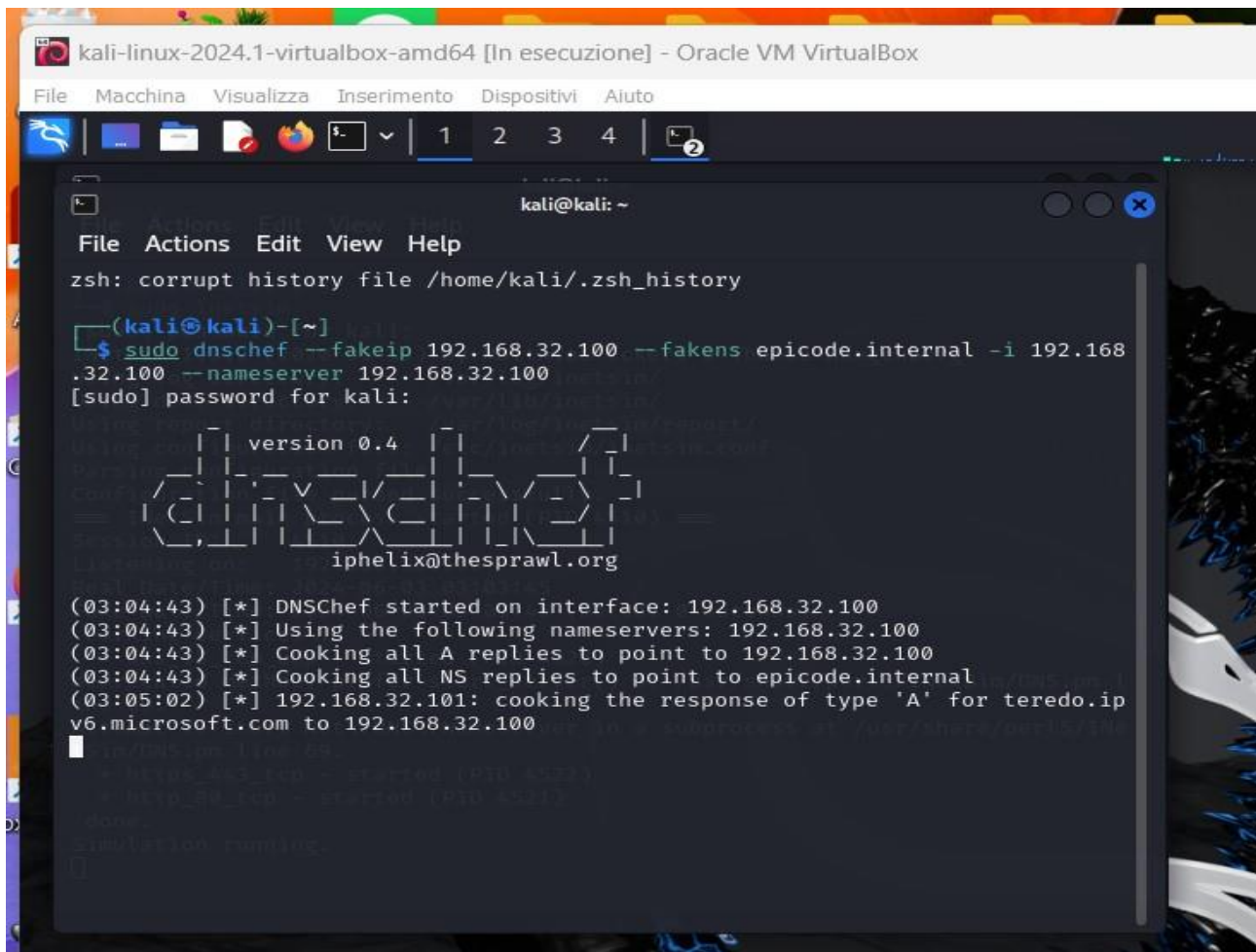
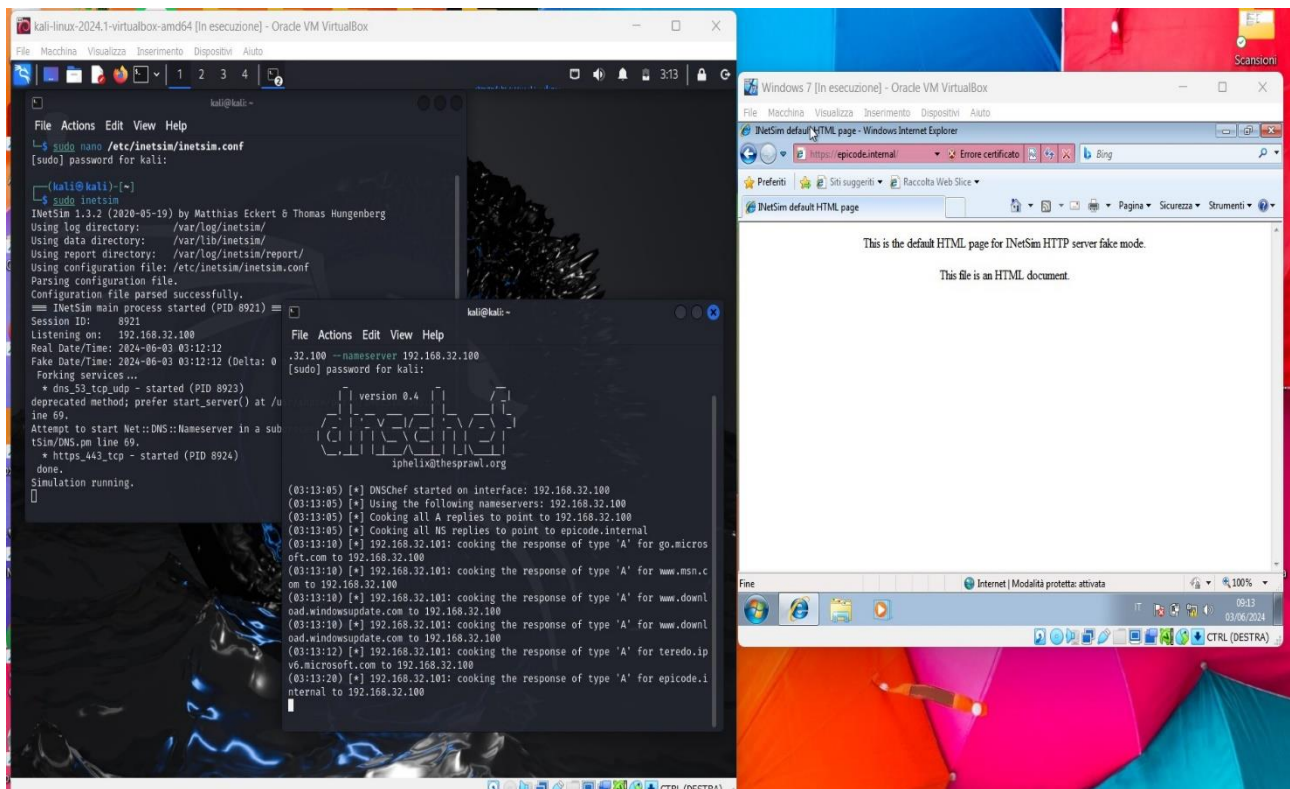


W4D4 -Esame del modulo 1 "introduzione all'ethical hacking" di Maria Zanchetta.

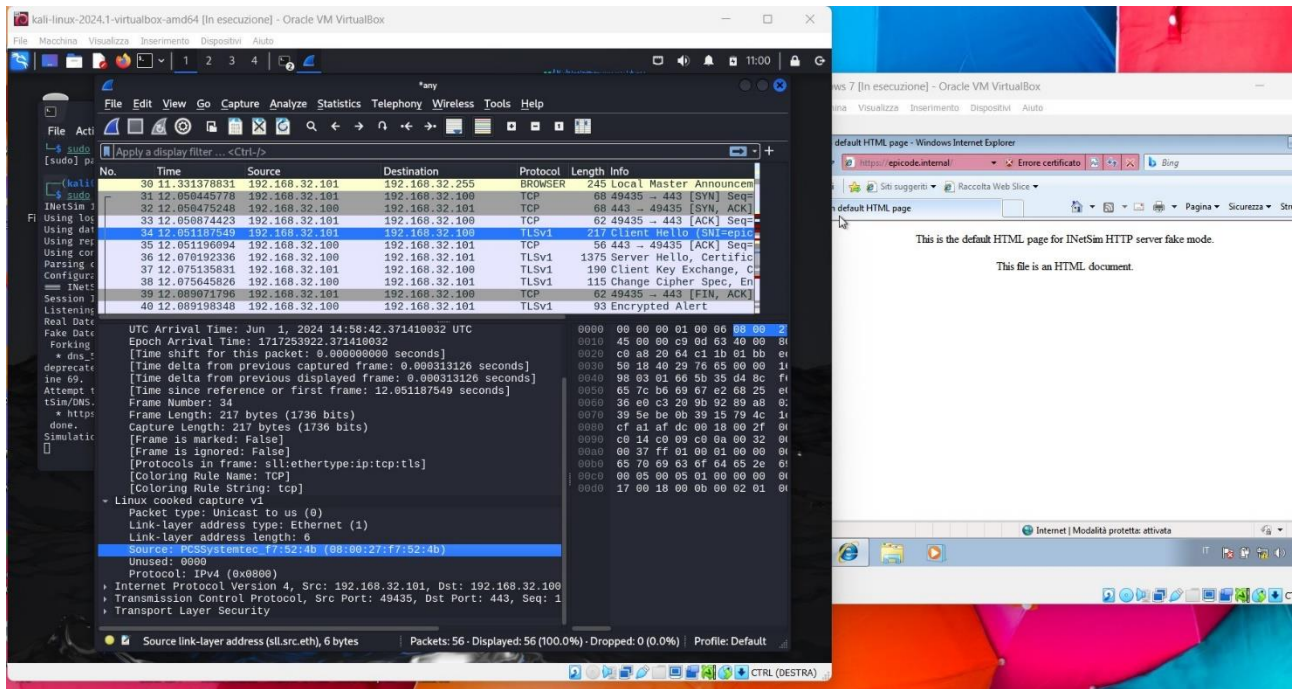
Per prima cosa ho configurato correttamente inetsim, attivando i servizi dns e https. Notando dei problemi nella risoluzione corretta di epicode.internal, ho optato per l'utilizzo del DNS proxy DNSChef, configurandolo in questo modo.

The image shows a screenshot of a Kali Linux virtual machine running in Oracle VM VirtualBox. The terminal window is titled 'kali@kali: ~' and shows the execution of the 'dnschef' command. The command is: `sudo dnschef --fakeip 192.168.32.100 --fakens epicode.internal -i 192.168.32.100 --nameserver 192.168.32.100`. The terminal output shows the version (0.4), the IP address (192.168.32.100), and the nameserver (192.168.32.100). It also shows the command 'dnschef' being executed and the output 'DNSChef started on interface: 192.168.32.100'. The terminal output continues with: `(03:04:43) [*] Using the following nameservers: 192.168.32.100`, `(03:04:43) [*] Cooking all A replies to point to 192.168.32.100`, `(03:04:43) [*] Cooking all NS replies to point to epicode.internal`, and `(03:05:02) [*] 192.168.32.101: cooking the response of type 'A' for teredo.ipv6.microsoft.com to 192.168.32.100`. The terminal output ends with 'Simulation running'.

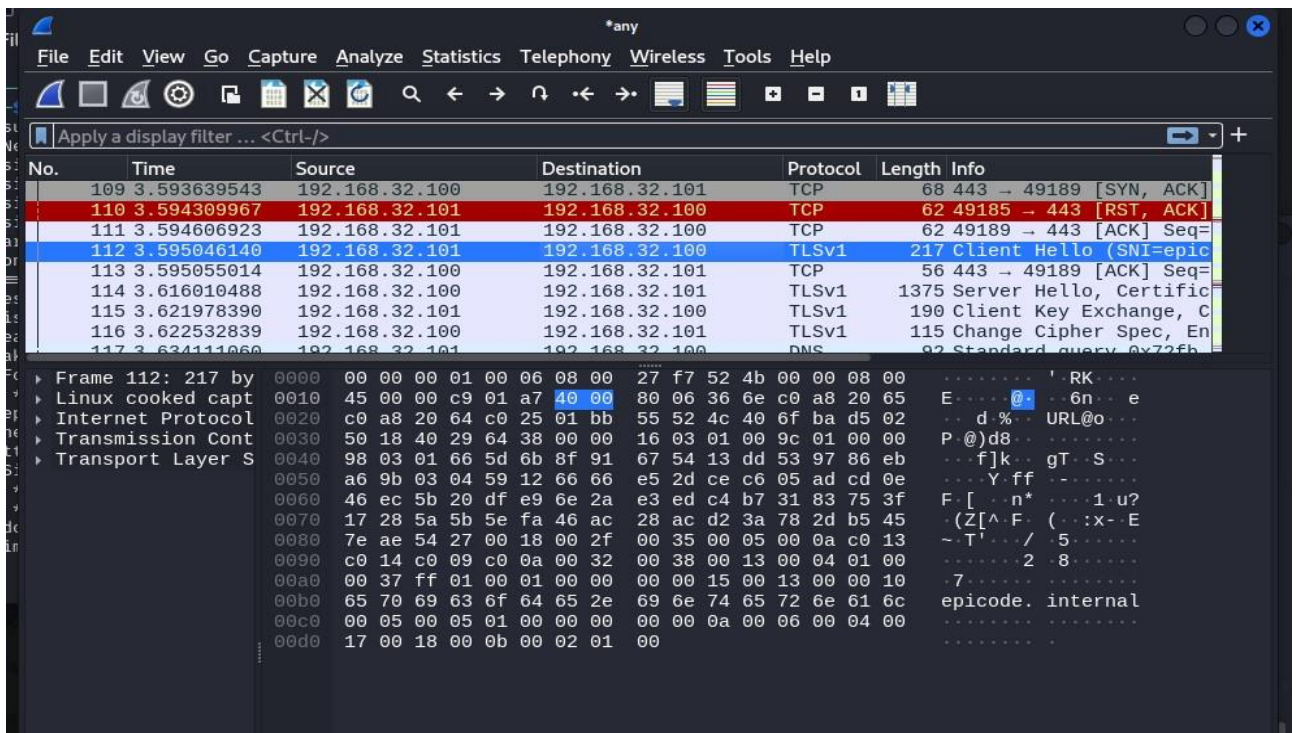
Dopo aver avviato la simulazione con inetsim con i server DNS e HTTPS attivi e aver avviato DNSChef, ho verificato che epicode.internal viene risolto correttamente. Questo è evidente nello screenshot seguente, che dimostra anche il funzionamento dell'architettura client-server, in cui il client è Windows 7 e il server è Kali Linux.



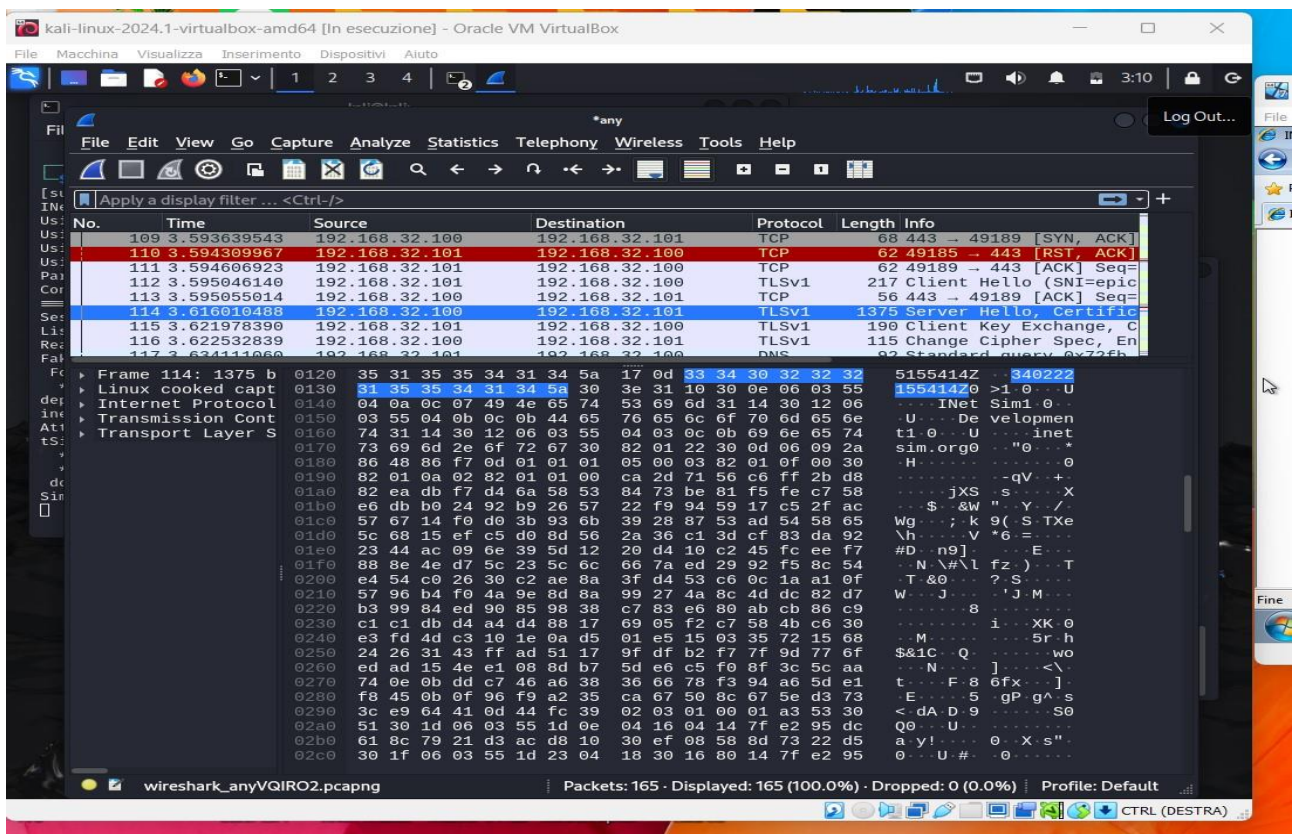
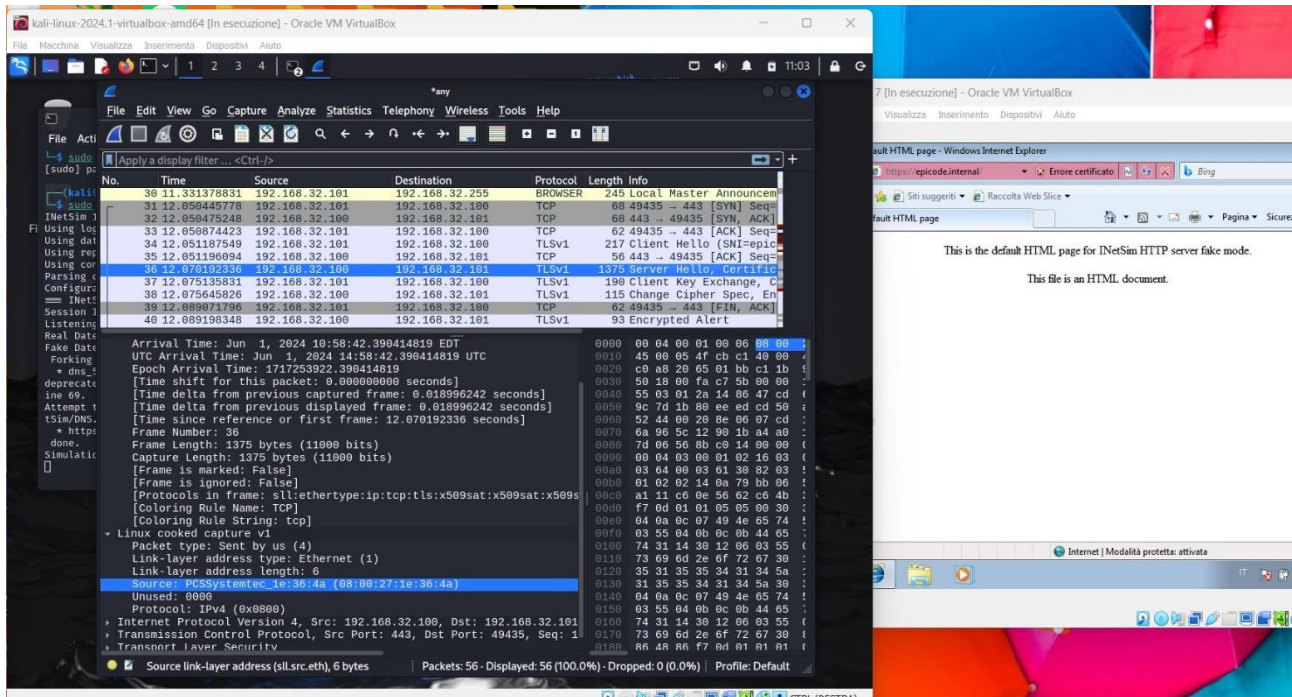
Lo screenshot seguente dimostra i pacchetti del traffico catturati con Wireshark e sono subito evidenti le differenze tra http ed HTTPS, che verranno approfondite alla fine dell'esercizio pratico. Si nota subito che HTTPS è criptato, perché dopo il three-way handshake, necessario per stabilire un canale di comunicazione in base agli standard del protocollo del layer di trasporto TCP, vengono scambiati dei pacchetti TLSv1. TLS, ovvero Transport Layer Security, è un protocollo crittografico tipico di HTTPS che permette una comunicazione sicura end-to-end tra client e server ed è l'evoluzione di SSL, ossia Secure Sockets Layer. TLS va a garantire l'integrità dei dati, previene il tampering, ovvero la manomissione dei dati, e l'autenticazione TLS è unilaterale, ovvero solo il server si autentica presso il client. Il browser valida il certificato del server controllando la firma digitale dei certificati. Nel caso di questo esercizio, lo screenshot seguente indica la request che il client Windows fa al server Kali. Si nota che l'IP source è quello di Windows e l'IP destination è quello del server Kali ed ho evidenziato anche l'indirizzo MAC sorgente, che è quello di Windows, mentre il MAC di destinazione è quello del server Kali.



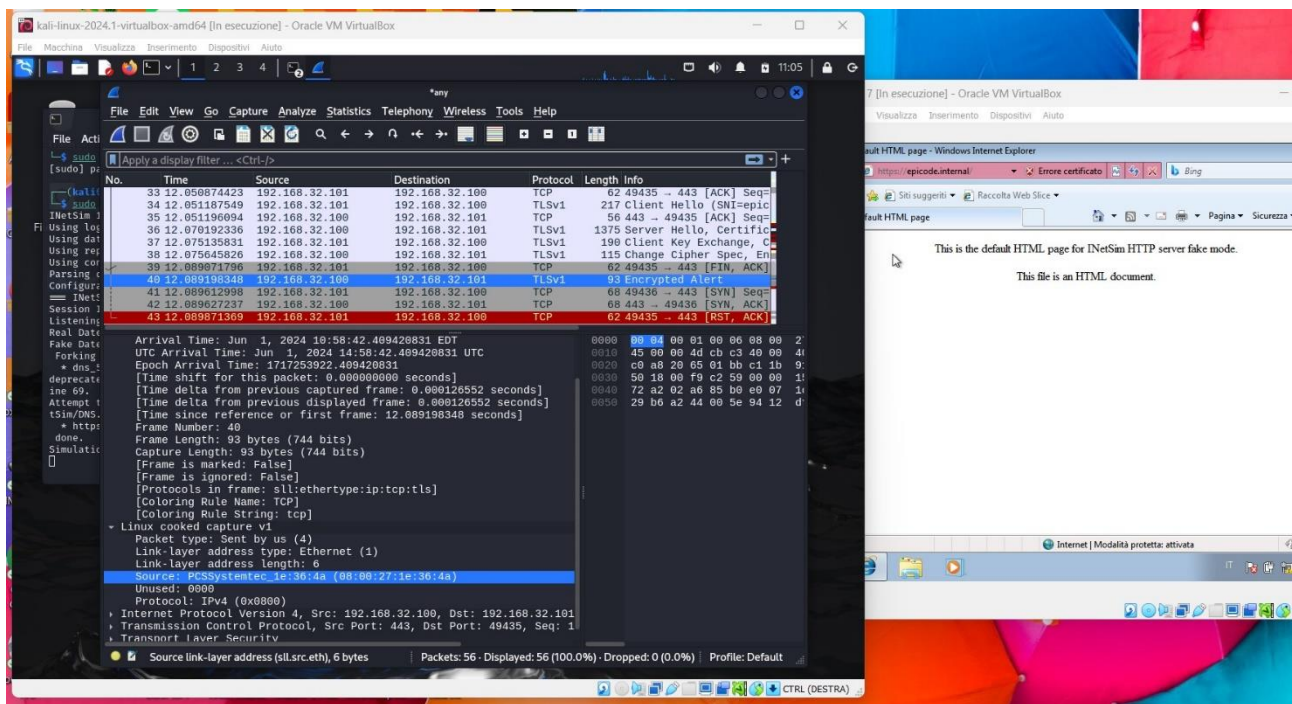
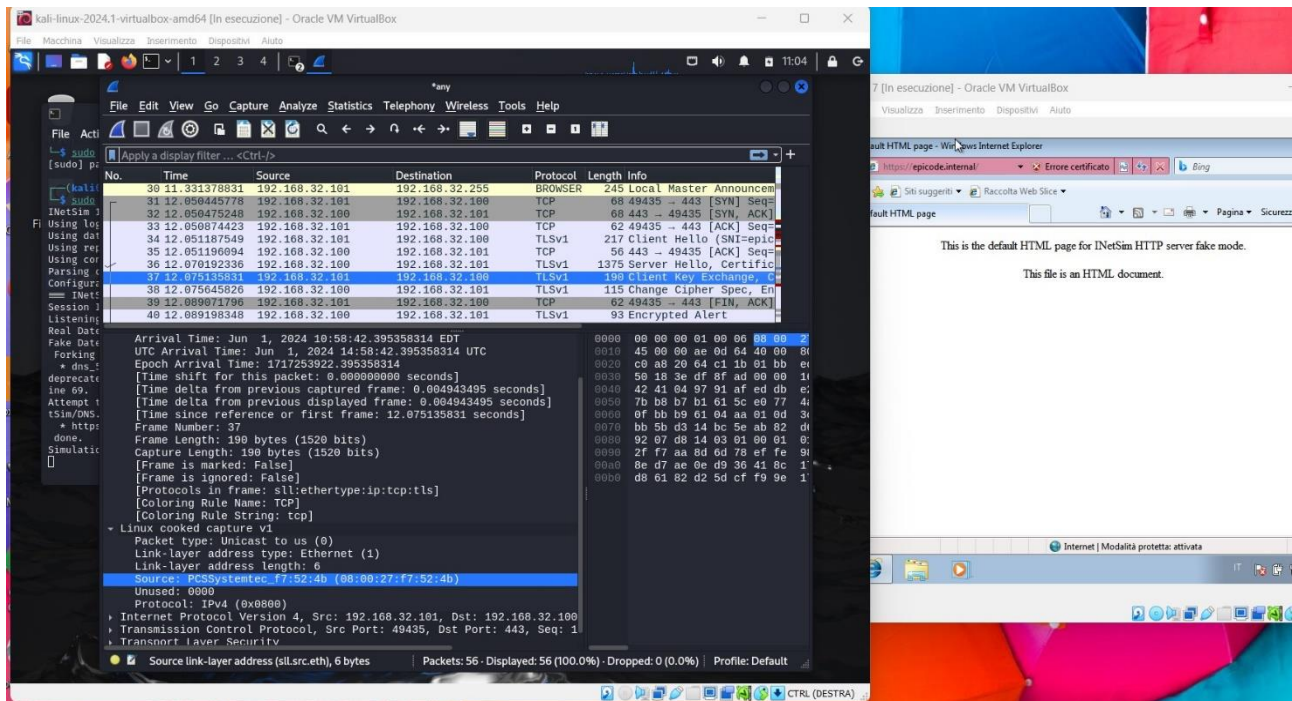
I messaggi tra client e server sono cifrati, infatti ho trovato il messaggio della request cifrato in basso a destra.



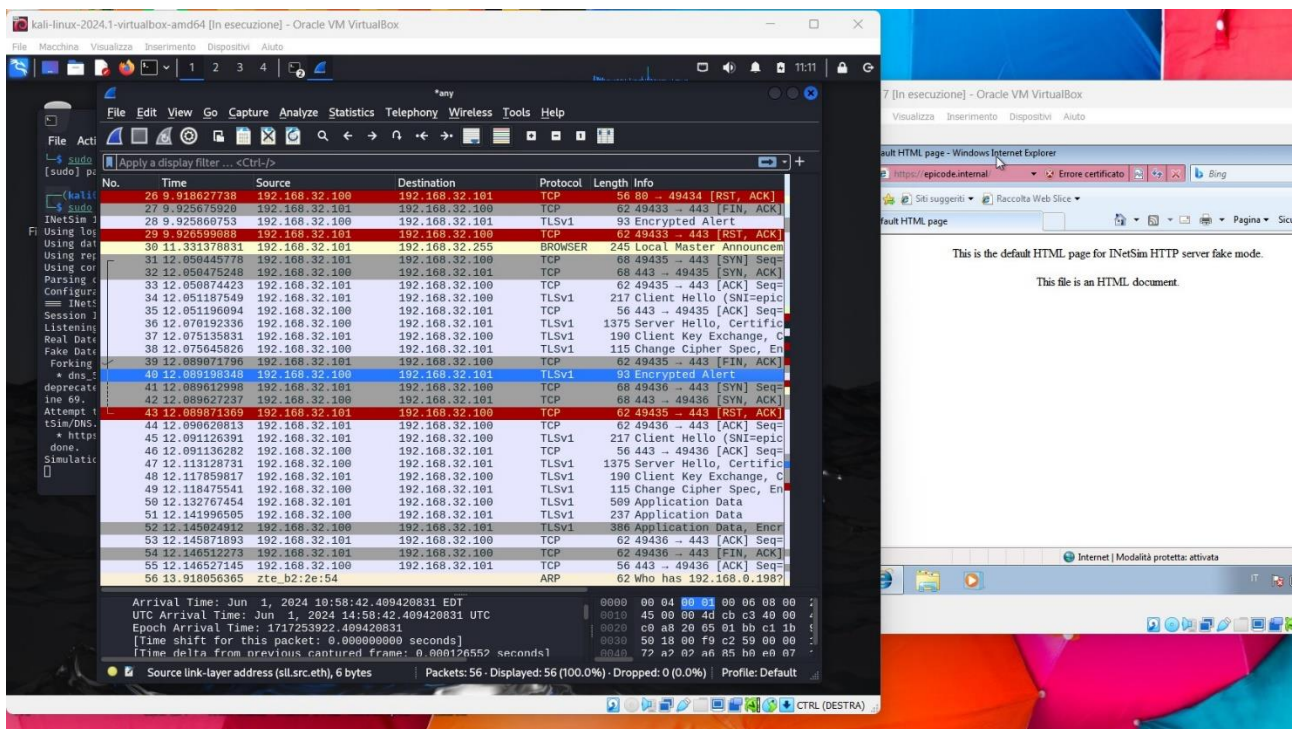
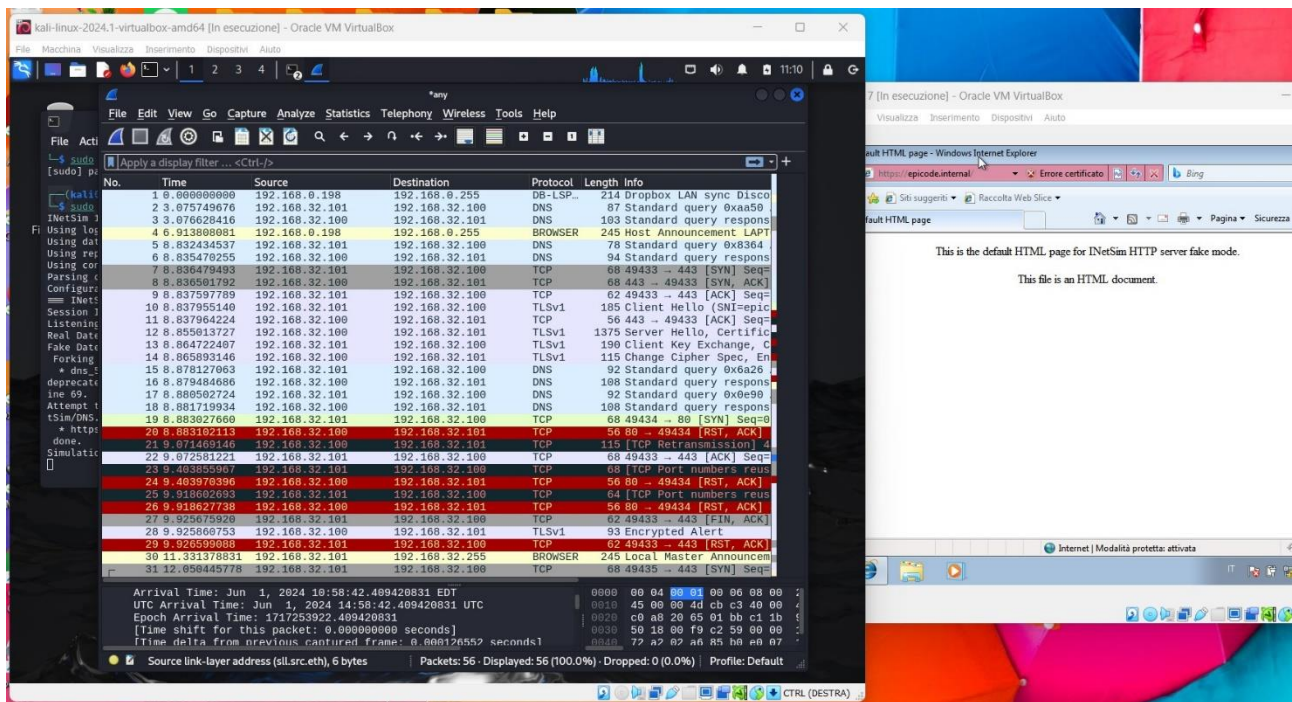
Lo screenshot successivo evidenzia la response del server Kali, per questo l'indirizzo IP di origine è quello del server Kali, il MAC source è quello di Kali evidenziato, l'indirizzo IP di destinazione è quello del client Windows ed anche il MAC di destinazione è quello di Windows. Dall'analisi dei pacchetti emerge anche l'uso della porta 443, la porta generalmente utilizzata per HTTPS. Anche in questo caso, la response è cifrata, proprio come nel caso della request e il pacchetto scambiato è ancora un pacchetto TLSv1.



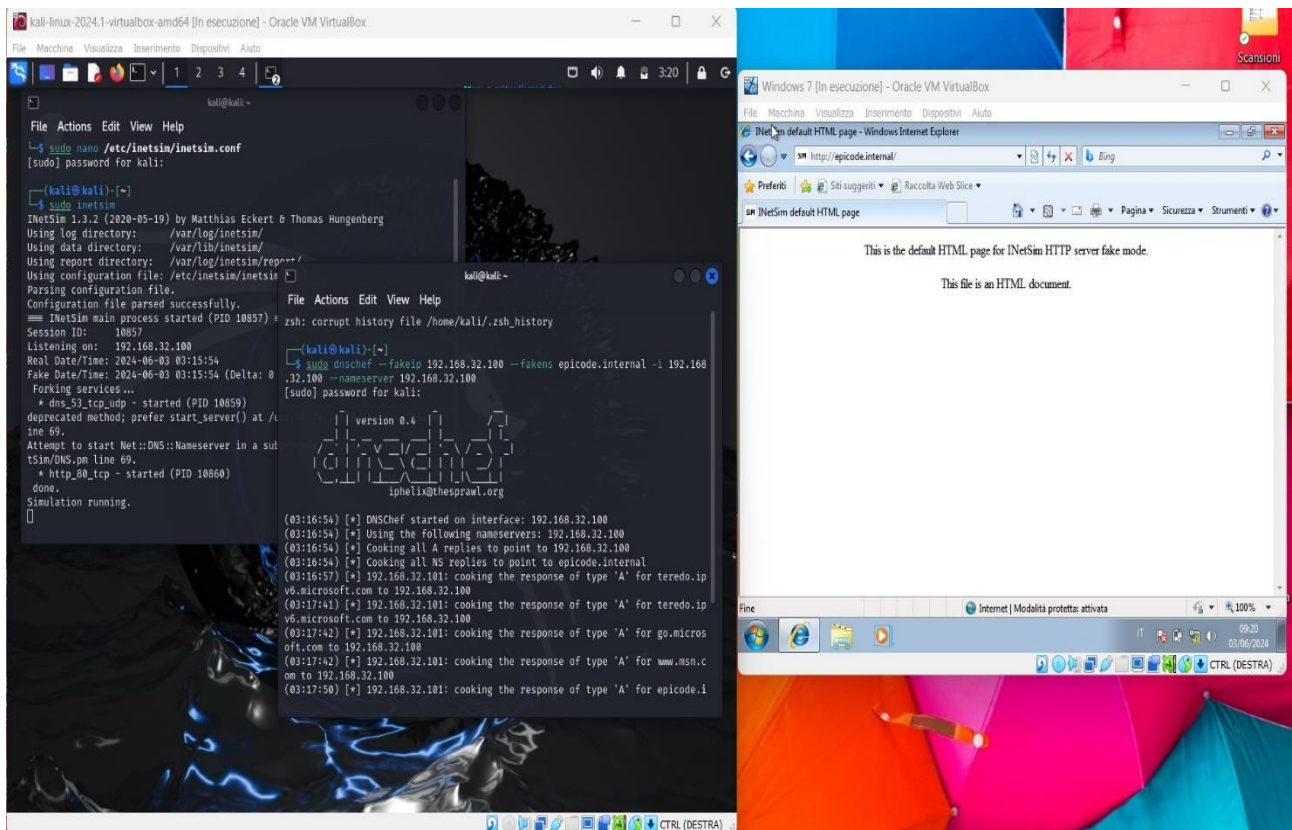
Lo screenshot che segue evidenzia lo scambio delle chiavi per la crittografia, mentre lo screenshot ancora successivo dimostra ulteriormente come la comunicazione sia criptata.



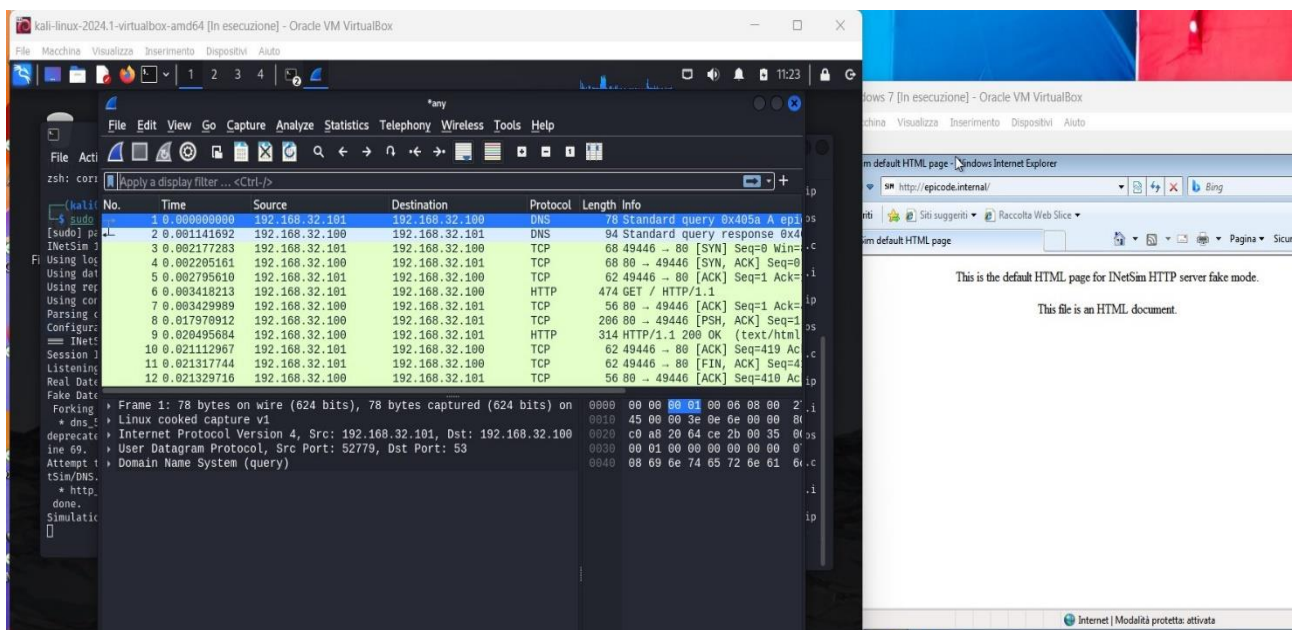
Ho riportato anche tutti i pacchetti catturati per mostrare le connessioni criptate tipiche di HTTPS e i problemi riscontrati con il certificato di sicurezza mancante del sito ricercato.



Il prossimo screenshot dimostra la risoluzione corretta di `epicode.internal` anche utilizzando un server HTTP. Per prima cosa, ho avviato `inetsim` con la configurazione corretta, sostituendo il server HTTPS con il server HTTP, poi ho avviato `DNSchef` per la risoluzione corretta di `epicode.internal`. Ho raggiunto la pagina web senza riscontrare i problemi con il certificato di sicurezza che si erano verificati con HTTPS, proprio perché il protocollo HTTP non è cifrato e il browser non richiede un certificato di sicurezza al sito.

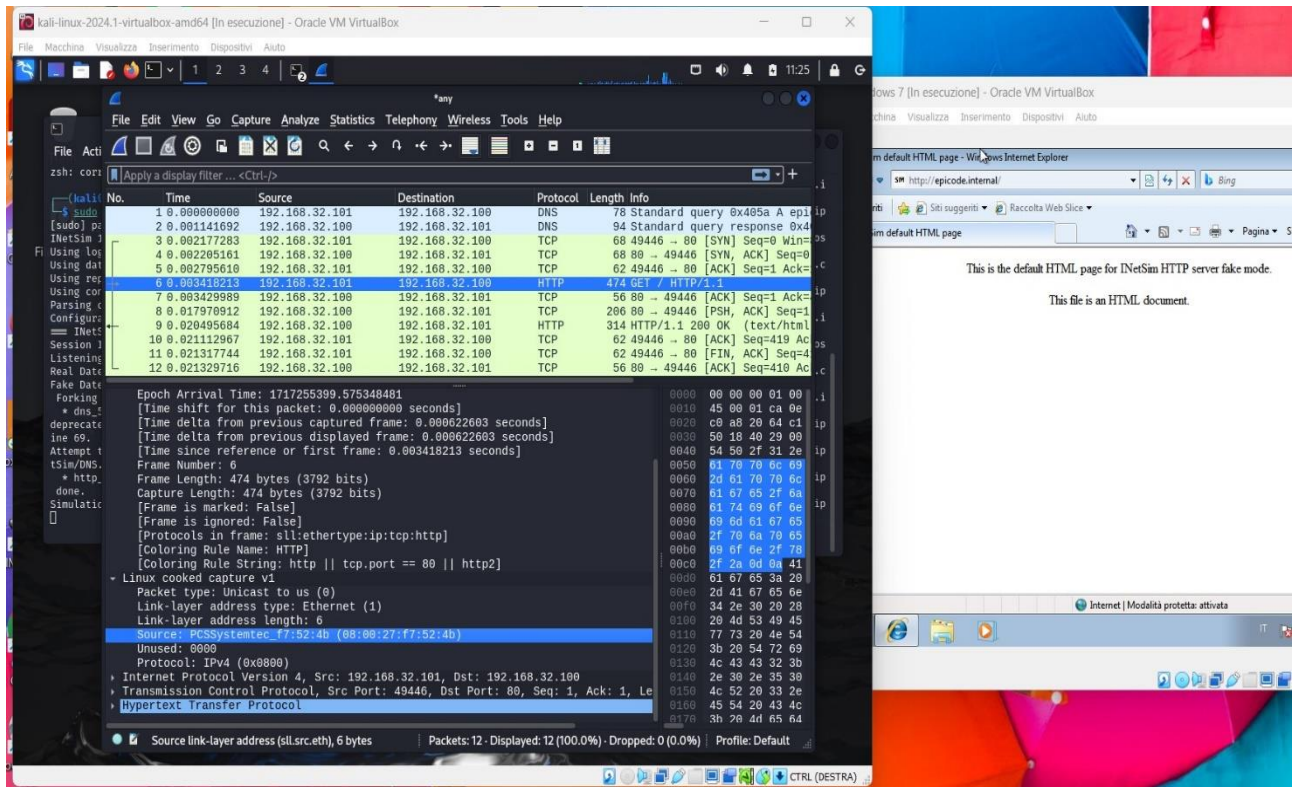


Le differenze tra http e HTTPS sono evidenti già osservando il traffico catturato con Wireshark e verranno spiegate nel dettaglio successivamente.

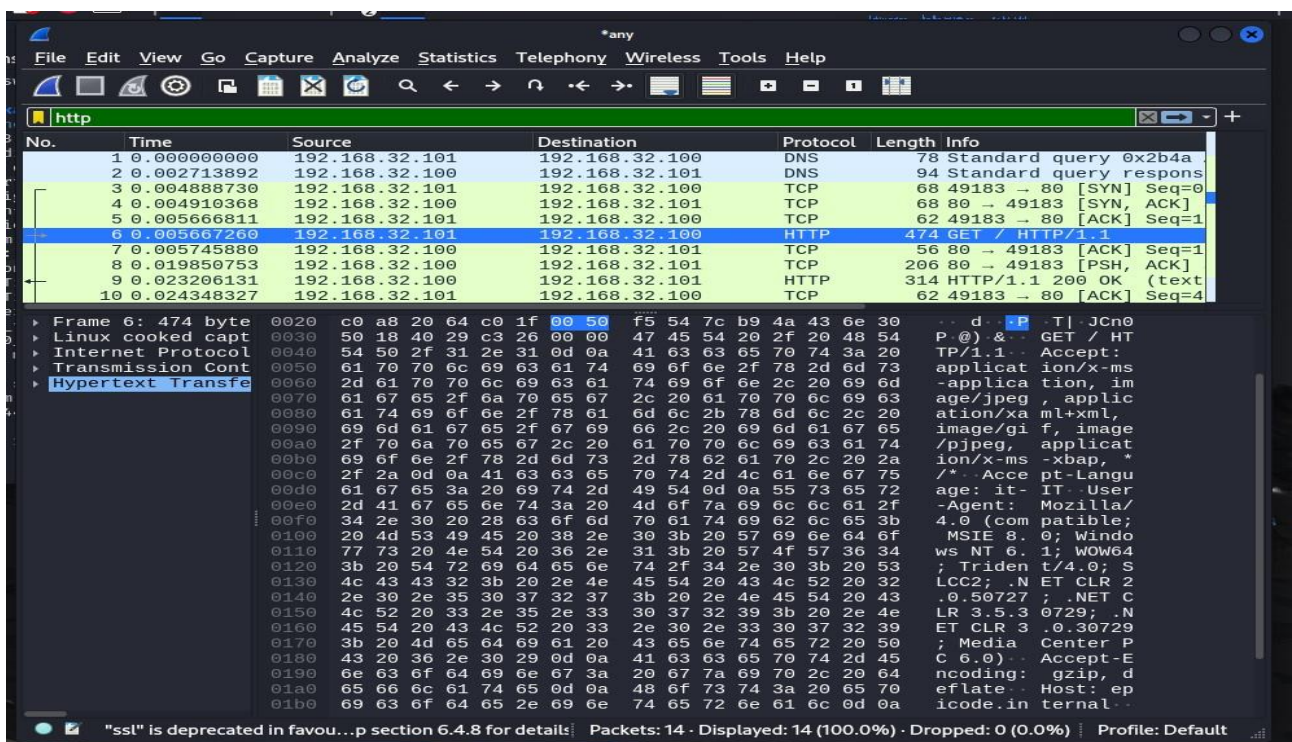


Dopo il three-way-handshake, vi è la request del client Windows al server Kali: nello screenshot si vede il metodo o http verb GET per ottenere una risorsa specifica, l'IP source

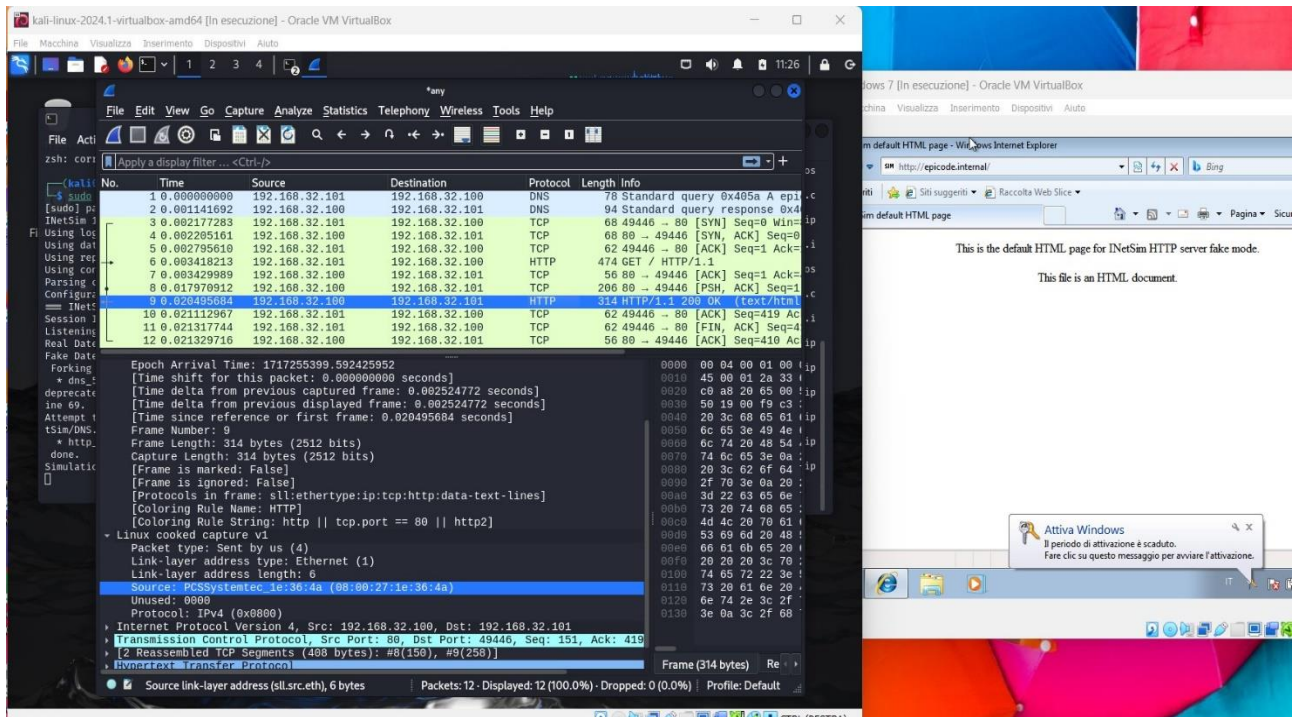
e il MAC source evidenziato sono quelli del client Windows, mentre l'IP destination è quello del server Kali. Si tratta ovviamente di un pacchetto http.



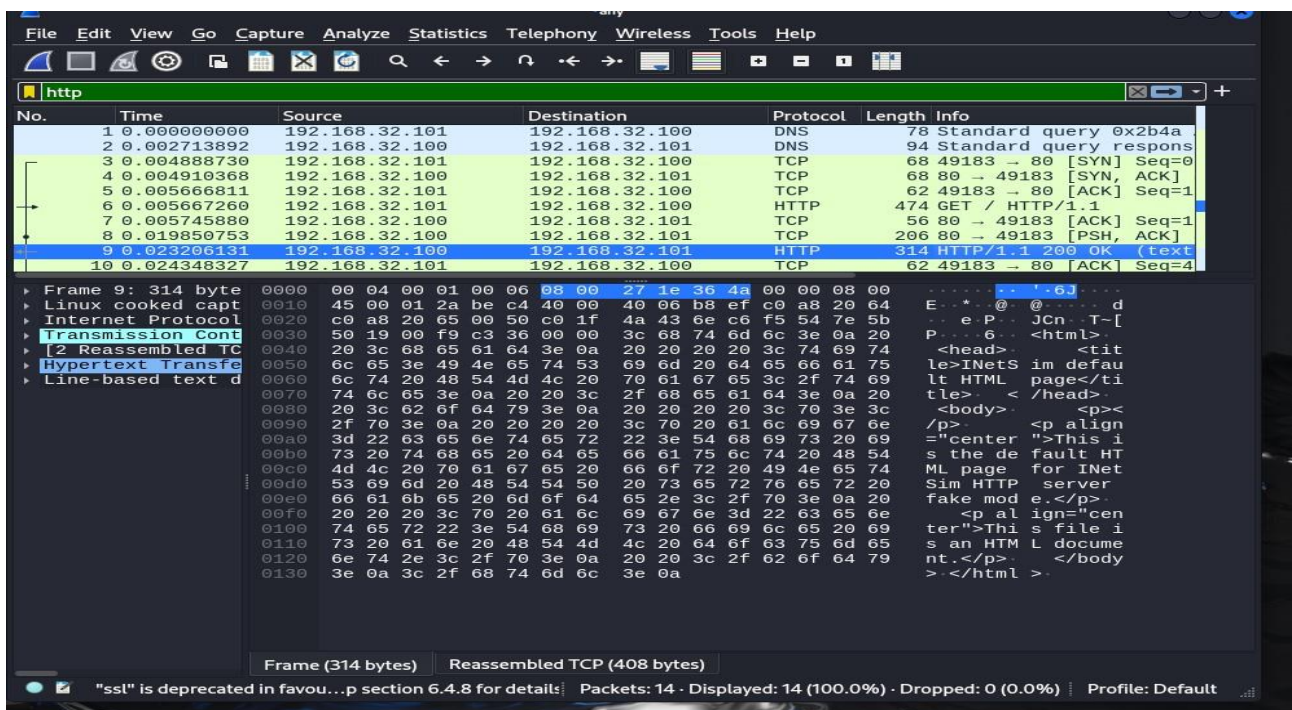
Ho esaminato anche l'header di questa request, osservando la struttura dell'header e toccando con mano quanto studiato durante le lezioni.



Si vede successivamente la riga di stato HTTP/1.1 200 OK, che è la response del server che trova e rende disponibile la risorsa per il client. Si vedono evidenziati l'indirizzo IP di sorgente e l'indirizzo MAC sorgente del server Kali, mentre l'IP e il MAC di destinazione sono quelli del client Windows. Si nota anche la porta 80, ossia la porta standard di HTTP. Anche questo è un pacchetto http.



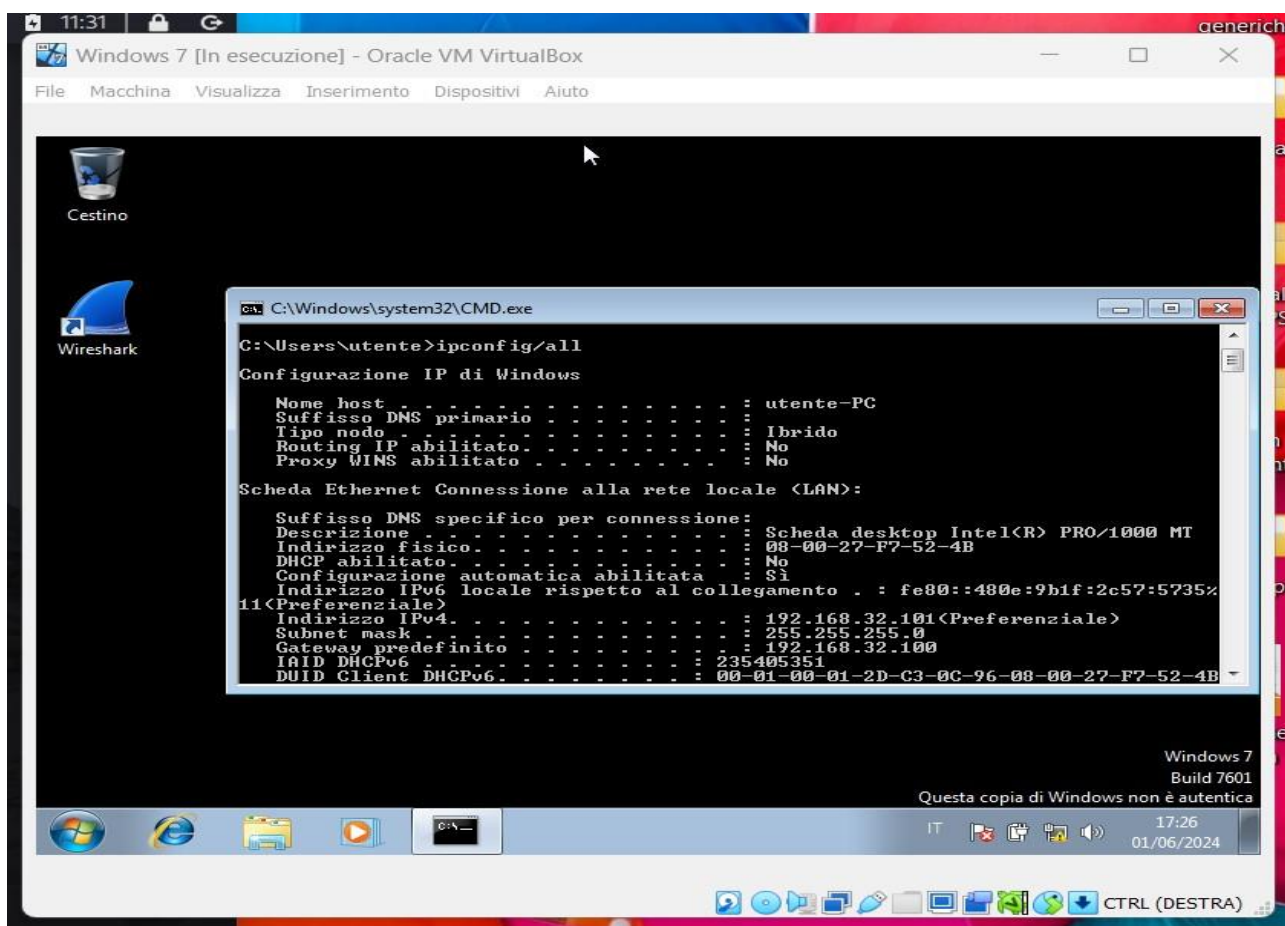
Anche in questo caso ho analizzato il pacchetto della response da parte del server e ho trovato il messaggio inviato dal server al client.



Gli ultimi screenshot permettono di verificare la correttezza degli indirizzi MAC ed IP, mostrando prima quelli del server Kali osservati con ifconfig e ip a e successivamente quelli del client Windows, trovati con ipconfig/all.

```
kali@kali: ~  
File Actions Edit View Help  
zsh: corrupt history file /home/kali/.zsh_history  
(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:fe1e:364a prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:1e:36:4a txqueuelen 1000 (Ethernet)  
    RX packets 14 bytes 6354 (6.2 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 23 bytes 7840 (7.6 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 4 bytes 240 (240.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 4 bytes 240 (240.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
(kali@kali)-[~]  
$
```

```
kali@kali: ~  
File Actions Edit View Help  
zsh: corrupt history file /home/kali/.zsh_history  
(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 group default qlen 1000  
    link/ether 08:00:27:1e:36:4a 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:fe1e:364a/64 scope link proto kernel_ll  
        valid_lft forever preferred_lft forever  
  
(kali@kali)-[~]  
$
```

A questo punto è utile analizzare ancora più dettagliatamente le differenze tra il protocollo HTTP e il protocollo HTTPS, che già sono evidenti osservando i pacchetti catturati con Wireshark. HTTP hypertext transfer protocol, è uno dei protocolli principali per la trasmissione sul web di informazioni ed è spesso usato nelle architetture client server come quella dell'esercizio, ma presenta delle problematiche in termini di sicurezza perché il traffico è in chiaro e quindi a rischio di essere intercettato da utenti non autorizzati per scopi malevoli. La cattura di Wireshark dimostra come tutti i pacchetti HTTP siano sempre in chiaro, anche perché si possono leggere i vari messaggi una volta aperto un pacchetto per analizzarlo. Invece HTTPS, ovvero HTTP Secure, è più sicuro, perché consiste nella comunicazione tramite HTTP all'interno di una connessione criptata grazie al protocollo TLS, transport layer security, e al protocollo utilizzato precedentemente SSL, secure sockets layer. HTTPS garantisce l'integrità dei dati, protegge da attacchi man in the middle e fornisce una cifratura bidirezionale delle comunicazioni tra client e server. Il canale di comunicazione creato è sicuro e i web browser verificano i certificati SSL/TLS dei siti ai quali il client chiede di accedere, e nel caso di un certificato non valido (come è avvenuto svolgendo l'esercizio) informano l'utente dei potenziali rischi ai quali va incontro visitando il sito. In pratica, il protocollo TLS (o in passato SSL) cripta il messaggio http prima della trasmissione e lo decripta una volta a destinazione utilizzando la cifratura a chiave asimmetrica e crea un canale di comunicazione criptato tra client e server grazie allo scambio dei certificati, importante anche per associare la chiave pubblica all'identità del server. Effettivamente, andando a ricercare i messaggi dei pacchetti catturati con Wireshark si nota come i messaggi

dei pacchetti HTTP siano sempre leggibili in chiaro, mentre i messaggi dei pacchetti TLS sono crittografati e non leggibili. Il protocollo HTTPS fornisce una maggiore sicurezza rispetto ad HTTP ma presenta comunque delle vulnerabilità, come nel caso dell'attacco man in the middle SSL Stripping, che cambia il link HTTPS in un link HTTP, contro il quale è stato elaborato HSTS, ossia http strict transport security. Riepilogando quindi le differenze tra HTTP ed HTTPS, i dati scambiati con http sono sempre in chiaro e quindi a rischio, mentre HTTPS implementa la crittografia, per cui dopo il three-way handshake vengono scambiati dei pacchetti TLS che consentono di proteggere la comunicazione con la crittografia. È quindi un modo per eseguire http su un protocollo cifrato come SSL o TLS. Inoltre sono diverse anche le porte utilizzate, perché la porta di http è la porta 80, mentre la porta di HTTPS è 443.