di "epicode.internal" come una richiesta all'indirizzo IP **192.168.32.100.** Ho controllato il funzionamento della parte DNS eseguendo il comando "nslookup epicode.internal" dalla VM con Windows. La schermata di Windows ci conferma il funzionamento del DNS. È importante modificare la configurazione di iNetSim in modo che non si occupi più del servizio DNS. Questo evita conflitti con DNSMasq.

```
C:\>ipconfig /flushdns

Configurazione IP di Windows

Cache del resolver DNS svuotata.

C:\>nslookup epicode.internal
Server:  UnKnown
Address:  192.168.32.100

Nome:    epicode.internal
Address:  192.168.32.100
```

A questo punto tutte le richieste preliminari dell'esercizio sono rispettate. La parte successiva dell'esercitazione è utilizzare Wireshark per intercettare la comunicazione fra le due VM quando Windows naviga verso l'indirizzo epicode.internal. Prima di procedere, è importante controllare che i servizi iNetSim e DNSMasq siano attivo (**sudo systemctl status inetsim** e **sudo systemctl status dnsmasq)**.

Dalla VM di Kali Linux ho avviato Wireshark, mentre dalla VM di Windows ho aperto il browser Edge e navigato verso https://epicode.internal. Wireshark catturerà tutto il traffico in arrivo. Nella schermata di Wireshark è comodo inserire il filtro **ip.src == 192.168.32.101**, così da isolare solo il traffico proveniente dalla VM con Windows.

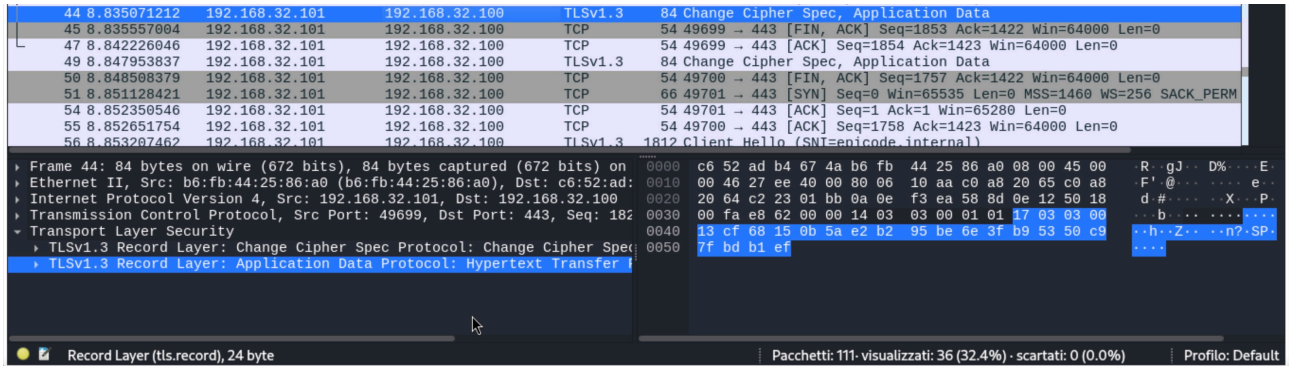
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 31 | 8.777865796 | 192.168.32.101 | 192.168.32.100 | TCP | 66 | 49699 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM |
| 32 | 8.778020337 | 192.168.32.100 | 192.168.32.101 | TCP | 66 | 443 → 49699 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_P |
| 33 | 8.778923129 | 192.168.32.101 | 192.168.32.100 | TCP | 54 | 49699 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0 |
| 34 | 8.790171296 | 192.168.32.101 | 192.168.32.100 | TLSv1.3 | 1876 | Client Hello (SNI=epicode.internal) |
| 35 | 8.790194837 | 192.168.32.100 | 192.168.32.101 | TCP | 54 | 443 → 49699 [ACK] Seq=1 Ack=1823 Win=62464 Len=0 |
| 36 | 8.798976962 | 192.168.32.101 | 192.168.32.100 | DNS | 76 | Standard query 0x0206 A epicode.internal |
| 37 | 8.799177671 | 192.168.32.100 | 192.168.32.101 | DNS | 92 | Standard query response 0x0206 A epicode.internal A 192.168.32.100 |
| 38 | 8.819871046 | 192.168.32.100 | 192.168.32.101 | TLSv1.3 | 1475 | Server Hello, Change Cipher Spec, Application Data, Application Da |
| 39 | 8.831690671 | 192.168.32.101 | 192.168.32.100 | TCP | 66 | 49700 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM |
| 40 | 8.831817379 | 192.168.32.100 | 192.168.32.101 | TCP | 66 | 443 → 49700 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_P |
| 41 | 8.832395587 | 192.168.32.101 | 192.168.32.100 | TCP | 54 | 49700 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0 |
| 42 | 8.833592004 | 192.168.32.101 | 192.168.32.100 | TLSv1.3 | 1780 | Client Hello (SNI=epicode.internal) |
| 43 | 8.833604462 | 192.168.32.100 | 192.168.32.101 | TCP | 54 | 443 → 49700 [ACK] Seq=1 Ack=1727 Win=62592 Len=0 |
| 44 | 8.835071212 | 192.168.32.101 | 192.168.32.100 | TLSv1.3 | 84 | Change Cipher Spec, Application Data |
| 45 | 8.835557004 | 192.168.32.101 | 192.168.32.100 | TCP | 54 | 49699 → 443 [FIN, ACK] Seq=1853 Ack=1422 Win=64000 Len=0 |
| 46 | 8.841318379 | 192.168.32.100 | 192.168.32.101 | TCP | 54 | 443 → 49699 [FIN, ACK] Seq=1422 Ack=1854 Win=62464 Len=0 |
| 47 | 8.842226046 | 192.168.32.101 | 192.168.32.100 | TCP | 54 | 49699 → 443 [ACK] Seq=1854 Ack=1423 Win=64000 Len=0 |
| 48 | 8.846530004 | 192.168.32.100 | 192.168.32.101 | TLSv1.3 | 1475 | Server Hello, Change Cipher Spec, Application Data, Application Da |
| 49 | 8.847953837 | 192.168.32.101 | 192.168.32.100 | TLSv1.3 | 84 | Change Cipher Spec, Application Data |
| 50 | 8.848508379 | 192.168.32.101 | 192.168.32.100 | TCP | 54 | 49700 → 443 [FIN, ACK] Seq=1757 Ack=1422 Win=64000 Len=0 |
| 51 | 8.851128421 | 192.168.32.101 | 192.168.32.100 | TCP | 66 | 49701 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM |

La prima considerazione che possiamo fare è che evidenziando un pacchetto è facile risalire al MAC Address sia della VM con Windows che quello della VM con Linux. Nello specifico, nel blocco in basso a sinistra, alla riga Ethernet II possiamo leggere i valori Src e Dst. Il primo è il MAC Address della scheda di rete virtuale su VM Windows 11 (**b6:fb:44:25:86:a0**), il secondo è il MAC Address della scheda di rete virtuale su VM Kali Linux (**c6:52:ad:b4:67:4a**)

Dando uno sguardo in generale, possiamo poi vedere che nelle righe 31,32 e 33 c'è stato il processo di three-way handshake, con la VM Windows che inviato un pacchetto SYN per stabilire la connessione, la VM Kali Linux che alla riga 32 risponde positivamente con un pacchetto SYN-ACK e, alla riga 33, il pacchetto ACK inviato dalla VM Windows che stabilisce il buon esito della connessione.

Le successive righe mostrano invece l'inizio della comunicazione HTTPS fra la VM Windows e quella Linux. In particolare, la riga 34 è il primo messaggio inviato dal client durante la fase di negoziazione TLS per stabilire una connessione sicura. Le righe 38 e 44 completano la negoziazione TLS e lo scambio di dati criptati.

Nello specifico, il pacchetto alla riga 44 mostra il contenuto della richiesta HTTPS, evidenziando che si tratta di una comunicazione criptata (Encrypted) e che utilizza il protocollo Hypertext Transfer Protocol. Essendo una richiesta HTTPS e quindi criptata, non è possibile vederne il contenuto in chiaro, ma si può presupporre che sia una richiesta HTTP GET per la pagina web "epicode.internal"




Sono poi passato all'utilizzo di un server HTTP. La più grande differenza è che il traffico non passa più dalla porta 443, ma dalla porta 80. Un'altra importante differenza è data dal fatto che la comunicazione HTTP non è criptata, quindi potrò osservare il contenuto delle richieste e delle risposte in chiaro.

Come possiamo vedere dalle immagini successive, alle righe 37 e 38 c'è l'interrogazione del server DNS, con la "traduzione" di epicode.internal in indirizzo IP. Dalla riga 39 alla 41 c'è il processo di three-way handshake che funziona allo stesso modo di quanto visto in precedenza. Dalla riga 42 parte l'effettiva comunicazione, con la risposta HTTP che finisce alla riga 45. Guardando al contenuto dei pacchetti 44 e 45, possiamo vedere in chiaro l'intero contenuto della risposta HTTP.