TLS decryption in Wireshark

When I spoke with some people I found out that most of them had some hard time with TLS decryption in world's foremost and widely-used network protocol analyzer “Wireshark”.

TLS decryption could be very useful when we are analyzing some potential malicious web traffic or simply troubleshooting our own web server.

One of the method, I will be mentioning, works for every case covering situation like network traffic decryption of our web browser client “Google Chrome, Firefox, Python…”. The second method covers situations where we have RSA Private key of web server and captured network traffic as “.pcap, .pcapng…” or we are performing live network capturing. The second method, using RSA private key is also applicable for TLS decryption of different protocol than HTTP.

Both of these methods have their advantages and disadvantages.

This little post is about TLS decryption possibilities in Wireshark and I hope it will serve someone as a simple guide.

Let´s start ☺

As mentioned before, Wireshark supports TLS decryption when appropriate secrets are provided. The two available methods are:

- Key log file using per-session secrets (Using the (Pre)-Master-Secret)
- Decryption using RSA private key

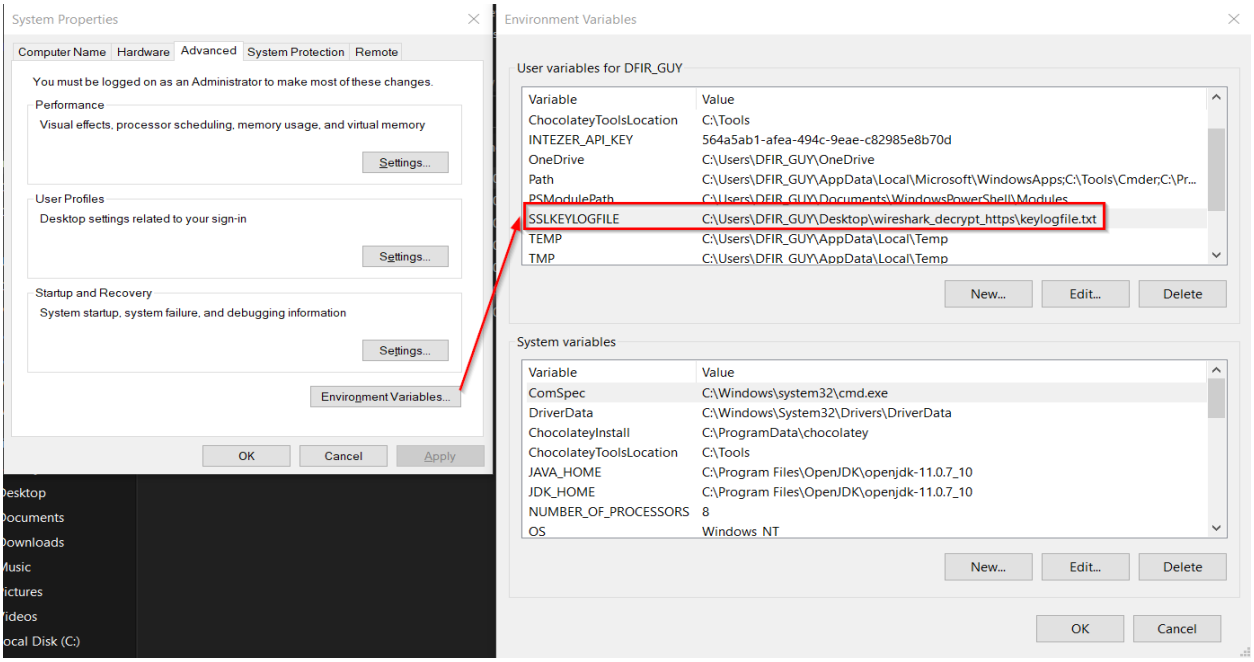
Using the (Pre)-Master-Secret

This method used for TLS decryption is using pre-master key logging. It is very simple and quick to set up but works only with some web browser clients (example: Google Chrome, Firefox, Python).

It works thanks to setting up the environment variable “SSLKEYLOGFILE”. The value of this environment variable is path, pointing to some writable location where our key logging file will be created. Mentioned web browser clients check this environment variable and if it is set, they are able to log there per-session secrets which are immediately used in Wireshark.

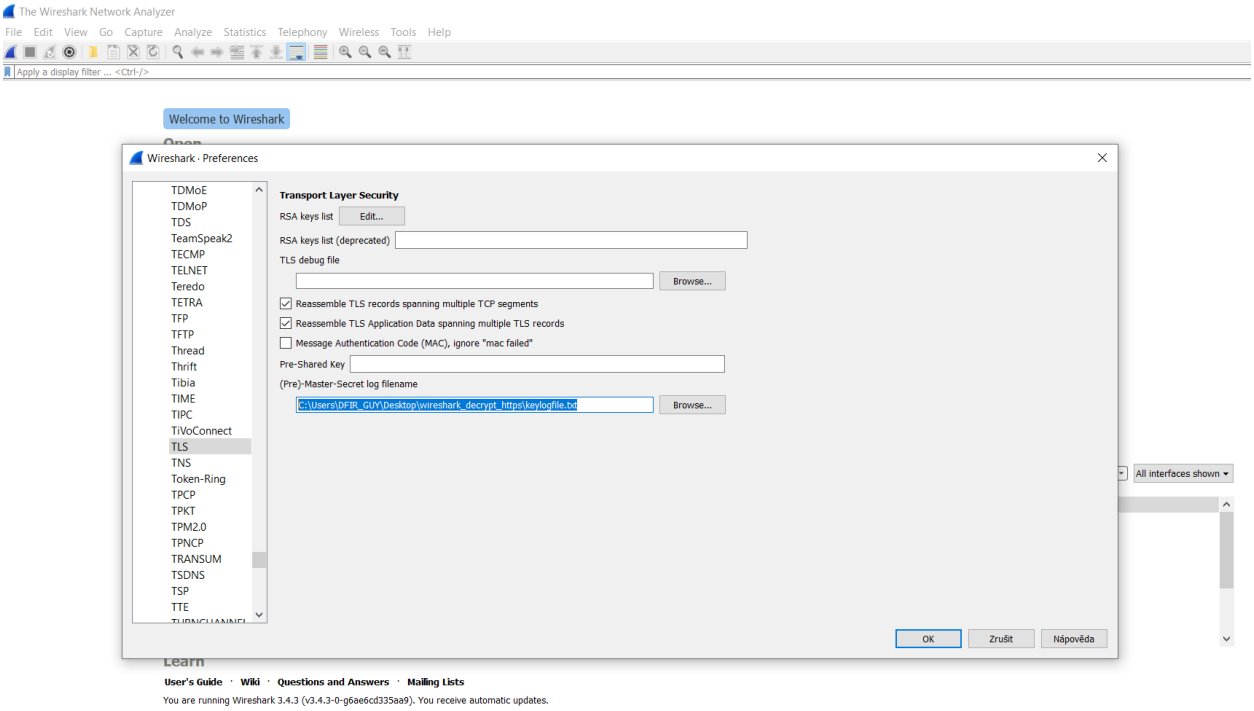
Advantage of this method is that it works in every case (SSL3. TLS1.0 -1.3), also when TLS cipher suite selected by the server is using (EC)DHE for key exchange algorithm.

We start with setting environment variable “SSLKEYLOGFILE”. As we can see in the picture below: (“SSLKEYLOGFILE = C:\Users\DFIR_GUY\Desktop\wireshark_decrypt_https\keylogfile.txt”)



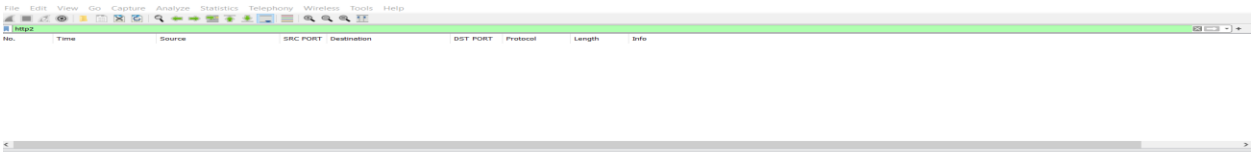
After that, we have to set this path to Key log file containing our per-session secrets in Wireshark.

Start Wireshark and go to -> Edit -> Preferences -> Protocols -> TLS -> Pre Master-Secret log filename. Set the appropriate path as we set before in environment variable.

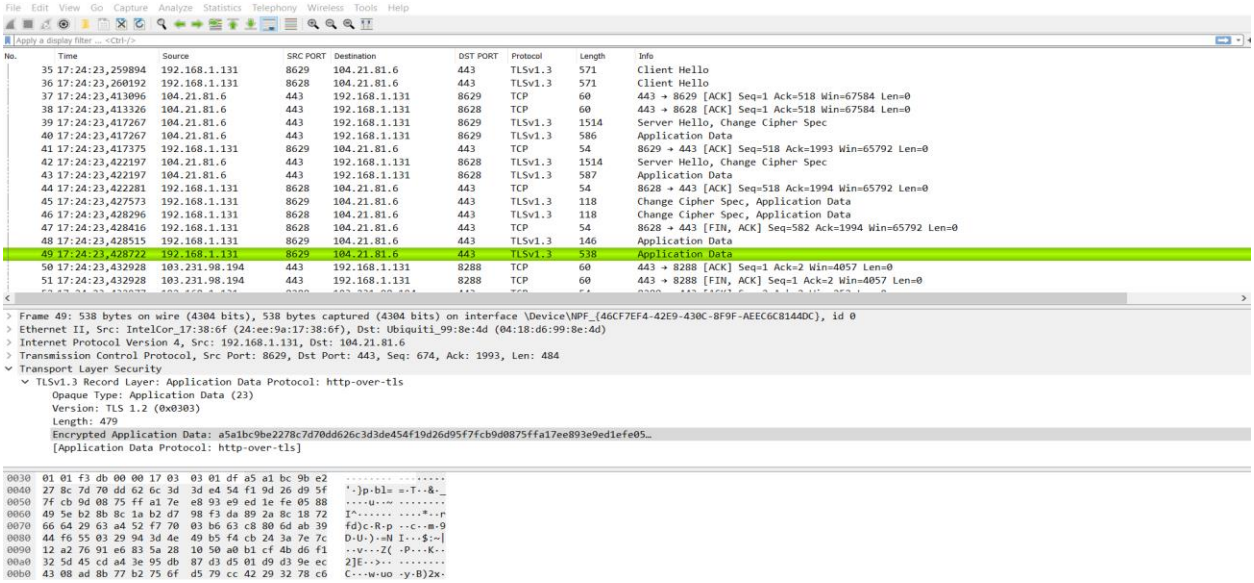


Now we are ready. Below is a network traffic of Google Chrome before and after setting the Key log file containing our per-session secrets:

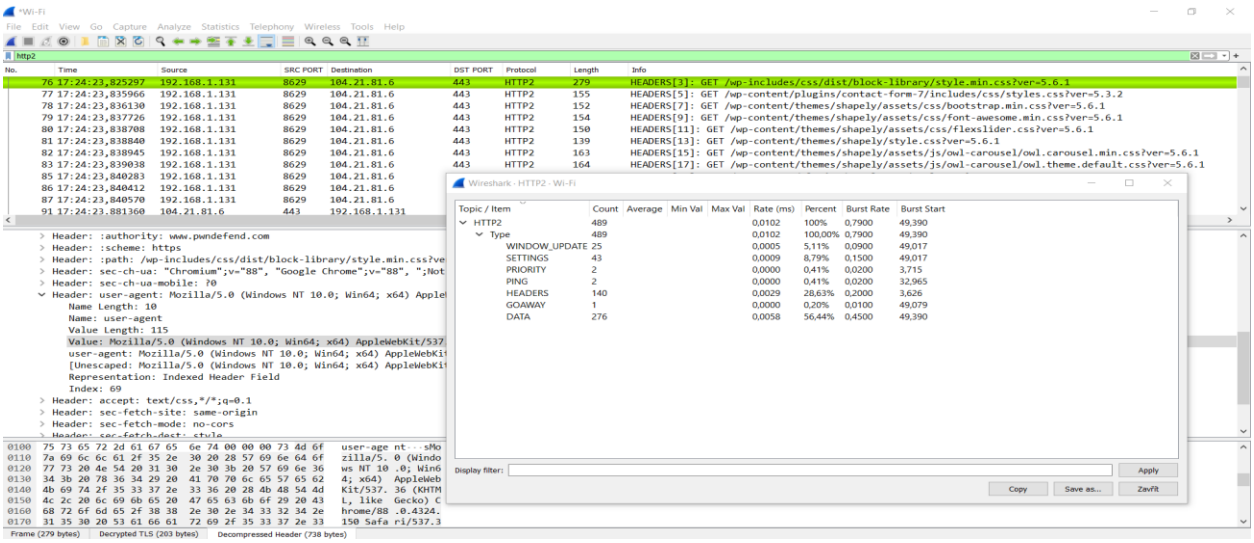
Before (No HTTP traffic available):



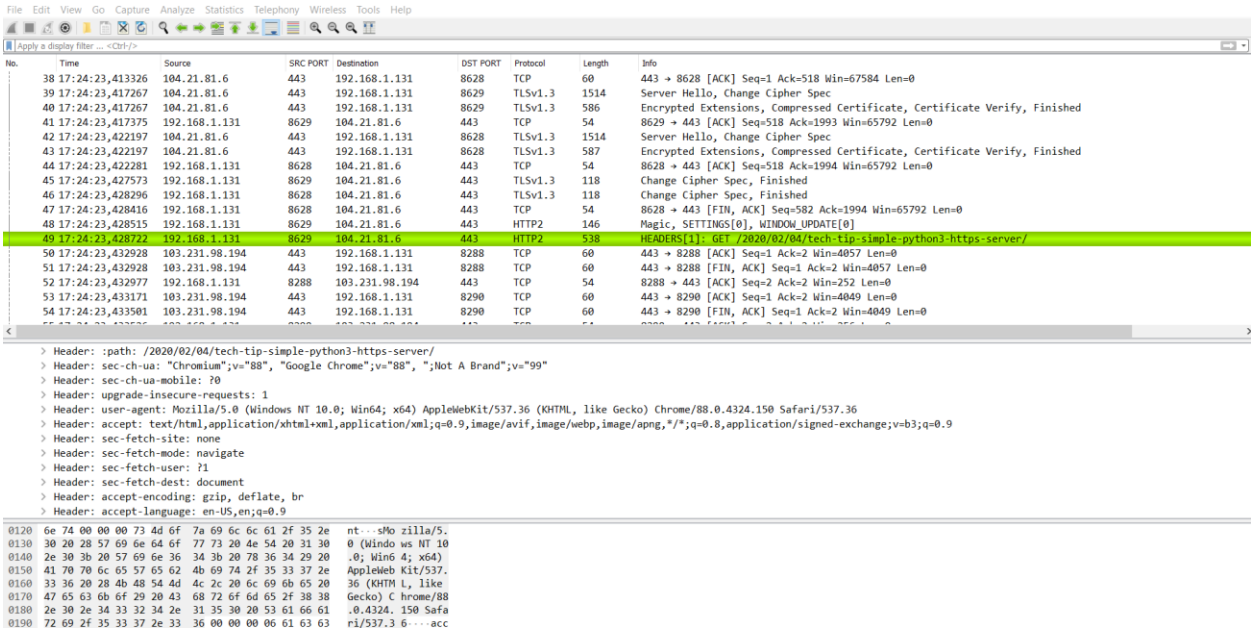
49. packet after TLS Handshake – Only TLS encrypted application data.



After:



49. packet after TLS Handshake – TLS decrypted application data HTTP2.



Using RSA private key

This method works in case where we own or get the RSA private key. It is applicable also for TLS decryption of different protocol than HTTP.

One of the limitations of using this method is that only SSL3, TLS1.0 -1.2 must be used. There is no support for TLS version 1.3. Another limitation is that (EC)DHE key exchange algorithm specified in the cipher suites must not be used because of principle how (EC)DHE key exchange algorithms work.

According to Wireshark docs only in these cases RSA private key method could be used:

- The cipher suite selected by the server is not using (EC)DHE.
- The protocol version is SSLv3, (D)TLS 1.0-1.2. It does not work with TLS 1.3.
- The private key matches the server certificate. It does not work with the client certificate, nor the Certificate Authority (CA) certificate.
- The session has not been resumed. The handshake must include the ClientKeyExchange handshake message.

So if we own the web server RSA private key and can manage to use supported version of TLS + not using cipher suites implementing (EC)DHE key exchange algorithm, we can successfully decrypt the TLS traffic.

These restrictions we can configure on the client side (web browser client) for example: to use only TLS 1.2 and cipher suites not using (EC)DHE key exchange algorithm but after that it all depends on server side if it enables to perform TLS handshake using these specified parameters.

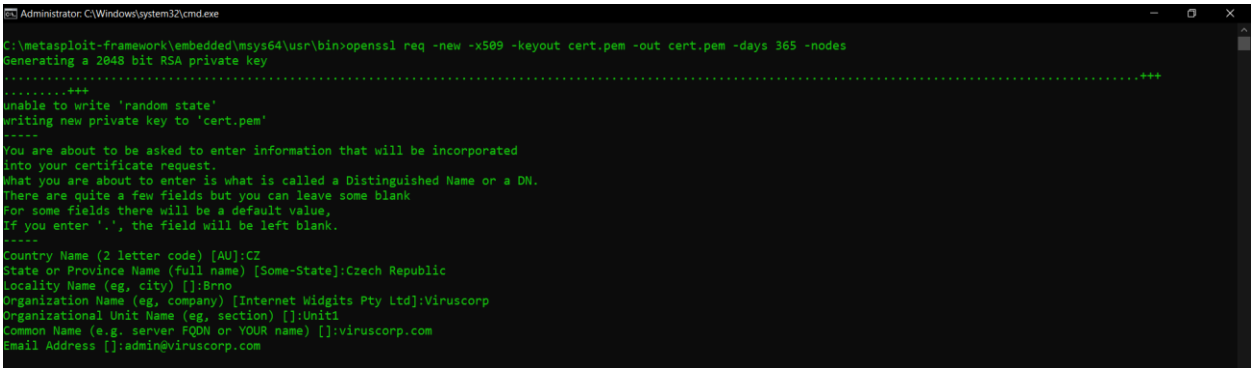
I will be focusing on how we can configure the server side to enable only TLS1.2 and cipher suites not using (EC)DHE key exchange algorithm. For this example, I created simple Python web server with our specified parameters which will be covered later.

First of all, we have to generate our RSA private key + Certificate for our web server. We will use "openssl". We can download it here: [openssl].

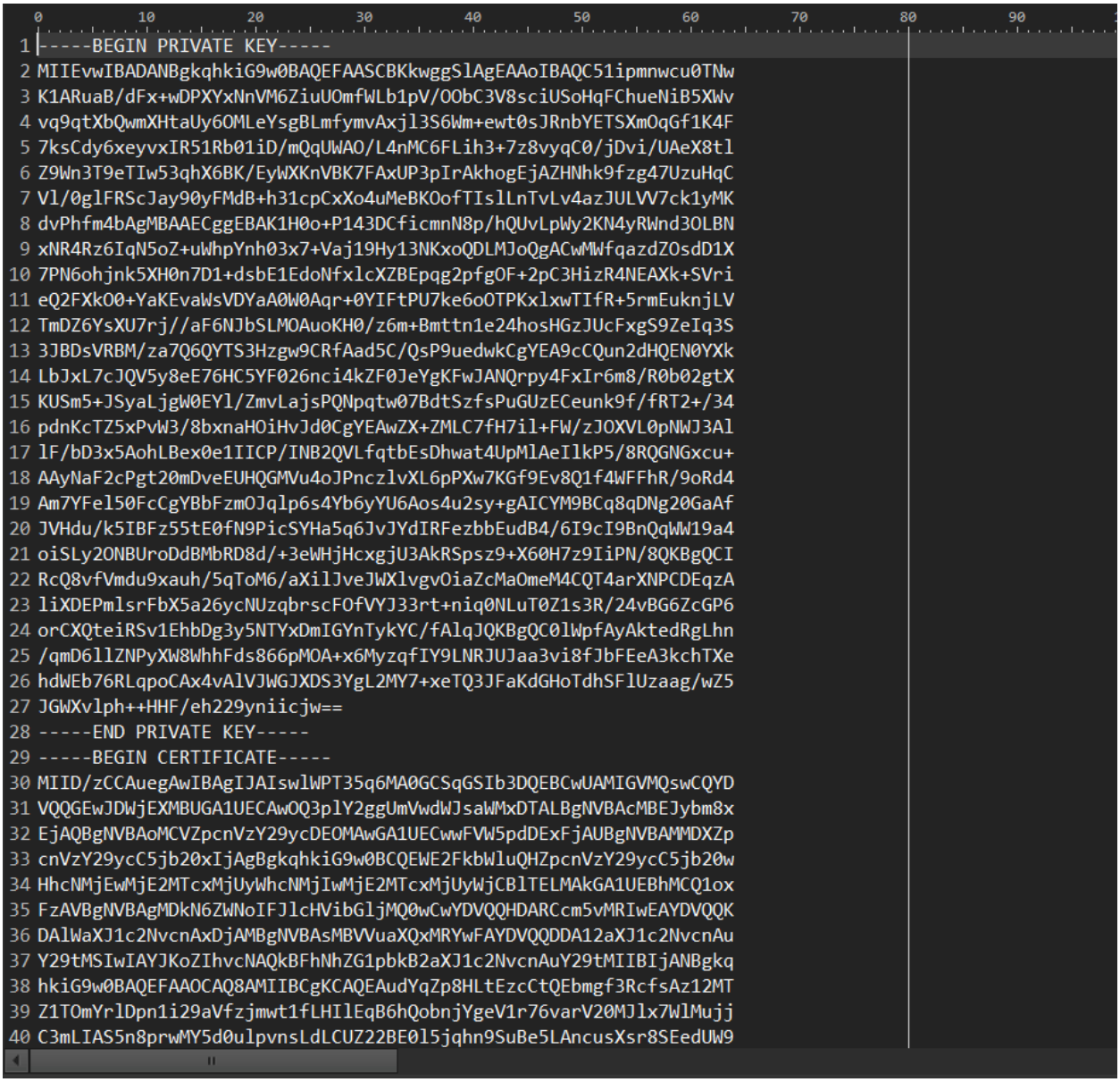
Basically, Certificate contains RSA public key + another data for CERT verification. RSA private key is self-explanatory.

Generating Certificate + RSA private key in openssl: "openssl req -new -x509 -keyout cert.pem -out cert.pem -days 365 –nodes"

Fill the CERT info as you wish.



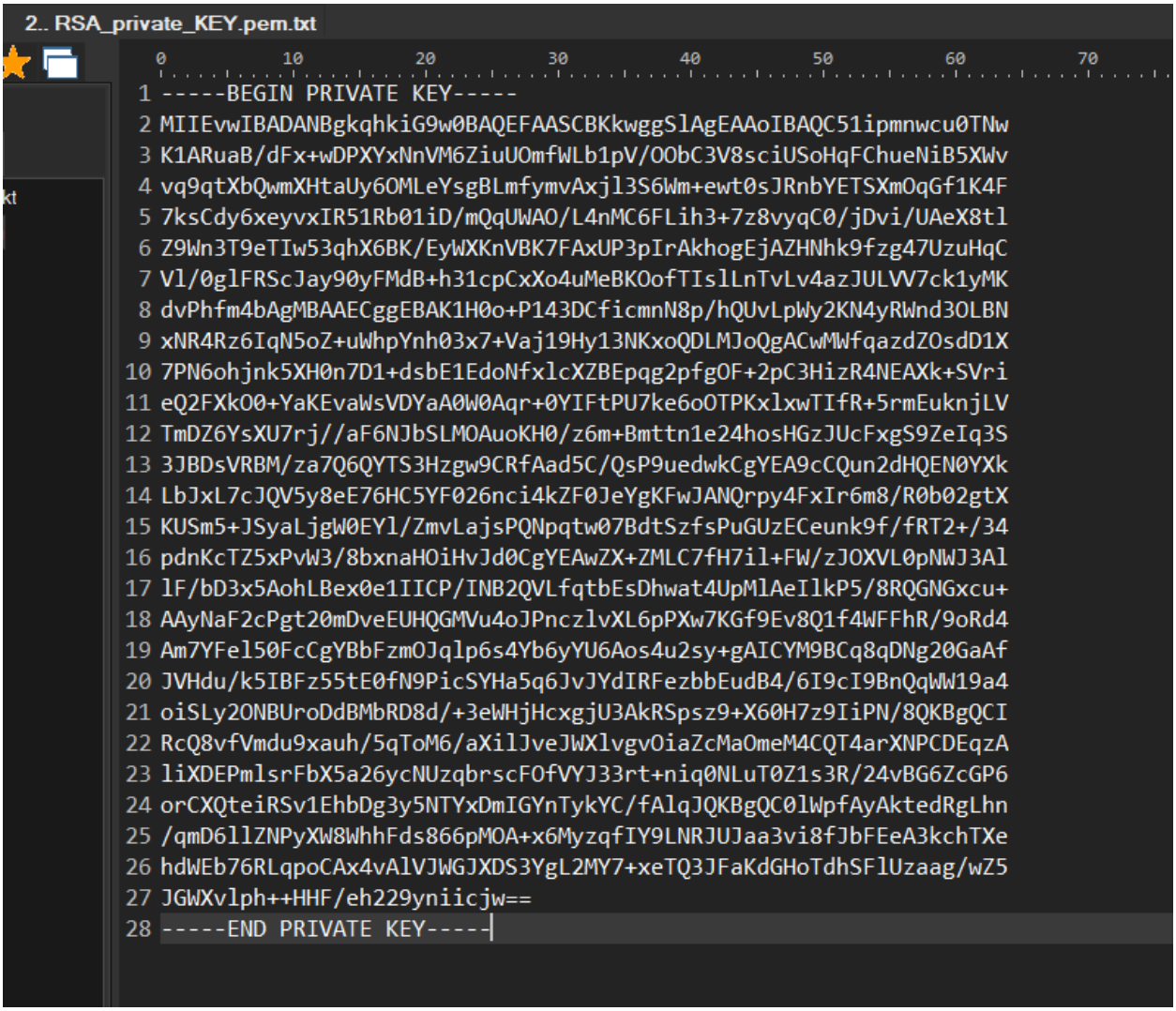
Our newly created cert.pem is containing both certificate and RSA private key as we can see in the picture below.



```
-----BEGIN PRIVATE KEY-----
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQC51ipmnwcu0TNw
K1ARuaB/dFx+wDPXYxNnVM6ZiuUOmfWLb1pV/OObC3V8sciUSoHqFChueNiB5XWv
vq9qtXbQwmXHtaUy6OMLeYsgBLmfymvAxjl3S6Wm+ewt0sJRnbYETSXmOqGf1K4F
7ksCdy6xeyvxIR51Rb01iD/mQqUWAO/L4nMC6FLih3+7z8vyqC0/jDvi/UAeX8tl
Z9Wn3T9eTIw53qhX6BK/EyWXKnVBK7FAxUP3pIrAkhogEjAZHNhk9fzg47UzuHqC
Vl/0glFRScJay90yFMdB+h31cpCxXo4uMeBKOofTIslLnTvLv4azJULVV7ck1yMK
dvPhfm4bAgMBAAECggEBAK1H0o+P143DCficmnN8p/hQUvLpWy2KN4yRWnd3OLBN
xNR4Rz6IqN5oZ+uWhpYnh03x7+Vaj19Hy13NKxoQDLMJoQgACwMWfqazdZOsdD1X
7PN6ohjnk5XH0n7D1+dsbE1EdoNfxlcXZBEpqg2pfgOF+2pC3HizR4NEAXk+SVri
eQ2FXkO0+YaKEvaWsVDYaA0W0Aqr+0YIFtPU7ke6oOTPKxlxwTIfR+5rmEuknjLV
TmDZ6YsXU7rj/aF6NJbSLMOAuoKH0/z6m+Bmttn1e24hosHGzJUcFxgS9ZeIq3S
3JBDsVRBM/za7Q6QYTS3Hzgw9CRfAad5C/QsP9uedwkCgYEA9cCQun2dHQEN0YXk
LbJxL7cJQV5y8eE76HC5YF026nci4kZF0JeYgKFwJANQrpy4FxIr6m8/R0b02gtX
KUSm5+JSyaLjgW0EYl/ZmvLajsPQNpqtw07BdtSzfsPuGUzECeunk9f/fRT2+/34
pdnKcTZ5xPvW3/8bxnaHOiHvJd0CgYEAwZX+ZMLC7fH7il+FW/zJOXVL0pNWJ3Al
lF/bD3x5AohLBex0e1IICP/INB2QVLfqtbEsDhwat4UpMlAeIlkP5/8RQGNGxcu+
AAyNaF2cPgt20mDveEUHQGMVu4oJPnczlvXL6pPXw7KGf9Ev8Q1f4WFFhR/9oRd4
Am7YFel50FcCgYBbFzmOJqlp6s4Yb6yYU6Aos4u2sy+gAICYM9BCq8qDNg20GaAf
JVHdu/k5IBFz55tE0fN9PicSYHa5q6JvJYdIRFezbbEudB4/6I9cI9BnQqWW19a4
oiSLy2ONBUroDdBMbRD8d/+3eWHjHcxgjU3AkRSpsz9+X60H7z9IiPN/8QKBgQCI
RcQ8vfVmdu9xauh/5qToM6/aXilJveJWXlvgvOiaZcMaOmeM4CQT4arXNPCDEqzA
liXDEPmlsrFbX5a26ycNUzqbrscFOfVYJ33rt+niq0NLuT0Z1s3R/24vBG6ZcGP6
orCXQteiRSv1EhbDg3y5NTYxDmIGYnTykYC/fAlqJQKBgQC0lWpfAyAktedRgLhn
/qmD6llZNPyXW8WhhFds866pMOA+x6MyzqfIY9LNRJUJaa3vi8fJbFEeA3kchTXe
hdWEb76RLqpoCAx4vAlVJWGJXDS3YgL2MY7+xeTQ3JFaKdGHoTdhSFlUzaag/wZ5
JGWXvlph++HHF/eh229yniicjw==
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIID/zCCAuegAwIBAgIJAIswlWPT35q6MA0GCSqGSIb3DQEBCwUAMIGVMQswCQYD
VQQGEwJDWjEXMBUGA1UECAwOQ3plY2ggUmVwdWJsaWMxDTALBgNVBAcMBEJybm8x
EjAQBgNVBAoMCVZpcnVzY29ycDEOMAwGA1UECwwFVW5pdDExFjAUBgNVBAMMDXZp
cnVzY29ycC5jb20xIjAgBgkqhkiG9w0BCQEWE2FkbWluQHZpcnVzY29ycC5jb20w
HhcNMjEwMjE2MTcxMjUyWhcNMjIwMjE2MTcxMjUyWjCBlTELMAkGA1UEBhMCQ1ox
FzAVBgNVBAgMDkN6ZWNoIFJlcHVibGljMQ0wCwYDVQQHDARCcm5vMRIwEAYDVQQK
DAlWaXJ1c2NvcnAxDjAMBgNVBAsMBVVuaXQxMRYwFAYDVQQDDA12aXJ1c2NvcnAu
Y29tMSIwIAYJKoZIhvcNAQkBFhNhZG1pbkB2aXJ1c2NvcnAuY29tMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAudYqZp8HLtEzcCtQEbmgf3RcfsAz12MT
Z1TOmYrlDpn1i29aVfzjmwt1fLHIlEqB6hQobnjYgeV1r76varV20MJlx7WlMujj
C3mLIAS5n8prwMY5d0ulpvnsLdLCUZ22BE0l5jqhn9SuBe5LAncusXsr8SEedUW9
```

We will extract the RSA private key from cert.pem which we will be using later in Wireshark. The cert.pem file with both RSA private key and certificate is used in our Python server implementation later on.

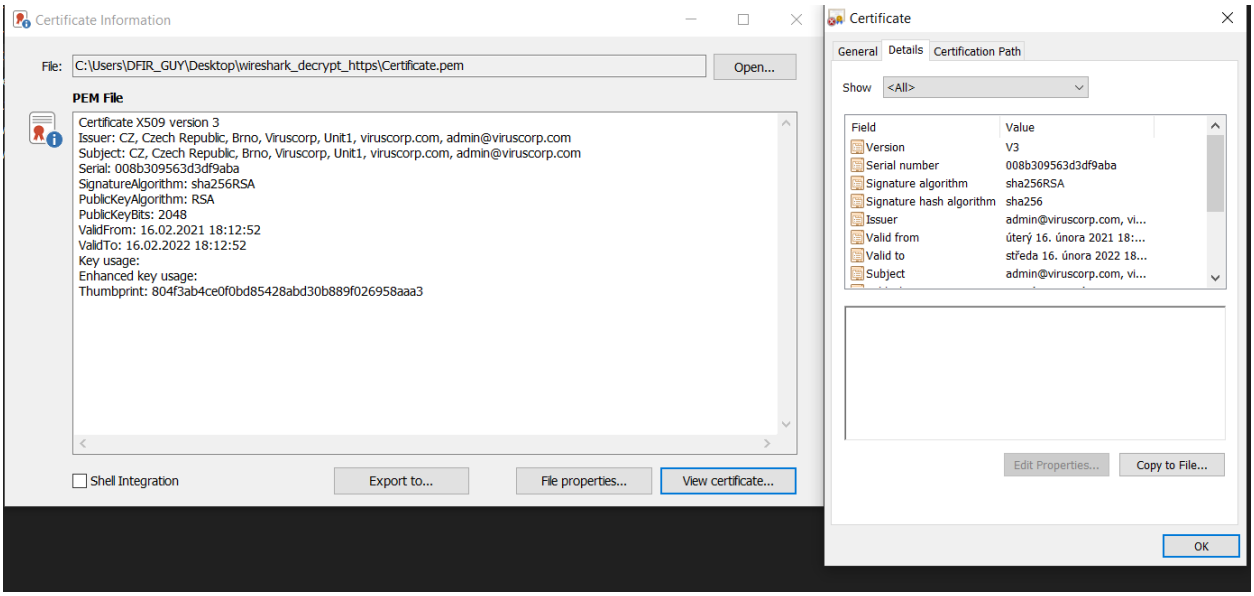
Extracted RSA private key:



We can also extract the certificate from cert.pem, save it as certificate.pem and check containing info we filled during its creation before. We can check these information in tool like [certinfo].

```
-----BEGIN CERTIFICATE-----
MIID/zCCAuegAwIBAgIJAIswlWPT35q6MA0GCSqGSIb3DQEBCwUAMIGVMQswCQYD
VQQGEwJDWjEXMBUGA1UECAwOQ3plY2ggUmVwdWJsaWMxDTALBgNVBAcMBEJybm8x
EjAQBgNVBAoMCVZpcnVzY29ycDEOMAwGA1UECwwFVW5pdDExFjAUBgNVBAMMDXZp
cnVzY29ycC5jb20xIjAgBgkqhkiG9w0BCQEWE2FkbWluQHZpcnVzY29ycC5jb20w
HhcNMjEwMjE2MTcxMjUyWhcNMjIwMjE2MTcxMjUyWjCBlTELMAkGA1UEBhMCQ1ox
FzAVBgNVBAgMDkN6ZWNoIFJlcHVibGljMQ0wCwYDVQQHDARCcm5vMRIwEAYDVQQK
DAlWaXJ1c2NvcnAxDjAMBgNVBAsMBVVuaXQxMRYwFAYDVQQDDA12aXJ1c2NvcnAu
Y29tMSIwIAYJKoZIhvcNAQkBFhNhZG1pbkB2aXJ1c2NvcnAuY29tMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAudYqZp8HLtEzcCtQEbmgf3RcfsAz12MT
Z1TOmYrlDpn1i29aVfzjmwt1fLHIlEqB6hQobnjYgeV1r76varV20MJlx7WlMujj
C3mLIAS5n8prwMY5d0ulpvnsLdLCUZ22BE0l5jqhn9SuBe5LAncusXsr8SEedUW9
NYg/5kKlFgDvy+JzAuhS4od/u8/L8qgtP4w74v1AHl/LZWfVp90/XkyMOd6oV+gS
vxMllyp1QSuxQMVD96SKwJIaIBIwGRzYZPX84OO1M7h6glZf9IJRUUnCWsvdMhTH
Qfod9XKQsV6OLjHgSjqH0yLJS507y7+GsyVC1Ve3JNcjCnbz4X5uGwIDAQABo1Aw
TjAdBgNVHQ4EFgQUQLEJdDk5RFaoX2ct7c/pj4np5vAwHwYDVR0jBBgwFoAUQLEJ
dDk5RFaoX2ct7c/pj4np5vAwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOC
AQEAtF1PjKoz3/J9l0hxUPtrW5FoHOYCmftDX9msTx8YIqCYkitbmNIv0Qujb5Z3
uQ1QyHR6phi4NuTnXOCsKQc8po4CGaVK9tVC9NqnQ01S4OYb2JyvTd8PGty+E2II
7MnnihbUwtv8GqWUPJz0oBfjdtFSWT4grwVol5NgDEZUZdoWnXtCyaZ3RaviMx+c
Eq7S3jFY02/MiId4G3A7Tlc5EtB4h4Ko/kOovUY7M+w6bF6CVetHRrERuz+lOLM7
u2lTTGj1USk82q7KLP3LPyYjevkFBHN6LNXMgB/R/CNFCEJpxFrQHjDwfLK2uq+o
FbYuONDv25jHPFcM0pD/QXAQlg==
-----END CERTIFICATE-----
```

Using Certinfo:

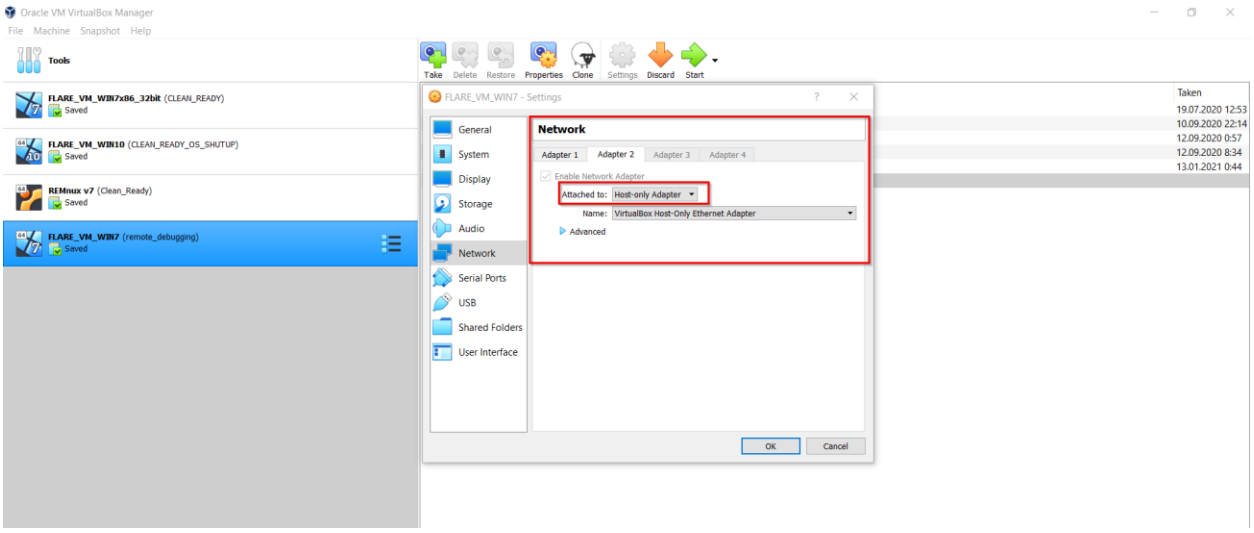


In real scenario/case, we will already have some web server private key and captured network traffic data but here, we will be implementing our own python web server with our previously created certificate + RSA private key.

Let´s start to run some VM where we have installed python3+. For virtualization platform I will be using VirtualBox and as Guest OS – Windows 7.

Let´s configure our VM network setting in VirtualBox:

Choose VM and go to Settings → Network and enable Network adapter (Host-only Adapter) – this settings will create Network interface only between our Host and Guest VM machine.

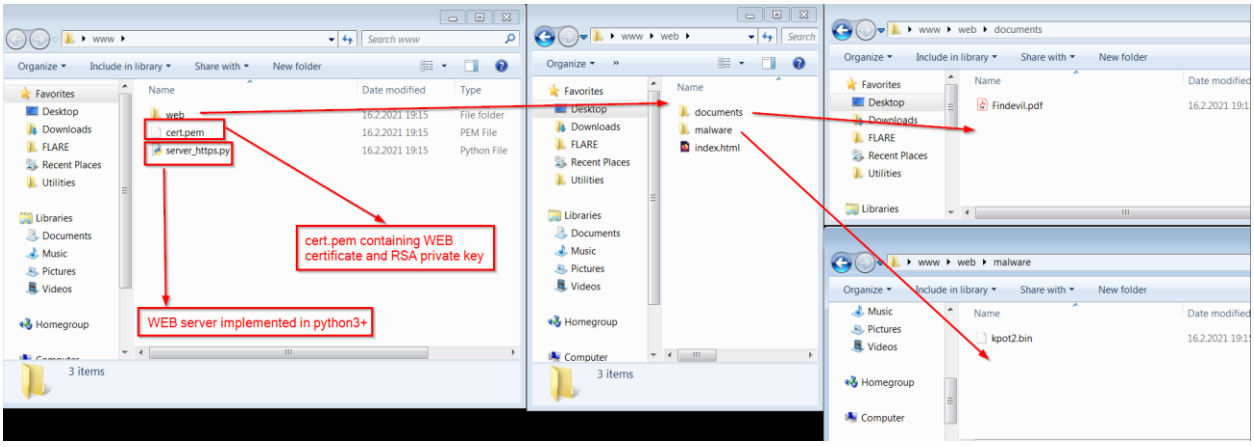


Start our VM and check the IP address on our newly created network interface (Host-only Adapter). In my case the IP “192.168.56.2” is used.

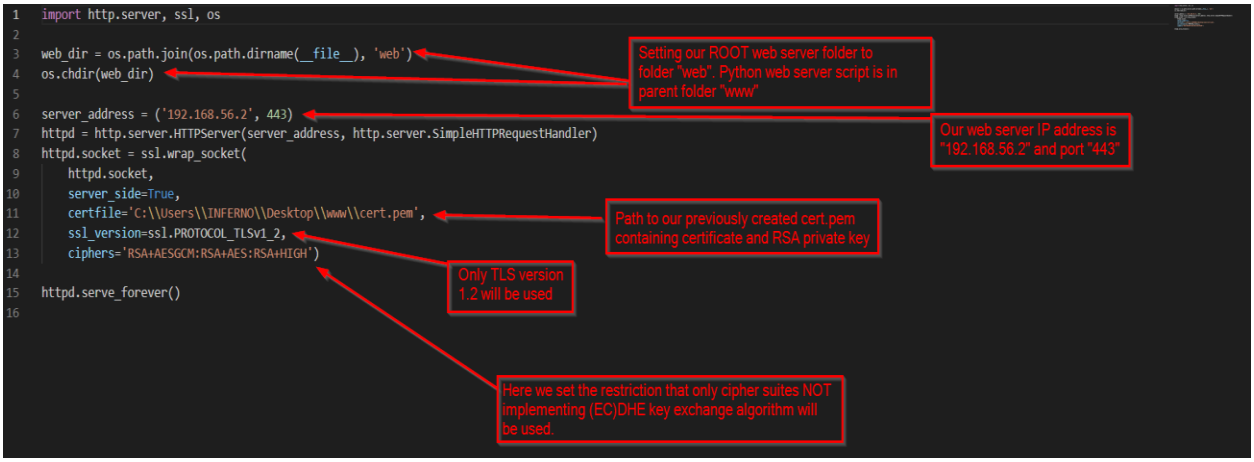
```
C:\Users\INFERNO\Desktop>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::fdeb:536f:c94e:e7b2%15
   IPv4 Address. . . . . . . . . . . : 192.168.56.2
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter Npcap Loopback Adapter:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::eddf:3ecf:6e61:4d25%13
   Autoconfiguration IPv4 Address. . : 169.254.77.37
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . :

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f149:4696:15bd:6174%11
   IPv4 Address. . . . . . . . . . . : 10.0.2.15
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.2.2

Tunnel adapter isatap.{7F2E8261-B763-4A33-8154-B7ECDDD00C73}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter isatap.{9489E21D-B7A4-4904-8AFC-7DE9310DB243}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter isatap.{CA791937-459D-4EBD-987C-AACBFEE4ABE5}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\INFERNO\Desktop>_
```

Now we will copy our previously created cert.pem containing both certificate and RSA private key to the VM.

We also implement the web server in python, some optional folders/files and some example html web content (Download here: [www content])



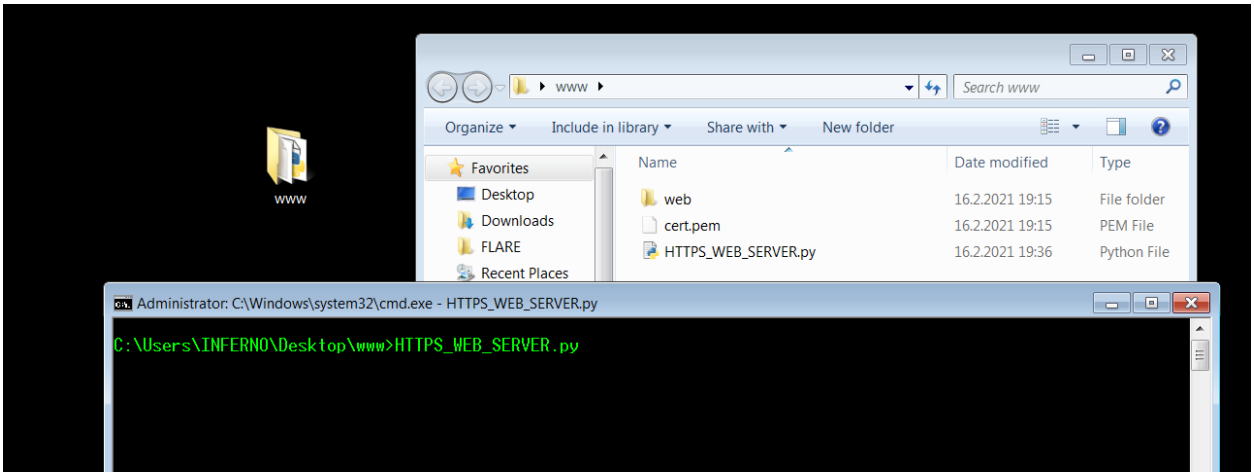
The web server implementation in Python3+ is in the picture below and you can download this example from: [HTTPS_WEB_SERVER.py]



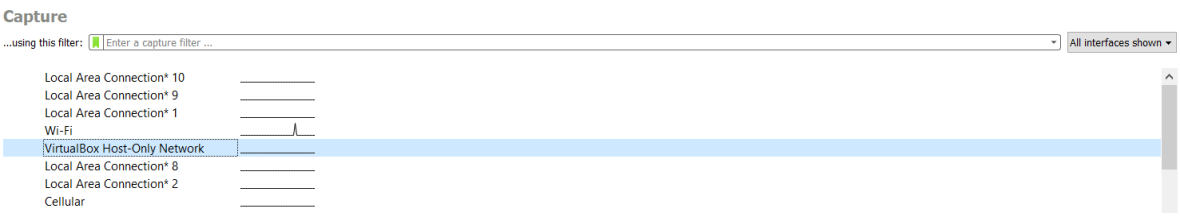
As we can see, we configured the Python web server to meet all conditions for TLS decryption via RSA private key method in Wireshark. (TLS version 1.2, only cipher suites NOT implementing (EC)DHE key exchange algorithm, cert.pem containing our RSA private key)

The used IP address is that we previously obtained from interface (Host-only Adapter).

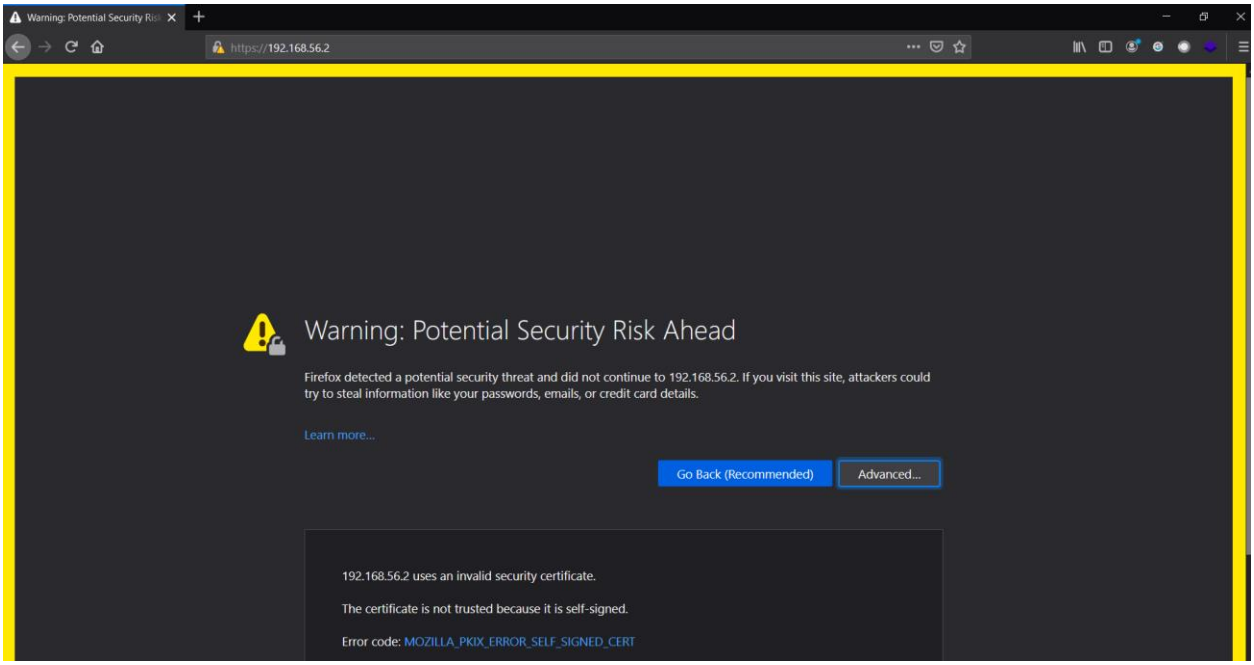
Now we can start our Python web server in VM.



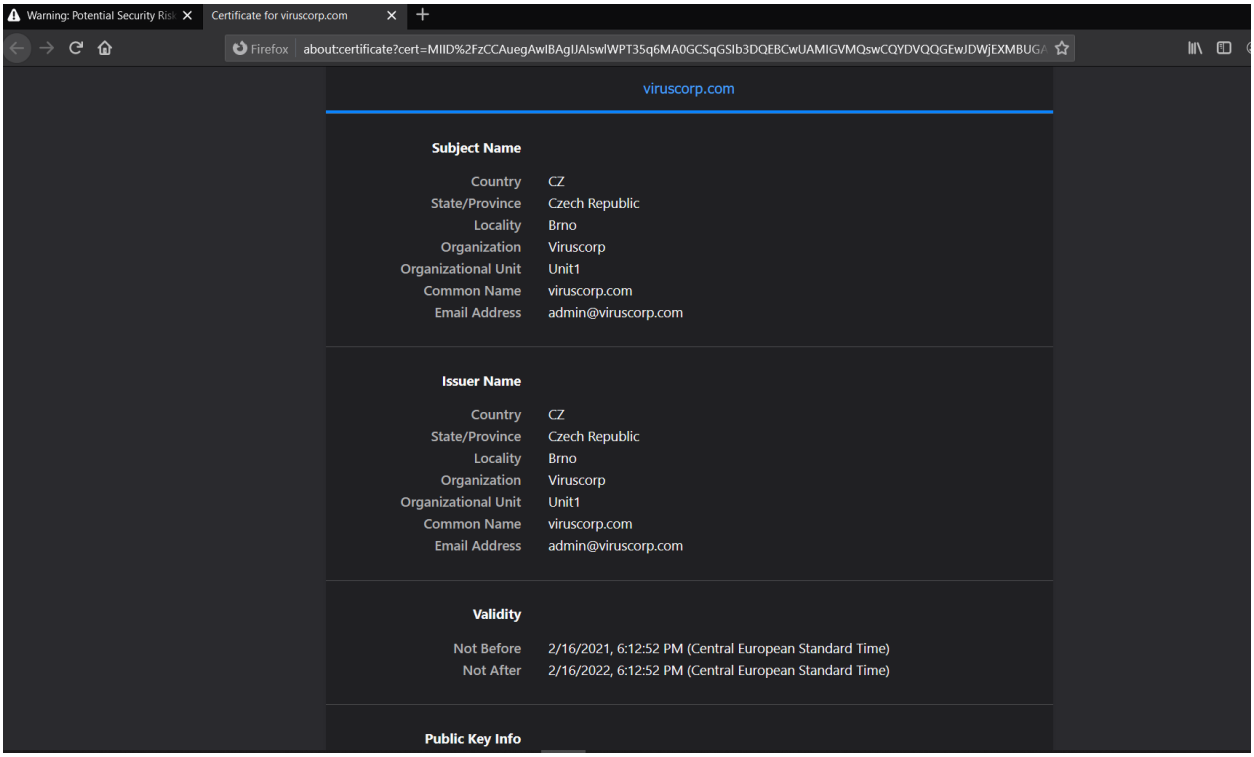
Start Wireshark in Host and choose interface “VirtualBox Host-Only Network”.



Start web browser in host and navigate to hxxps[://]192.168.56.2

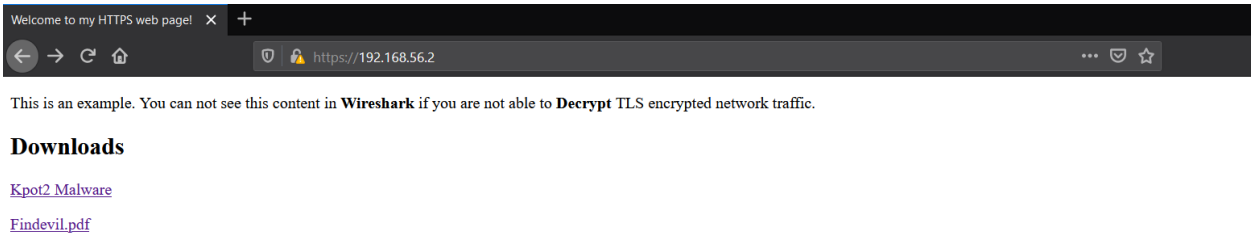


As we can see our certificate is self-signed and it is not trusted (cannot be verified). Let´s check the certificate. We got our CERT information as we expected.



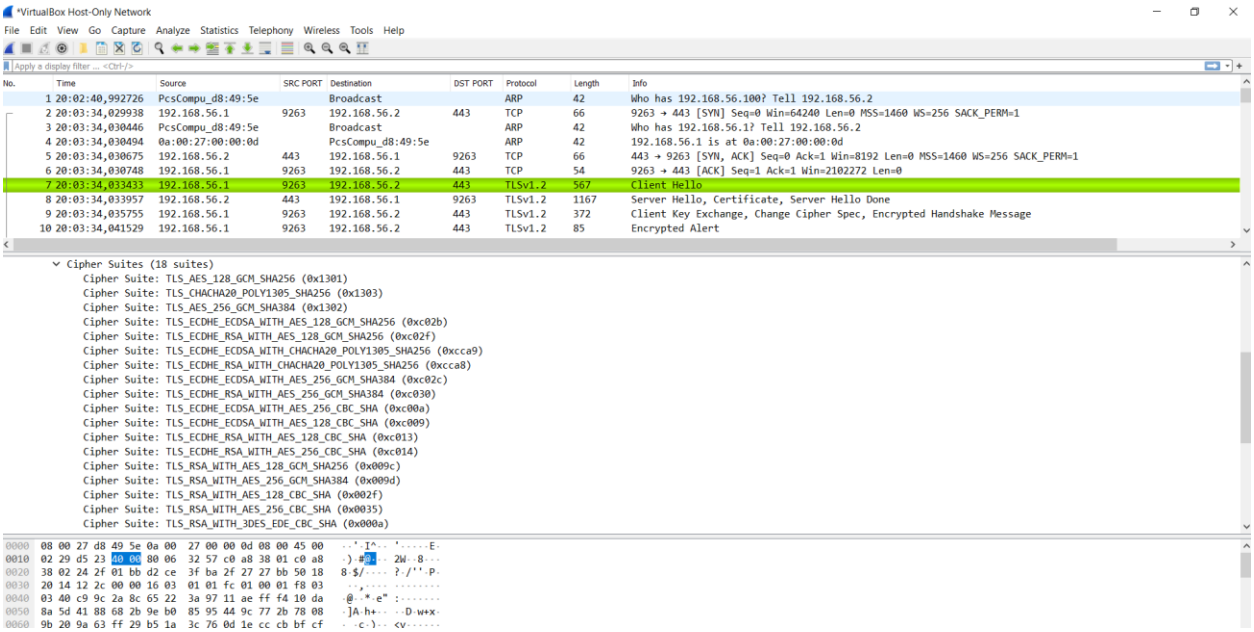
Let´s proceed (Accept the RISK and Continue).

We reached our previously created index.html example served via our Python web server implementation. You can download this index.html example here: [index.html]

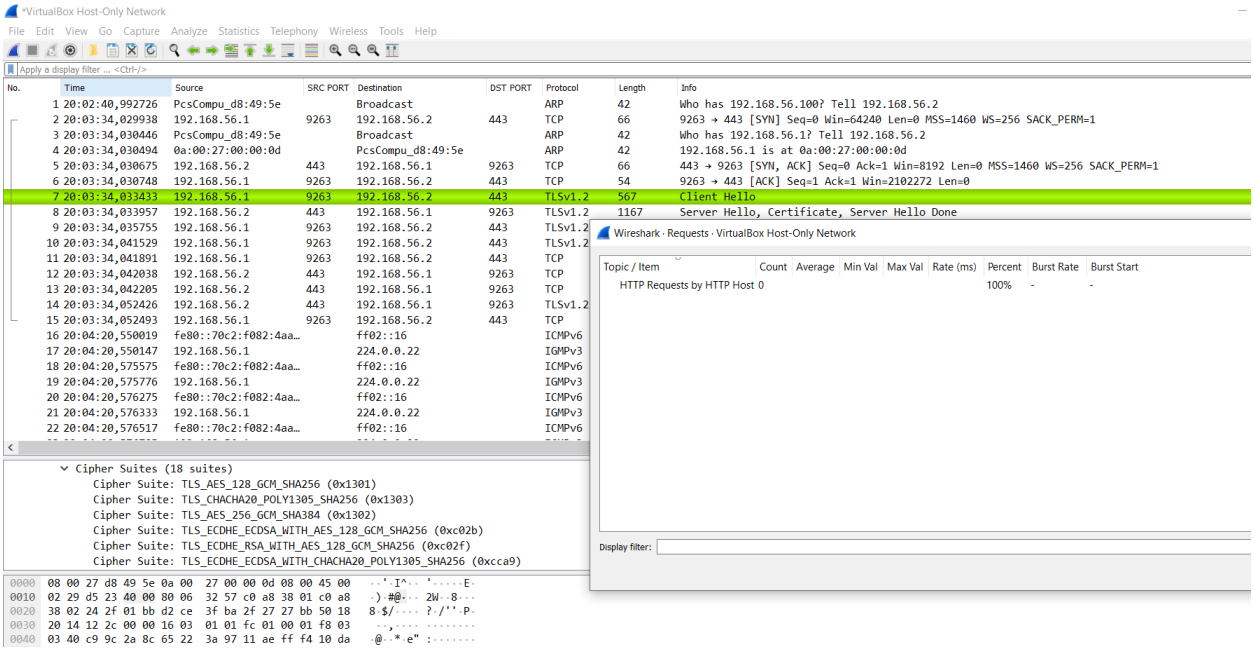


We will download included files, exit the web browser and check the Wireshark.

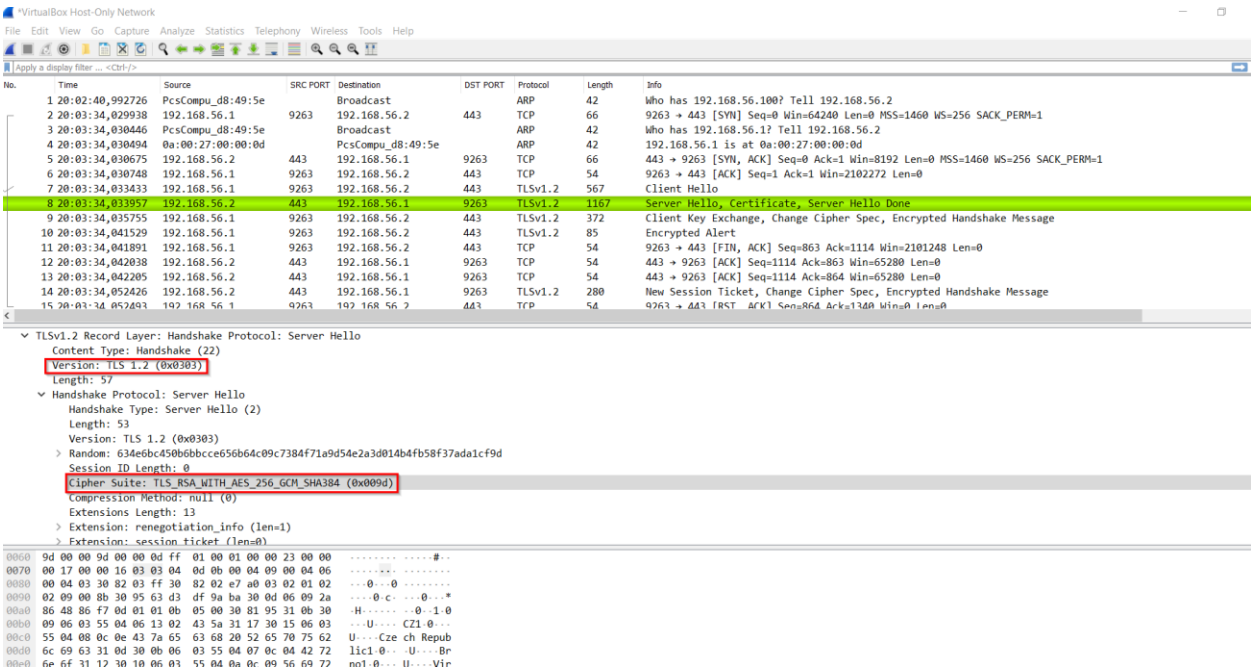
We can see in Wireshark that during TLS handshake, our web browser is offering 18 cipher suites.



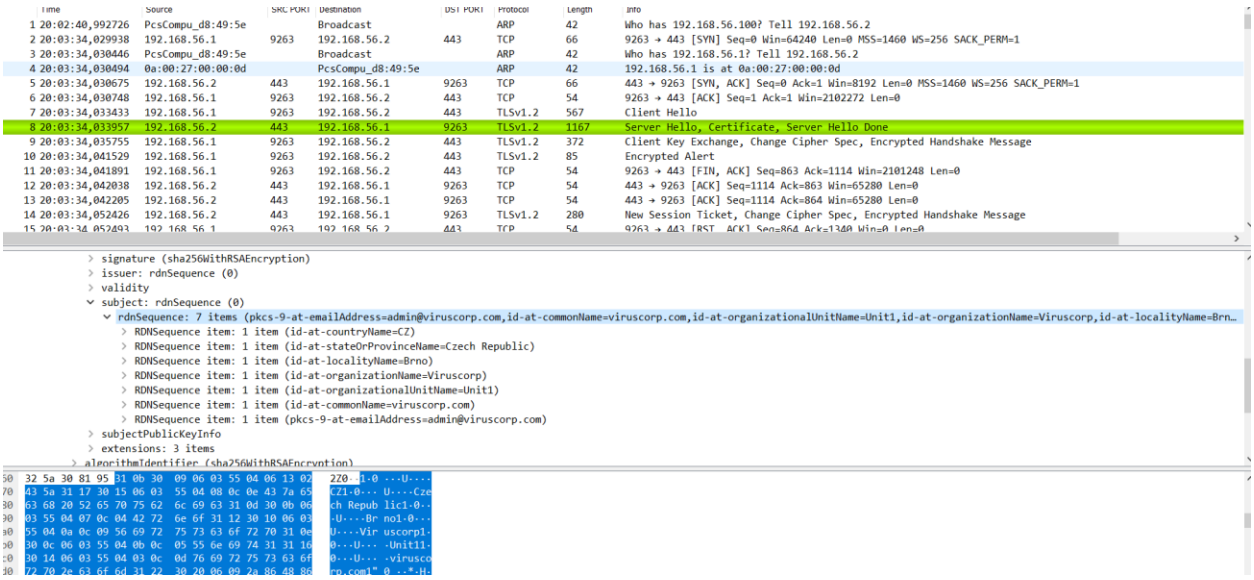
There is no HTTP traffic available and all is encrypted via TLS.



We can check that our Python web server is choosing TLS version 1.2 and cipher suite "TLS_RSA_WITH_AES_256_GCM_SHA384" during TLS negotiation (Handshake) as we previously configured.



We can also check the web server certificate information.



Now is the time to decrypt TLS and obtain HTTP traffic.

According to Wireshark documentation there are 2 methods how we can perform TLS decryption via RSA private key in Wireshark:

Starting with Wireshark 3.0, a new RSA Keys dialog can be found at *Edit* -> *Preferences* -> *RSA Keys*. In this dialog, use the *Add new keyfile...* button to select a file. You will be prompted for a password if necessary. The *Add new token...* button can be used to add keys from a HSM which might require using *Add new provider...* to select select a DLL/.so file, and additional vendor-specific configuration.

The RSA key file can either be a *PEM* format private key or a PKCS#12 keystore (typically a file with a .pfx or .p12 extension). The PKCS#12 key is a binary file, but the PEM format is a text file which looks like this:

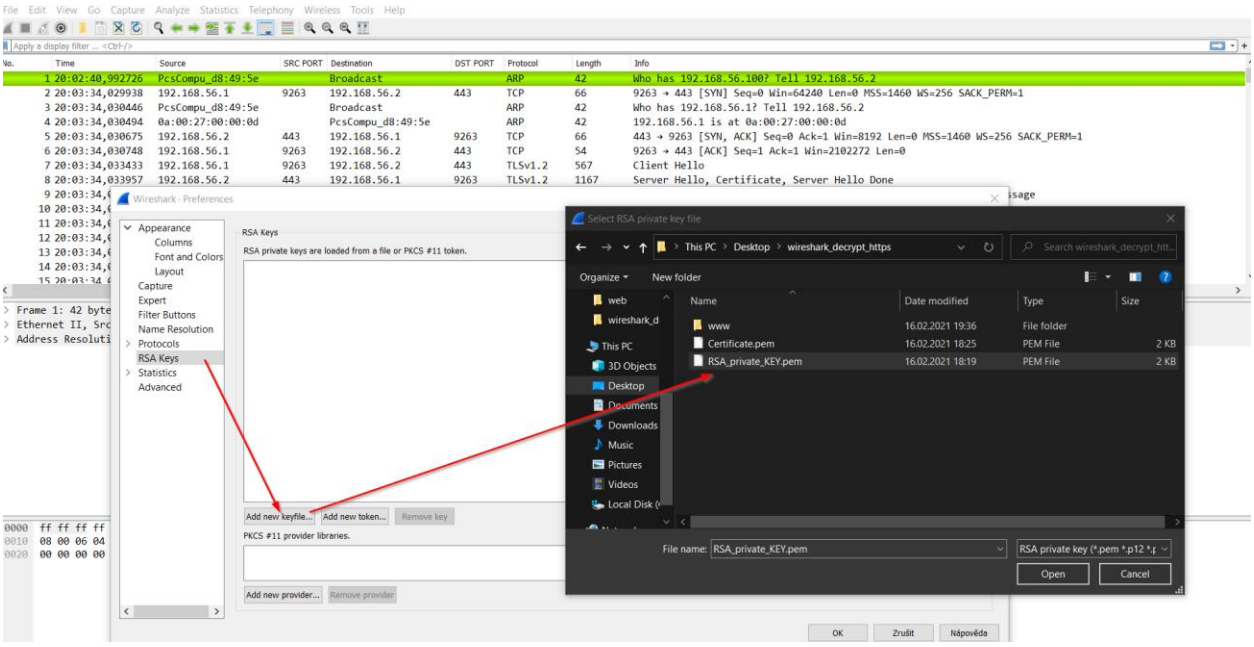
- -----BEGIN PRIVATE KEY-----
- MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDReQzlKVeAK8b5
- TRcRBhSi9IYwHX8Nqc8K4HeDRvN7HiBQQP3bhUkVekdoXpRLYVuc7A8h1BLr93Qw
- ...
- KOi8FZl+jhG+p8vtpK5ZAIyp
- -----END PRIVATE KEY-----

The deprecated *RSA keys list* dialog may be removed at some point. To configure keys, use the *RSA keys* dialog instead. To change the protocol for decrypted network data, right-click on a TLS packet and use *Decode As* to change the *Current* protocol for the *TLS port*. The *IP address* and *Port* fields are unused. -

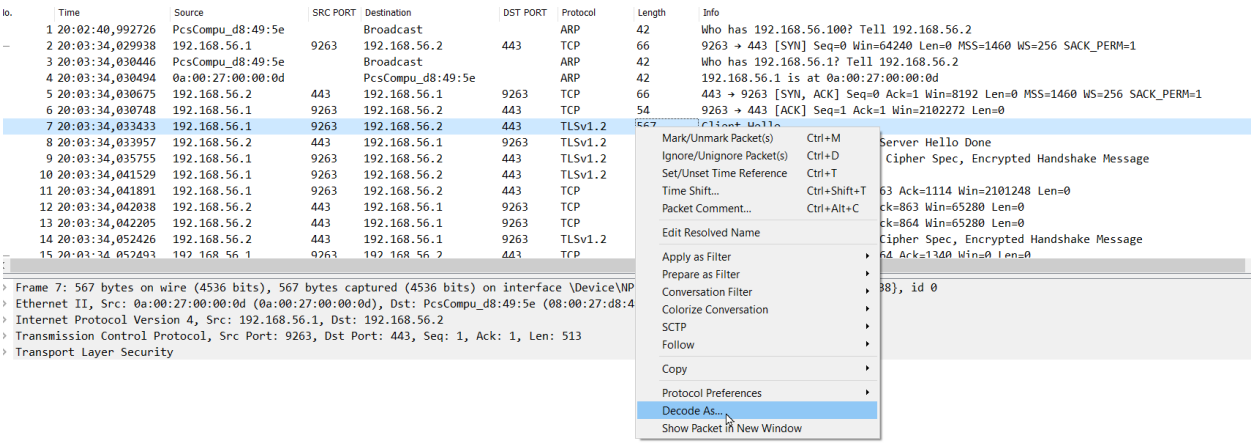
---

So navigate to Wireshark -> Edit -> Preferences -> RSA Keys

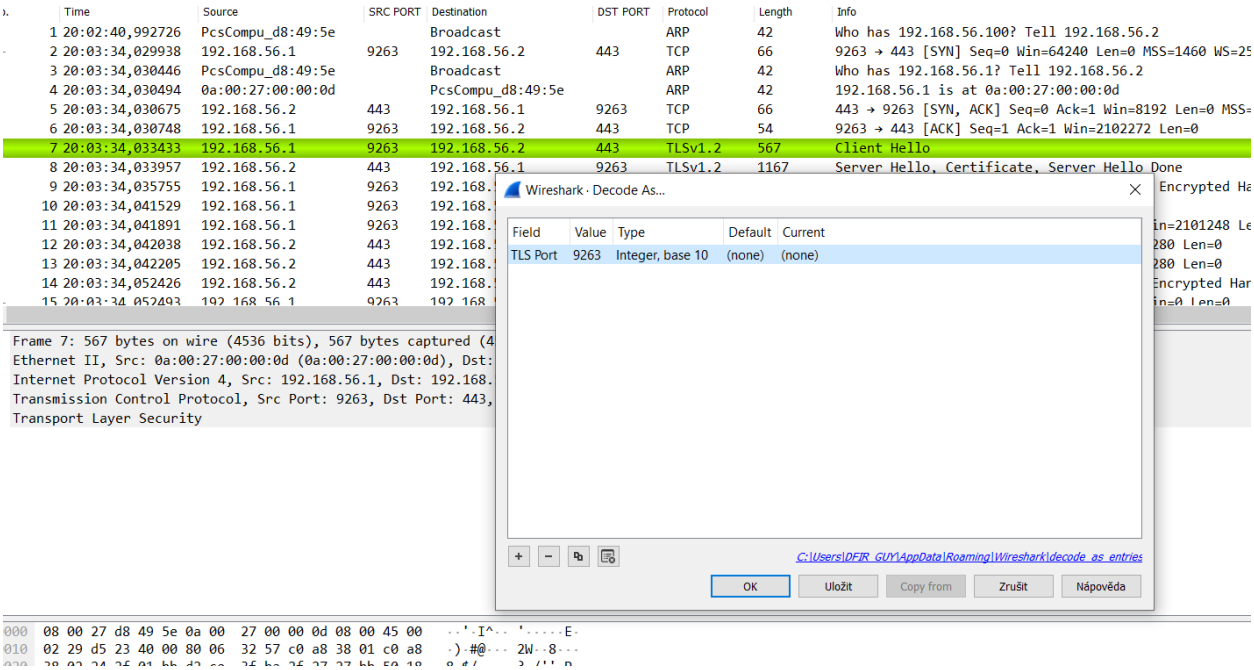
Add new key file (Our web server private key “RSA_private_KEY.pem”) and confirm importing.



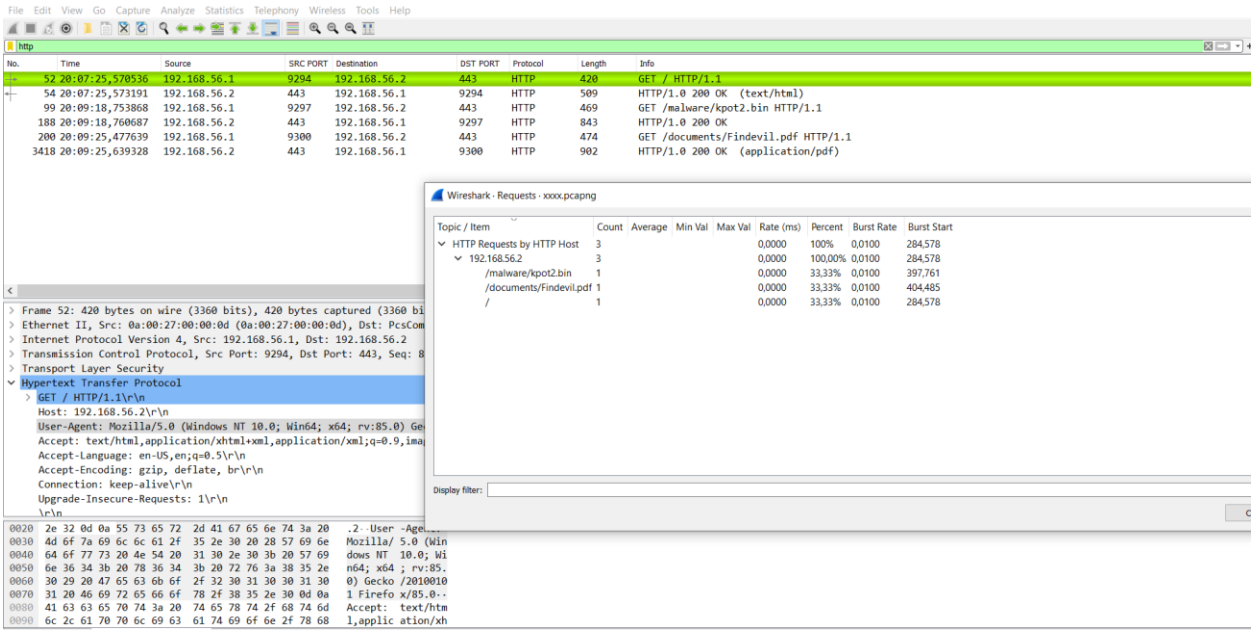
Go back to Packet list view in Wireshark and find the first packet of TLS protocol (DST IP of our web server). Mouse right click → Decode As…



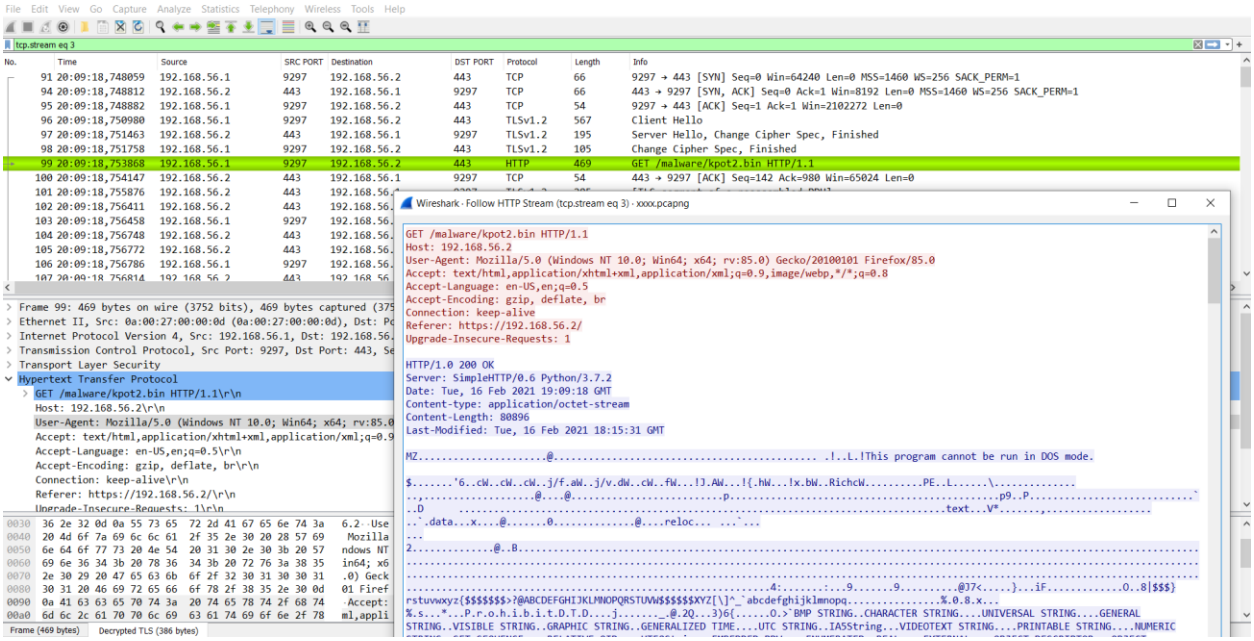
Click OK.



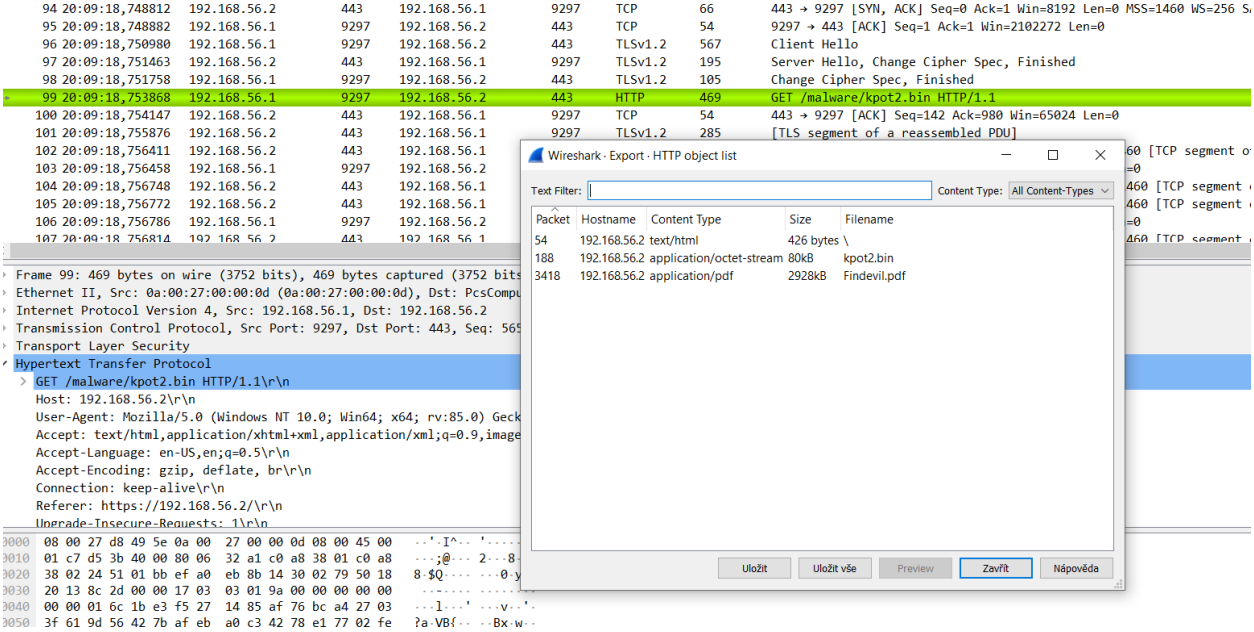
Now we can see all TLS decrypted web traffic in Wireshark.



We can see decrypted HTTP streams.



We can also export all HTTP objects as we can in normal HTTP traffic when TLS is not used.



Conclusion:

Each of method, which was shown, has its advantages and disadvantages. You can perform TLS decryption via RSA private key not only for TLS encrypted HTTP traffic but also for another protocols. In both methods, there are some restriction in use. I hope that this post will serve somebody as simple guide how one can decrypt TLS traffic in Wireshark. The main point of this was to show that even if you have the RSA private key, there are situation where it cannot be used for successful TLS decryption as mentioned in post. I would also like to thank Wireshark for its one of the best and well maintained documentation (not only for the best network protocol analyzing tool).

Useful links:

[https://wiki.wireshark.org/TLS]

[https://en.wikipedia.org/wiki/Cipher_suite]

Author:

[Twitter]

[Github]