

TLS mit Wireshark entschlüsseln



TLS mit Wireshark entschlüsseln

Was es beim kriminellen Man in the Middle zu verhindern gilt, gehört bei legal agierenden Systemadmins zum notwendigen Handwerkszeug: der Zugriff auf verschlüsselte Datenströme zwecks Fehlersuche.

Was es beim kriminellen Man in the Middle zu verhindern gilt, gehört bei legal agierenden Systemadmins zum notwendigen Handwerkszeug: der Zugriff auf verschlüsselte Datenströme zwecks Fehlersuche.

Von Benjamin Pfister

Der Anteil des verschlüsselten Datenverkehrs nimmt ständig zu. Fast alle Webdienste nutzen Transport Layer Security (TLS) und aktuelle Browser warnen bei unverschlüsselten HTTP-Verbindungen ausdrücklich vor dem damit verbundenen Risiko. Das ist aus Sicht der Sicherheit und des Datenschutzes sehr zu

begrüßen – doch die Verschlüsselung verhindert auch eine legale Analyse des Datenstroms, etwa seitens berechtigter Admins. Es gibt jedoch Möglichkeiten der Fehlersuche trotz TLS-Verschlüsselung, zum Beispiel mit dem im Folgenden beschriebenen Paketanalysewerkzeug Wireshark.

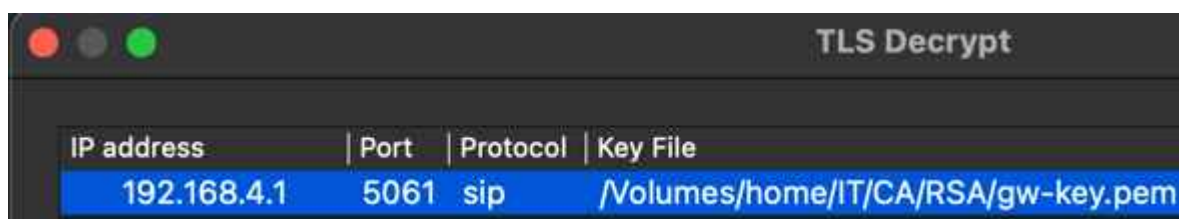
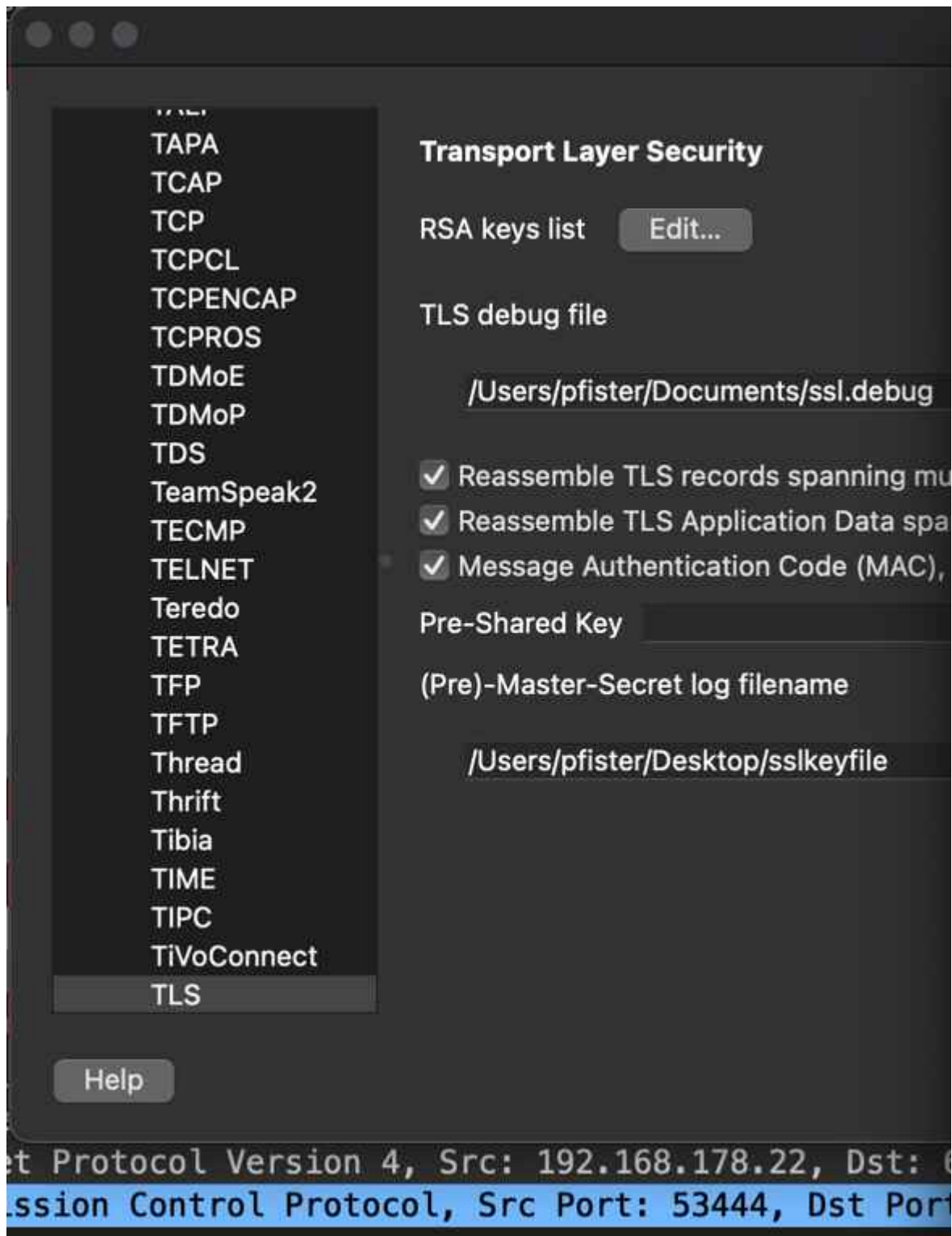
Wireshark bringt einen eigenen Dissector (wörtlich übersetzt Sezierer) für TLS mit. Er ermöglicht neben der Aufteilung und Darstellung der Protokolle auch die Entschlüsselung der Nutzdaten. Dazu bedarf es der passenden Schlüssel. Je nach eingesetzter Cipher Suite kommen unterschiedliche Entschlüsselungsmethoden zum Einsatz: auf Basis eines Session- (Pre-Master Secret) oder eines privaten RSA-Schlüssels.

Welche der beiden zur Anwendung kommt, hängt von der Cipher Suite ab: mit Perfect Forward Secrecy (PFS) oder ohne. Falls für die Übertragung keine PFS Cipher Suites vorgesehen sind, kann die Entschlüsselung auf Basis des privaten Schlüssels des Serverauthentifizierungszertifikats stattfinden. In diesem Fall kann Wireshark jedoch auch die Methode Pre-Master Secret nutzen. Dies ist beispielsweise dann interessant, wenn man – wie bei öffentlichen Webdiensten – nicht im Besitz der privaten Schlüssel ist.

Bei Nutzung von Perfect Forward Secrecy (PFS) lässt sich der Datenstrom selbst bei Kenntnis des privaten Schlüssels nicht nachträglich entschlüsseln. Daher empfiehlt das BSI zum Schutz personenbezogener oder anderer sensibler Daten diese Cipher Suites. Darunter fallen die Cipher Suites mit Diffie-Hellman Ephemeral (DHE) und Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Um diese Varianten zu entschlüsseln, muss man die Methode Pre-Master Secret einsetzen.

Der Besitz des privaten Schlüssels nützt also nur dann etwas, wenn keine (EC)DHE Cipher Suites zum Einsatz kommen. Zudem funktioniert diese erste Variante nicht mit TLS 1.3. Einen weiteren Fallstrick birgt der TLS Session Resume, bei dessen Anwendung das Entschlüsseln fehlschlägt. Es bedarf der

Aufzeichnung eines ClientKeyExchange im TLS Handshake. Zum Entschlüsseln der Daten benötigt man das Serverauthentifizierungszertifikat – genauer dessen privaten Schlüssel. In den TLS-Protokolleinstellungen von Wireshark und dem Menüpunkt „RSA keys list“ referenziert man die Datei mit dem privaten Schlüssel und verknüpft ihn mit der IP-Adresse, dem Port und dem Protokoll des Servers. Abbildung 1 zeigt eine solche Hinterlegung für die IP-Adresse 192.168.4.1 mit dem Port 5061 und dem Protokoll SIP. Der Private Key liegt im Beispiel unter /Volumes/Home/IT/CA/RSA/gw-key.pem. Daran erkennt man, dass nicht nur HTTPS als Applikationsprotokoll zur Verfügung steht. Die Referenzen liegen im Beispiel von macOS unter /Users/<username>/.config/wireshark/ssl_keys.



Hinterlegung des Private Key aus dem Serverauthentifizierungszertifikat (Abb. 1).

Nach der korrekten Hinterlegung beginnt der Dissector mit der Entschlüsselung. Bei eventuellen Fehlern lohnt ein Blick in

die TLS-Debug-Datei, die beispielsweise fehlerhafte Private-Key-Zuweisungen oder Probleme beim Laden der Private Keys aufzeigt. Deren Zielverzeichnis und Namen kann man selbst wählen (siehe Abbildung 1).

Auf der Kommandozeile kann man das in Wireshark enthaltene CLI-Tool tshark nutzen. Für die RSA-Methode lautet der Befehl

```
tshark -o "ssl.keys_list:192.168.4.1,5061,sip,/Volumes/Home/IT/CA/RSA/gw-key.pem" -r siptls.pcapng -Y sip
```

Über die Option

```
-o "ssl.keys_list:192.168.4.1,5061,sip,/Volumes/Home/IT/CA/RSA/gw-key.pem"
```

verknüpft man die in Abbildung 1 dargestellten Einstellungen – ähnlich wie mit der GUI-Variante. Das Argument `-r siptls.pcapng` liest dabei lediglich die PCAPNG-Datei. Das Argument `-Y sip` setzt einen Display-Filter auf das VoIP-Signalisierungsprotokoll SIP, sodass keine Pakete anderer Protokolle die Ausgabe fluten.

Die zweite Variante – keine Kenntnis des privaten Schlüssels und der Einsatz von (EC)DHE – setzt eine Keylog-Datei voraus, also eine Textdatei, die von unterschiedlichen Kryptobibliotheken bereitgestellt wird, beispielsweise OpenSSL oder NSS. Darauf aufbauende Applikationen wie Chrome, Firefox oder Curl generieren diese Datei, wenn die Umgebungsvariable `SSLKEYLOGFILE` gesetzt ist. Unter macOS kann man diese beispielsweise wie folgt anlegen: `export SSLKEYLOGFILE="/Users/<username>/Desktop/sslkeyfile"`.

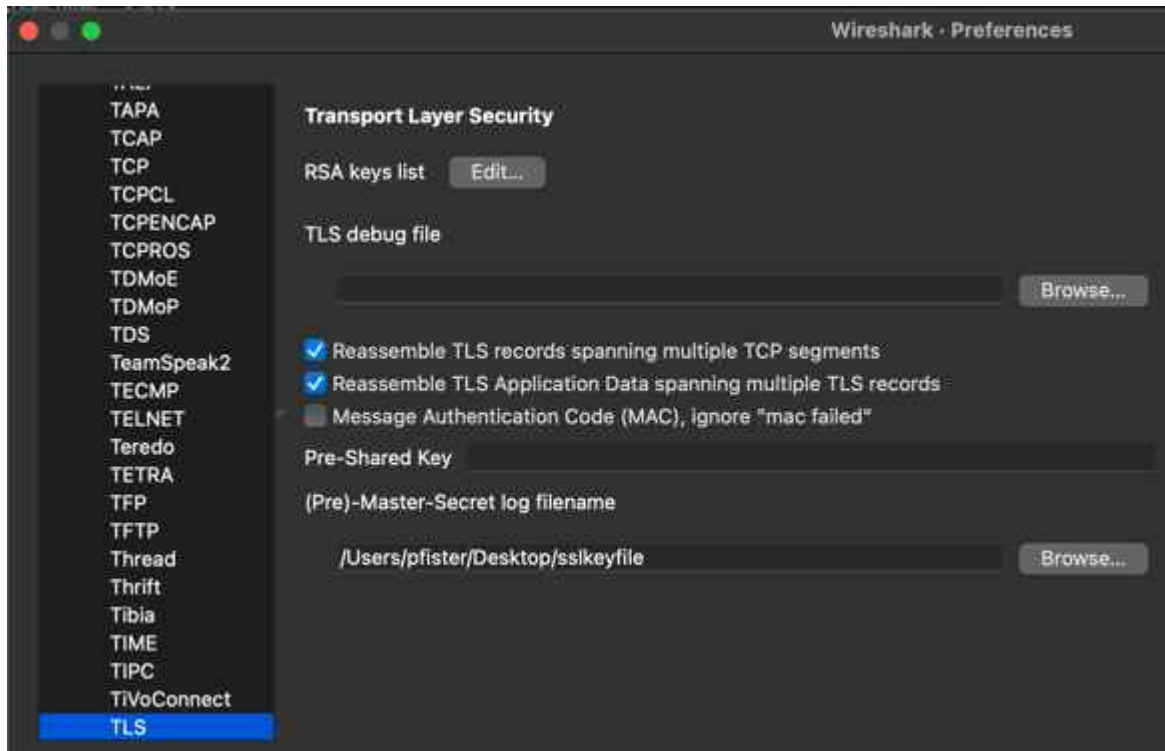
Die Bibliotheken schreiben den Pre-Master Key dann in die in der Umgebungsvariablen referenzierte Datei. Der Client generiert diesen in der Client Exchange Phase des TLS Handshake. Der Export kann auf Client- oder Serverseite stattfinden. Ein Mitlesen auf dem Transportweg ist somit nicht möglich. Wireshark kann den Pre-Master Key aus dem Handshake

dafür nutzen, den Master Key abzuleiten und damit den Datenverkehr zu entschlüsseln. Im Anschluss an die Konfiguration der Umgebungsvariablen startet man den Mitschnitt in Wireshark und öffnet dann über die Konsole beispielsweise Firefox mittels `open /Applications/Firefox.app` unter macOS. Nachdem die erste TLS-verschlüsselte Seite aufgerufen wurde, zeigt sich, ob die Schlüsseldatei korrekt gespeichert wurde. In der ersten Zeile der Datei erkennt man auch, dass die Bibliothek NSS für den Schreibvorgang zuständig war (siehe Abbildung 2).

```
pfister@wic ~ % cat Desktop/sslkeyfile
# SSL/TLS secrets log file, generated by NSS
CLIENT_HANDSHAKE_TRAFFIC_SECRET d9f54f9b831c8a29ee6438f4a86e6277f84663ba25f32bd08e466ce82d18488 75bcb0fef64e3386f7d2a23c39e98c45a77f84666b627138b1d880fca7e5V
a4e
SERVER_HANDSHAKE_TRAFFIC_SECRET d9f54f9b831c8a29ee6438f4a86e6277f84663ba25f32bd08e466ce82d18488 19d7275d9ee9fff7c615228e38f73a8ac29f930c235d67a34aa3788183c89e6
13a
CLIENT_TRAFFIC_SECRET_0 d9f54f9b831c8a29ee6438f4a86e6277f84663ba25f32bd08e466ce82d18488 07c8432787a3d944c13e8a338d137adfe761fde21885e81a9c869304a461619
SERVER_TRAFFIC_SECRET_0 d9f54f9b831c8a29ee6438f4a86e6277f84663ba25f32bd08e466ce82d18488 88986491e54e71a2e3a3748b6732c2a48eb085199b36448a973e7284392d6b0
EXPORTER_SECRET d9f54f9b831c8a29ee6438f4a86e6277f84663ba25f32bd08e466ce82d18488 b29eb720a38e3a18546c6eccfa2251ba2c3b18c46b52288b6df1727e0bfc
CLIENT_HANDSHAKE_TRAFFIC_SECRET 548a1296b77b22a686c5970184b1391a6ef7b5f2be5a66e3fa368838589a9c5 c378c843a2d2a8d81eadf8c3638e46294d535a12d046f86722823dc7456
4ed
SERVER_HANDSHAKE_TRAFFIC_SECRET 548a1296b77b22a686c5970184b1391a6ef7b5f2be5a66e3fa368838589a9c5 c8216553efbc3780ec52b1956f37ef5c02849764a9e95d67a4d689e6c63
a37
CLIENT_TRAFFIC_SECRET_0 548a1296b77b22a686c5970184b1391a6ef7b5f2be5a66e3fa368838589a9c5 815c3f0ea95e98673bf8205eb4323e7344896eb2ef86cd9890388551624868
SERVER_TRAFFIC_SECRET_0 548a1296b77b22a686c5970184b1391a6ef7b5f2be5a66e3fa368838589a9c5 9e776a8ba98a3bc38a87c558c1f1fa31de7c977a418e1865d34579aebdfca73a
EXPORTER_SECRET 548a1296b77b22a686c5970184b1391a6ef7b5f2be5a66e3fa368838589a9c5 24d1a864a980dfc77b82ef8b76448a0c3e2115b0e7e3ba1d6c9b089f8e871c062b
CLIENT_HANDSHAKE_TRAFFIC_SECRET 997a334266ba1a91eb07ae4e69ea72a7842f77e672a8d79a633b4c783314744 9c0866fad18d8d3b3fcc4d7d9a669f9e1748ad2c2b99dd3de208c98132f866
81d
SERVER_HANDSHAKE_TRAFFIC_SECRET 997a334266ba1a91eb07ae4e69ea72a7842f77e672a8d79a633b4c783314744 f91ba3841d91ce886f2198013b6739cf8fe75fa3a9f7d3caa673119aeb08
f4e
CLIENT_HANDSHAKE_TRAFFIC_SECRET 51f19abff428V3a59c0fe7ebcac84b3da2589594a4687d386796c6446356799 bf5850e31a8aaa498a79ba853fc8cd52760bcb97c48c4299791a053cdca4
f07
SERVER_HANDSHAKE_TRAFFIC_SECRET 51f19abff428V3a59c0fe7ebcac84b3da2589594a4687d386796c6446356799 bf5850e31a8aaa498a79ba853fc8cd52760bcb97c48c4299791a053cdca4
82e
CLIENT_TRAFFIC_SECRET_0 997a334266ba1a91eb07ae4e69ea72a7842f77e672a8d79a633b4c783314744 bffba13231a2d8eb5a5ff9d9d6078285de1dc218b79da1bcd77b94c351c
SERVER_TRAFFIC_SECRET_0 997a334266ba1a91eb07ae4e69ea72a7842f77e672a8d79a633b4c783314744 d22de98e1f2a1de1f27a7c0df1fd4ed16c8839997eb083c9e25e02a4b718f1b8
EXPORTER_SECRET 997a334266ba1a91eb07ae4e69ea72a7842f77e672a8d79a633b4c783314744 93c3a8ea9a108a92091652f632b6f5d67nb8c538e73ec1ae287cc1f75ac8061d
CLIENT_TRAFFIC_SECRET_0 51f19abff42893a59c0fe7ebcac84b3da2589594a4687d386796c6446356799 1a6cfb03ac436ba73ac92ad599a0887f384827379f833f233470b08046d8d5
SERVER_TRAFFIC_SECRET_0 51f19abff42893a59c0fe7ebcac84b3da2589594a4687d386796c6446356799 171baad411f22312b68380936c0d8f473dd841b86c47ad518498347593f03c
EXPORTER_SECRET 51f19abff42893a59c0fe7ebcac84b3da2589594a4687d386796c6446356799 92c2b8e7a4458a413762f11a3237fae349291179dacc28ba986728c378f88a
CLIENT_RANDOM c3b69cc79b55934f99f1e28f9ea9b1a9862a50f786a42ee08ad2712f0cfd15 b6f81aa198ad7mb8c635991ee5a748a37d6ee7b74d59978237d435aa7fddfb72766441a75649
4b8bd84f93a2a543
```

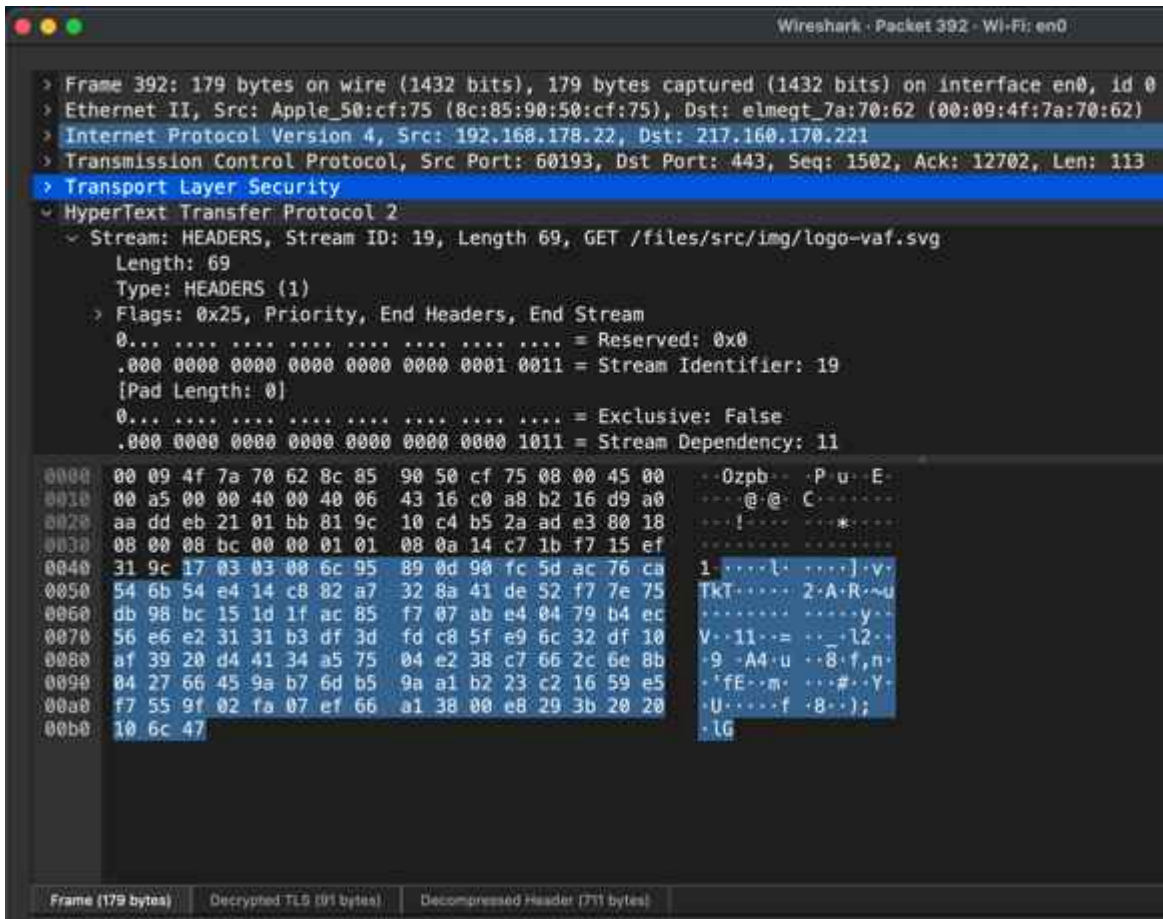
Nach dem Laden der ersten TLS-verschlüsselten Daten zeigt sich am `SSLKEYLOG`, ob die Schlüsseldatei korrekt gespeichert wurde (Abb. 2).

Damit Wireshark die Datei mit den passenden Schlüsseln zum Entschlüsseln heranzieht, ist sie als „(Pre)-Master-Secret log filename“ unter „Preferences/Protocols/TLS“ zu referenzieren (siehe Abbildung 3).



Referenzierung der PMK-Datei in den TLS-Protokolleinstellungen in Wireshark – in diesem Fall /Users/pfister/Desktop/sslkeyfile (Abb. 3).

Sobald der TLS Dissector in Wireshark den Traffic entschlüsselt hat, wird der HTTP2-GET-Request im Klartext lesbar (siehe Abbildung 4). Dass eine Entschlüsselung stattgefunden hat, zeigen die Angabe „HyperText Transport Protocol 2“ unterhalb der Zeile „Transport Layer Security“ und der Hinweis „Decrypted TLS“ im unteren Bereich.



Entschlüsselter HTTP2-GET-Request (Abb. 4).

Wer die Kommandozeile bevorzugt, kann mit tshark arbeiten – es folgt ein Beispiel einer Aufzeichnung und Entschlüsselung per tshark. Zunächst wird wieder die Umgebungsvariable angelegt, gefolgt vom Öffnen des Browsers Mozilla Firefox. Anschließend startet tshark für 60 Sekunden (-a duration:60) ohne direkte Ausgabe (-Q) und schreibt die aufgezeichneten Daten in eine PCAPNG-Datei (-w /Users/pfister/Desktop/tls_decrypt.pcapng). In der letzten Zeile liest tshark die PCAPNG-Datei (-r) mit dem Argument für die Referenz zur Keylog-Datei ein (-o tls.keylog_file:\$SSLKEYLOGFILE) und filtert die Ausgabe über einen Displayfilter auf HTTP (-Y http):

```
export SSLKEYLOGFILE="/Users/<username>/Desktop/sslkeyfile"
open /Applications/Firefox.app
tshark -Q -a duration:60 -w /Users/pfister/Desktop/tls_decrypt.pcapng &
tshark -r /Users/pfister/Desktop/tls_decrypt.pcapng -o
tls.keylog_file:$SSLKEYLOGFILE -Y http
```

Fazit

Mit der Session-Key-Methode lassen sich selbst aktuelle Protokolle wie TLS 1.3 entschlüsseln. Dafür bedarf es jedoch einer Applikation, die den Sessionschlüssel in eine Logdatei schreibt. Falls dies nicht der Fall ist und Server und Client keine (EC)DHE Cipher Suite nutzen, kann der Analyst als Fallback die RSA-Methode anwenden. Grundsätzlich kann die Möglichkeit einer Entschlüsselung ein Troubleshooting jedenfalls immens erleichtern. Wireshark bietet dafür einen recht einfach zu nutzenden Ansatz. (un@ix.de)

1. Quellen
2. [Weiterführende Informationen finden sich unter ix.de/ztmc.](https://www.ix.de/ztmc)