Simulazione Rete Complessa

Obiettivo della simulazione: evidenziare le differenze di traffico tra un client ed un server impostato inizialmente come HTTP e successivamente come HTTPS.

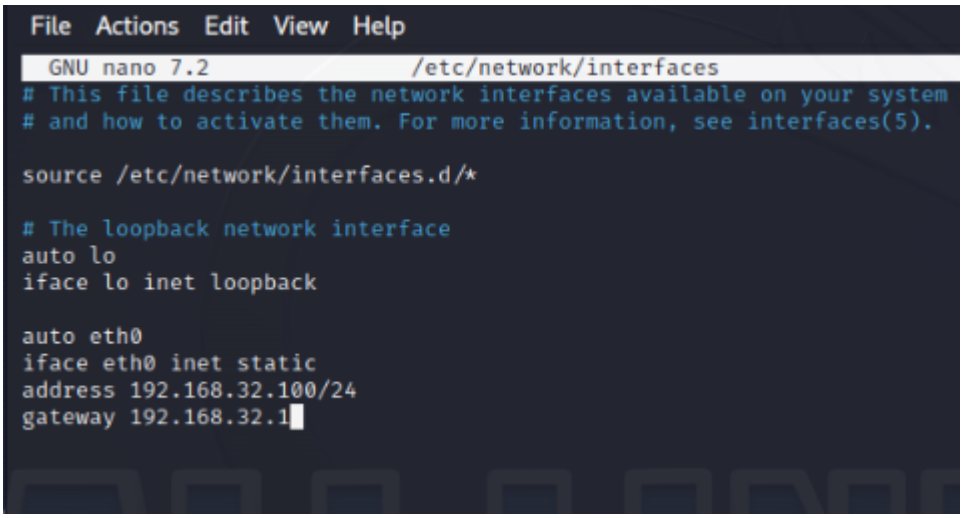
Strumenti utilizzati:

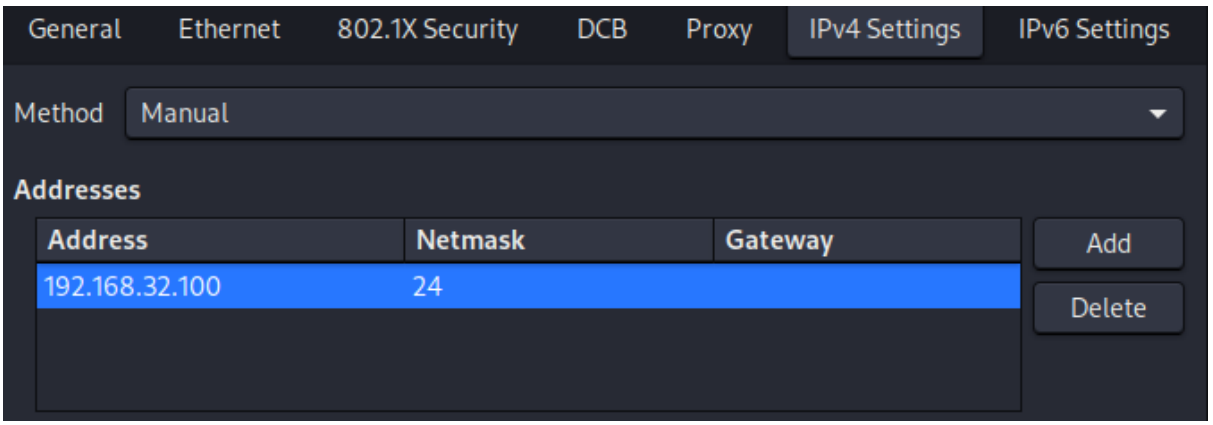
- Kali Linux : sul quale verrà simulato un server DNS ed un servizio HTTP/HTTPS con l'utilizzo del software INetSim;
- Windows 7 : che sarà utilizzato come client per inviare una richiesta tramite web browser;
- Wireshark : programma pre-installato su Kali Linux per analizzare i pacchetti che verranno inviati tra client e server.

1) Settare gli indirizzi IP di Kali Linux e Windows 7

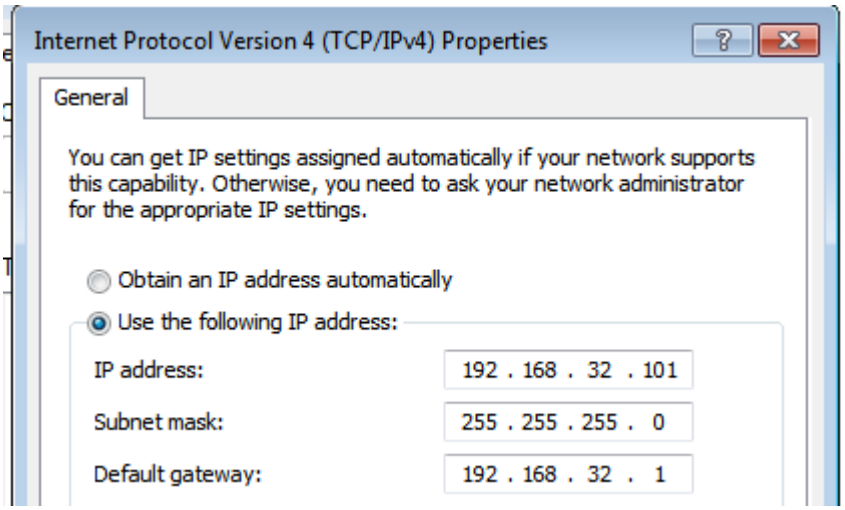
Per iniziare la simulazione ho settato gli indirizzi IP delle due macchine (entrambe virtuali) come mi era stato indicato dalla traccia dell'esercizio.

Kali Linux:





Windows:



2) Configurazione DNS e HTTP

2-a) Per settare un Server DNS e HTTP ho utilizzato il software INetSim già integrato in Kali Linux. Per prima cosa sono andato a modificare testualmente con il comando

sudo nano /etc/inetsim/inetsim.conf

i servizi che il programma offre abilitando solo quelli di DNS e HTTP "commentando" con **#** tutti quelli non utili alla simulazione (ovvero non verranno eseguiti sulla shell) modificando anche l'indirizzo IP a cui essi fanno riferimento (service_bind_address):

2-b) Infine ho modificato il Dominio e l'indirizzo DNS in modo che a all'indirizzo IP configurato corrispondesse il Dominio **epicode.internal**:

```
##########################################
# dns_default_ip
#
# Default IP address to return with DNS replies
#
# Syntax: dns_default_ip <IP address>
#
# Default: 127.0.0.1
#
dns_default_ip 192.168.32.100

##########################################
# dns_default_hostname
#
# Default hostname to return with DNS replies
#
# Syntax: dns_default_hostname <hostname>
#
# Default: www
#
dns_default_hostname epicode


##########################################
# dns_default_domainname
#
# Default domain name to return with DNS replies
#
# Syntax: dns_default_domainname <domain name>
#
# Default: inetsim.org
#
dns_default_domainname epicode.internal
```

3) Avvio servizio con INetSim e prova con browser su Windows

Una volta finita la configurazione ho avviato INetSim tramite riga di comando

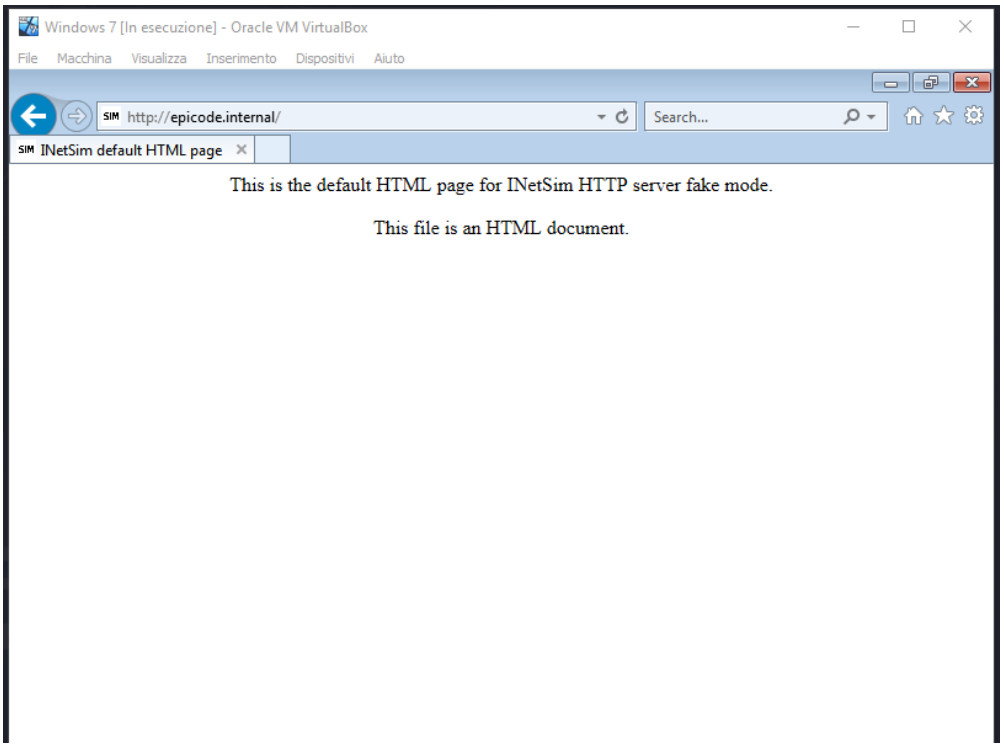
sudo inetsim

facendo cosi partire i due servizi che avevo abilitato

```
┌──(kali㉿kali)-[~]
└─$ sudo inetsim
[sudo] password for kali:
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenbe
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 1423) ===
Session ID:     1423
Listening on:   192.168.32.100
Real Date/Time: 2023-05-06 04:12:25
Fake Date/Time: 2023-05-06 04:12:25 (Delta: 0 seconds)
 Forking services...
  * dns_53_tcp_udp - started (PID 1425)
  * http_80_tcp - started (PID 1426)
 done.
Simulation running.
```

Possiamo vedere che **dns_53_tcp_udp** e **http_80_tcp** sono avviati senza alcun problema e l'indirizzo IP a cui fa riferimento il programma è 192.168.32.100.

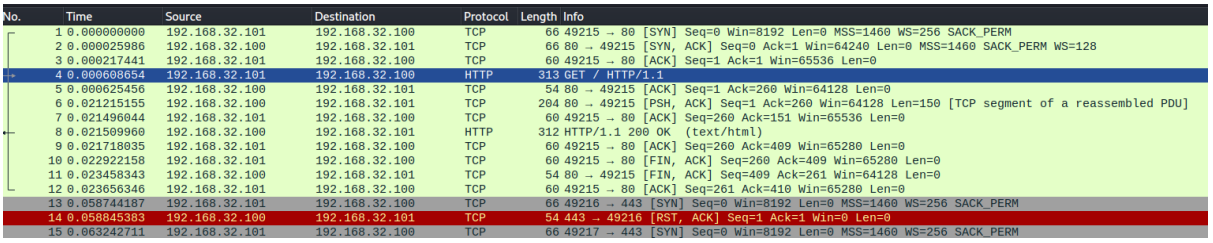
Adesso da windows bisogna provare se il dominio è in funzione utilizzando Internet Explorer:



Dal browser inserendo “epicode.internal” ci indirizza alla pagina di default di INetSim indicando che il server è effettivamente in funzione.

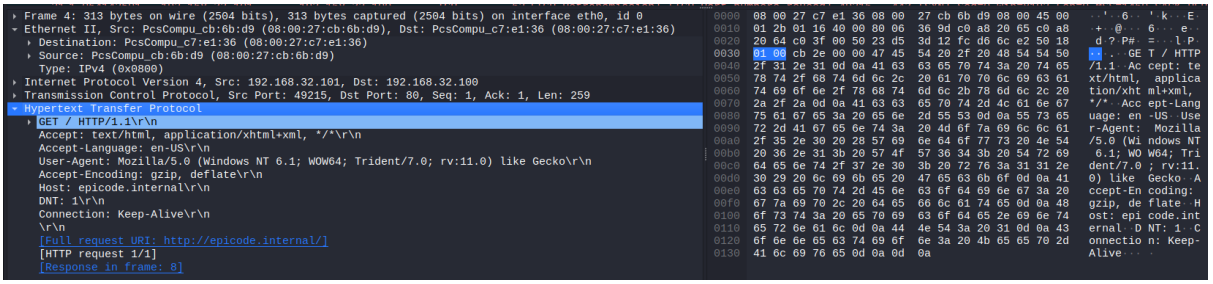
4) Utilizzo di Wireshark

La traccia dell’esercizio ci richiede di intercettare la comunicazione tra Client e Server con il programma Wireshark ed analizzare i pacchetti trasmessi sia tramite Server HTTP che HTTPS. Ho inviato una richiesta al Server abilitando inizialmente soltanto il protocollo HTTP



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.32.101	192.168.32.100	TCP	66	49215 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.000025986	192.168.32.100	192.168.32.101	TCP	66	80 → 49215 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
3	0.000217441	192.168.32.101	192.168.32.100	TCP	60	49215 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.000608654	192.168.32.101	192.168.32.100	HTTP	313	GET / HTTP/1.1
5	0.000625456	192.168.32.100	192.168.32.101	TCP	54	80 → 49215 [ACK] Seq=1 Ack=260 Win=64128 Len=0
6	0.021215155	192.168.32.100	192.168.32.101	TCP	204	80 → 49215 [PSH, ACK] Seq=1 Ack=260 Win=64128 Len=150 [TCP segment of a reassembled PDU]
7	0.021496044	192.168.32.101	192.168.32.100	TCP	60	49215 → 80 [ACK] Seq=260 Ack=151 Win=65536 Len=0
8	0.021509960	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
9	0.021718035	192.168.32.101	192.168.32.100	TCP	60	49215 → 80 [ACK] Seq=260 Ack=409 Win=65280 Len=0
10	0.022922158	192.168.32.101	192.168.32.100	TCP	60	49215 → 80 [FIN, ACK] Seq=260 Ack=409 Win=65280 Len=0
11	0.023458343	192.168.32.100	192.168.32.101	TCP	54	80 → 49215 [FIN, ACK] Seq=409 Ack=261 Win=64128 Len=0
12	0.023656346	192.168.32.101	192.168.32.100	TCP	60	49215 → 80 [ACK] Seq=261 Ack=410 Win=65280 Len=0
13	0.058744187	192.168.32.101	192.168.32.100	TCP	66	49216 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
14	0.058845383	192.168.32.100	192.168.32.101	TCP	54	443 → 49216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.063242711	192.168.32.101	192.168.32.100	TCP	66	49217 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM

avviando wireshark e catturando i pacchetti generati dalla richiesta.



La traccia chiede inoltre di indicare i MAC address delle macchine che sono rispettivamente:

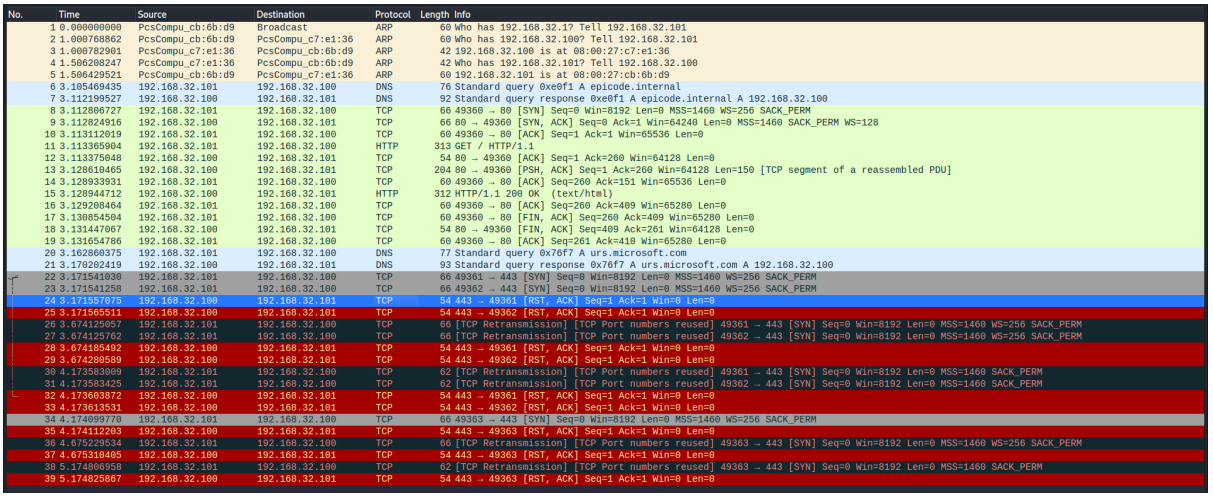
- Destination (08 : 00 : 27 : c7 : e1 : 36);
- Source (08 : 00 : 27 : cb : 6b : d9).

Entrambi sotto la voce Ethernet II.

5) Differenze tra HTTP e HTTPS

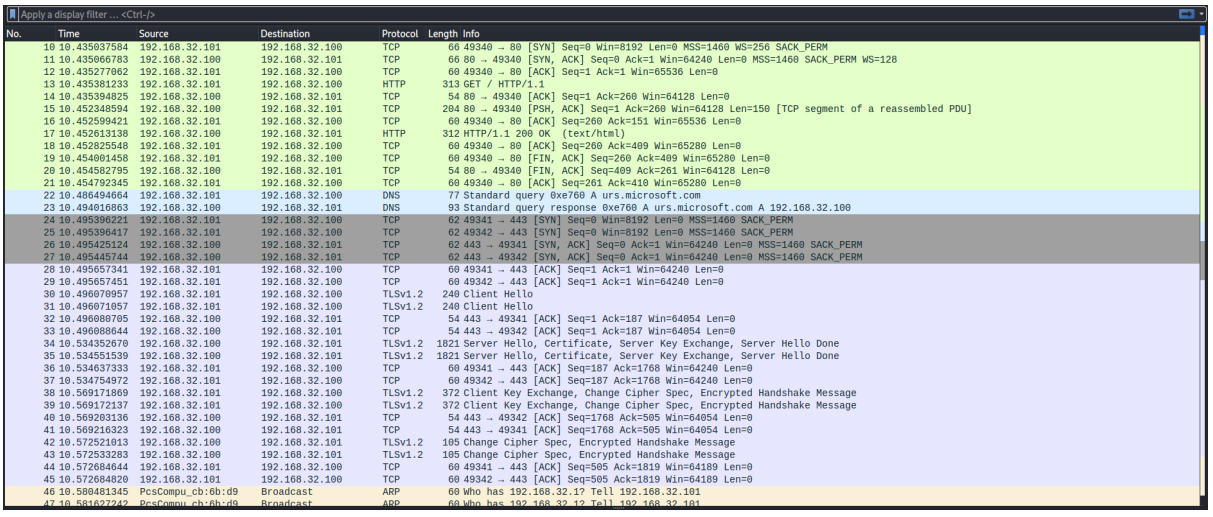
Una volta catturati i pacchetti con richiesta HTTP ho modificato i servizi di INetSim in maniera tale da abilitare anche HTTPS catturandone i protocolli con wireshark nella stessa maniera descritta precedentemente.

HTTP:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_cb:6b:d9	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101
2	1.000768862	PcsCompu_cb:6b:d9	PcsCompu_c7:e1:36	ARP	60	Who has 192.168.32.100? Tell 192.168.32.101
3	1.000782901	PcsCompu_c7:e1:36	PcsCompu_cb:6b:d9	ARP	42	192.168.32.100 is at 08:00:27:c7:e1:36
4	1.506208247	PcsCompu_c7:e1:36	PcsCompu_cb:6b:d9	ARP	42	Who has 192.168.32.101? Tell 192.168.32.100
5	1.506429521	PcsCompu_cb:6b:d9	PcsCompu_c7:e1:36	ARP	60	192.168.32.101 is at 08:00:27:cb:6b:d9
6	3.105469435	192.168.32.101	192.168.32.100	DNS	76	Standard query 0xe0f1 A epicode.internal
7	3.112199527	192.168.32.100	192.168.32.101	DNS	92	Standard query response 0xe0f1 A epicode.internal A 192.168.32.100
8	3.112806727	192.168.32.101	192.168.32.100	TCP	66	49360 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
9	3.112824916	192.168.32.100	192.168.32.101	TCP	66	80 → 49360 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
10	3.113112019	192.168.32.101	192.168.32.100	TCP	60	49360 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
11	3.113365904	192.168.32.101	192.168.32.100	HTTP	313	GET / HTTP/1.1
12	3.113375048	192.168.32.100	192.168.32.101	TCP	54	80 → 49360 [ACK] Seq=1 Ack=260 Win=64128 Len=0
13	3.128610465	192.168.32.100	192.168.32.101	TCP	204	80 → 49360 [PSH, ACK] Seq=1 Ack=260 Win=64128 Len=150 [TCP segment of a reassembled PDU]
14	3.128933931	192.168.32.101	192.168.32.100	TCP	60	49360 → 80 [ACK] Seq=260 Ack=151 Win=65536 Len=0
15	3.128944712	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
16	3.129208464	192.168.32.101	192.168.32.100	TCP	60	49360 → 80 [ACK] Seq=260 Ack=409 Win=65280 Len=0
17	3.130854504	192.168.32.101	192.168.32.100	TCP	60	49360 → 80 [FIN, ACK] Seq=260 Ack=409 Win=65280 Len=0
18	3.131447067	192.168.32.100	192.168.32.101	TCP	54	80 → 49360 [FIN, ACK] Seq=409 Ack=261 Win=64128 Len=0
19	3.131654786	192.168.32.101	192.168.32.100	TCP	60	49360 → 80 [ACK] Seq=261 Ack=410 Win=65280 Len=0
20	3.162860375	192.168.32.101	192.168.32.100	DNS	77	Standard query 0x76f7 A urs.microsoft.com
21	3.170202419	192.168.32.100	192.168.32.101	DNS	93	Standard query response 0x76f7 A urs.microsoft.com A 192.168.32.100
22	3.171541030	192.168.32.101	192.168.32.100	TCP	66	49361 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
23	3.171541258	192.168.32.101	192.168.32.100	TCP	66	49362 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
24	3.171557075	192.168.32.100	192.168.32.101	TCP	54	443 → 49361 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
25	3.171565511	192.168.32.100	192.168.32.101	TCP	54	443 → 49362 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
26	3.674125057	192.168.32.101	192.168.32.100	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 49361 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
27	3.674125762	192.168.32.101	192.168.32.100	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 49362 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
28	3.674185492	192.168.32.100	192.168.32.101	TCP	54	443 → 49361 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
29	3.674280589	192.168.32.100	192.168.32.101	TCP	54	443 → 49362 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
30	4.173583009	192.168.32.101	192.168.32.100	TCP	62	[TCP Retransmission] [TCP Port numbers reused] 49361 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM
31	4.173583425	192.168.32.101	192.168.32.100	TCP	62	[TCP Retransmission] [TCP Port numbers reused] 49362 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM
32	4.173603872	192.168.32.100	192.168.32.101	TCP	54	443 → 49361 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
33	4.173613531	192.168.32.100	192.168.32.101	TCP	54	443 → 49362 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
34	4.174099770	192.168.32.101	192.168.32.100	TCP	66	49363 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
35	4.174112203	192.168.32.101	192.168.32.100	TCP	54	443 → 49363 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36	4.675229534	192.168.32.101	192.168.32.100	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 49363 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
37	4.675310405	192.168.32.100	192.168.32.101	TCP	54	443 → 49363 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38	5.174806958	192.168.32.101	192.168.32.100	TCP	62	[TCP Retransmission] [TCP Port numbers reused] 49363 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM
39	5.174825867	192.168.32.100	192.168.32.101	TCP	54	443 → 49363 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

HTTPS:



Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
10	10.435037584	192.168.32.101	192.168.32.100	TCP	66	49340 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
11	10.435066783	192.168.32.100	192.168.32.101	TCP	66	80 → 49340 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
12	10.435277062	192.168.32.101	192.168.32.100	TCP	60	49340 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
13	10.435381233	192.168.32.101	192.168.32.100	HTTP	313	GET / HTTP/1.1
14	10.435394825	192.168.32.100	192.168.32.101	TCP	54	80 → 49340 [ACK] Seq=1 Ack=260 Win=64128 Len=0
15	10.452348594	192.168.32.100	192.168.32.101	TCP	204	80 → 49340 [PSH, ACK] Seq=1 Ack=260 Win=64128 Len=150 [TCP segment of a reassembled PDU]
16	10.452599421	192.168.32.101	192.168.32.100	TCP	60	49340 → 80 [ACK] Seq=260 Ack=151 Win=65536 Len=0
17	10.452613138	192.168.32.100	192.168.32.101	HTTP	312	HTTP/1.1 200 OK (text/html)
18	10.452825548	192.168.32.101	192.168.32.100	TCP	60	49340 → 80 [ACK] Seq=260 Ack=409 Win=65280 Len=0
19	10.454001458	192.168.32.101	192.168.32.100	TCP	60	49340 → 80 [FIN, ACK] Seq=260 Ack=409 Win=65280 Len=0
20	10.454582795	192.168.32.100	192.168.32.101	TCP	54	80 → 49340 [FIN, ACK] Seq=409 Ack=261 Win=64128 Len=0
21	10.454792345	192.168.32.101	192.168.32.100	TCP	60	49340 → 80 [ACK] Seq=261 Ack=410 Win=65280 Len=0
22	10.486494664	192.168.32.101	192.168.32.100	DNS	77	Standard query 0xe760 A urs.microsoft.com
23	10.494016863	192.168.32.100	192.168.32.101	DNS	93	Standard query response 0xe760 A urs.microsoft.com A 192.168.32.100
24	10.495390221	192.168.32.101	192.168.32.100	TCP	62	49341 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM
25	10.495396417	192.168.32.101	192.168.32.100	TCP	62	49342 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM
26	10.495425124	192.168.32.100	192.168.32.101	TCP	62	443 → 49341 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
27	10.495445744	192.168.32.100	192.168.32.101	TCP	62	443 → 49342 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
28	10.495657341	192.168.32.101	192.168.32.100	TCP	60	49341 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
29	10.495657451	192.168.32.101	192.168.32.100	TCP	60	49342 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
30	10.496070957	192.168.32.101	192.168.32.100	TLSv1.2	240	Client Hello
31	10.496071057	192.168.32.101	192.168.32.100	TLSv1.2	240	Client Hello
32	10.496080705	192.168.32.100	192.168.32.101	TCP	54	443 → 49341 [ACK] Seq=1 Ack=187 Win=64054 Len=0
33	10.496088644	192.168.32.100	192.168.32.101	TCP	54	443 → 49342 [ACK] Seq=1 Ack=187 Win=64054 Len=0
34	10.534352670	192.168.32.100	192.168.32.101	TLSv1.2	1821	Server Hello, Certificate, Server Key Exchange, Server Hello Done
35	10.534551539	192.168.32.100	192.168.32.101	TLSv1.2	1821	Server Hello, Certificate, Server Key Exchange, Server Hello Done
36	10.534637333	192.168.32.101	192.168.32.100	TCP	60	49341 → 443 [ACK] Seq=187 Ack=1768 Win=64240 Len=0
37	10.534754972	192.168.32.101	192.168.32.100	TCP	60	49342 → 443 [ACK] Seq=187 Ack=1768 Win=64240 Len=0
38	10.569171869	192.168.32.101	192.168.32.100	TLSv1.2	372	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
39	10.569172137	192.168.32.101	192.168.32.100	TLSv1.2	372	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
40	10.569203136	192.168.32.100	192.168.32.101	TCP	54	443 → 49342 [ACK] Seq=1768 Ack=505 Win=64054 Len=0
41	10.569216323	192.168.32.100	192.168.32.101	TCP	54	443 → 49341 [ACK] Seq=1768 Ack=505 Win=64054 Len=0
42	10.572521813	192.168.32.100	192.168.32.101	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
43	10.572533283	192.168.32.100	192.168.32.101	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
44	10.572684644	192.168.32.101	192.168.32.100	TCP	60	49341 → 443 [ACK] Seq=505 Ack=1819 Win=64189 Len=0
45	10.572684820	192.168.32.101	192.168.32.100	TCP	60	49342 → 443 [ACK] Seq=505 Ack=1819 Win=64189 Len=0
46	10.580481345	PcsCompu_cb:6b:d9	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101
47	10.581627242	PcsCompu_cb:6b:d9	Broadcast	ARP	60	Who has 192.168.32.1? Tell 192.168.32.101

Conclusioni

La differenze sono abbastanza evidenti nelle due catture perchè a parte la richiesta al server (in verde) che rimane sostanzialmente invariata possiamo notare che nei pacchetti HTTPS c'è un'intera zona color lilla (28-45) nella quale possiamo leggere Server Key Exchange o Encrypted Handshake Message. Queste indicano uno scambio di "messaggi" tra sorgente e destinatario per comunicare ed iniziare uno scambio di algoritmi e chiavi di cifratura per cosi ottenere una connessione crittografata.

Contrariamente i pacchetti HTTP invece non hanno un sistema di invio dati crittografato che si traduce in una serie di pacchetti di colore rosso con flag RST (reset) che inviano per resettare o addirittura chiudere la connessione stabilita; probabilmente ciò è dato dal fatto che ritengono la connessione non sicura, questo tipo di protocollo è infatti più vulnerabile ad attacchi esterni.