

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.

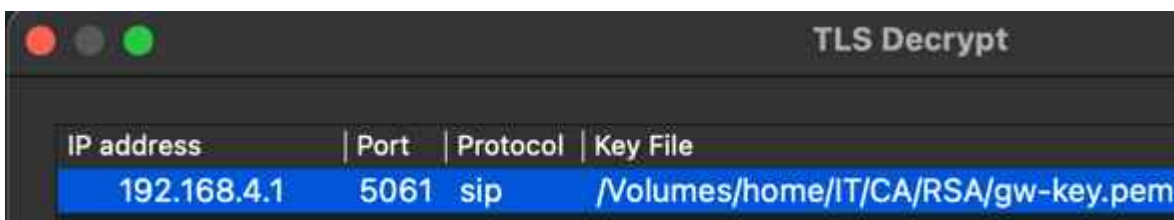
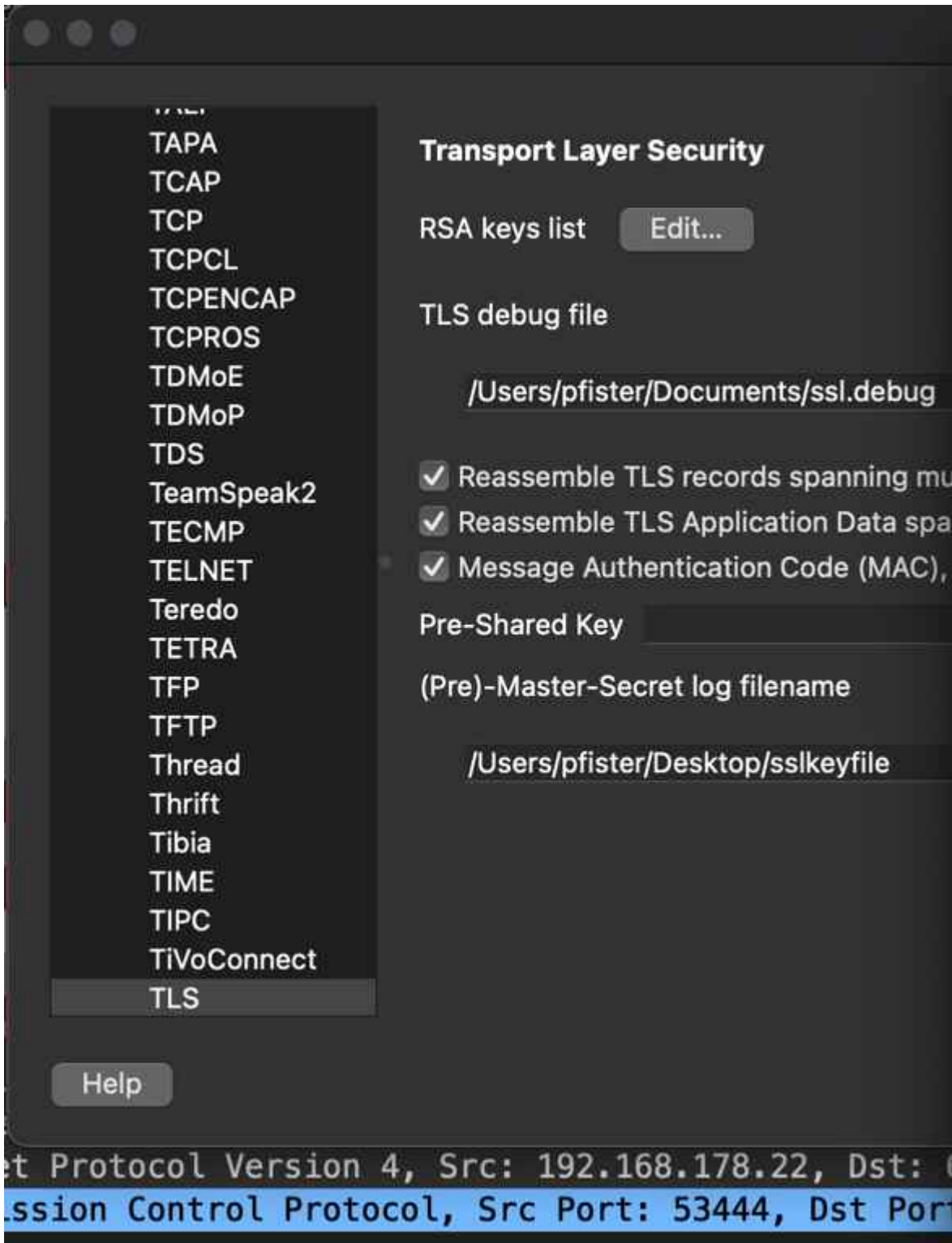
Wireshark bringt einen eigenen Dissector (wörtlich übersetzt Sezierer) für TLS mit. Er ermöglicht neben der Aufteilung und Darstellung der Protokolle auch die Entschlüsselung der Nutzdaten. Dazu bedarf es der passenden Schlüssel. Je nach eingesetzter Cipher Suite kommen unterschiedliche Entschlüsselungsmethoden 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"`.

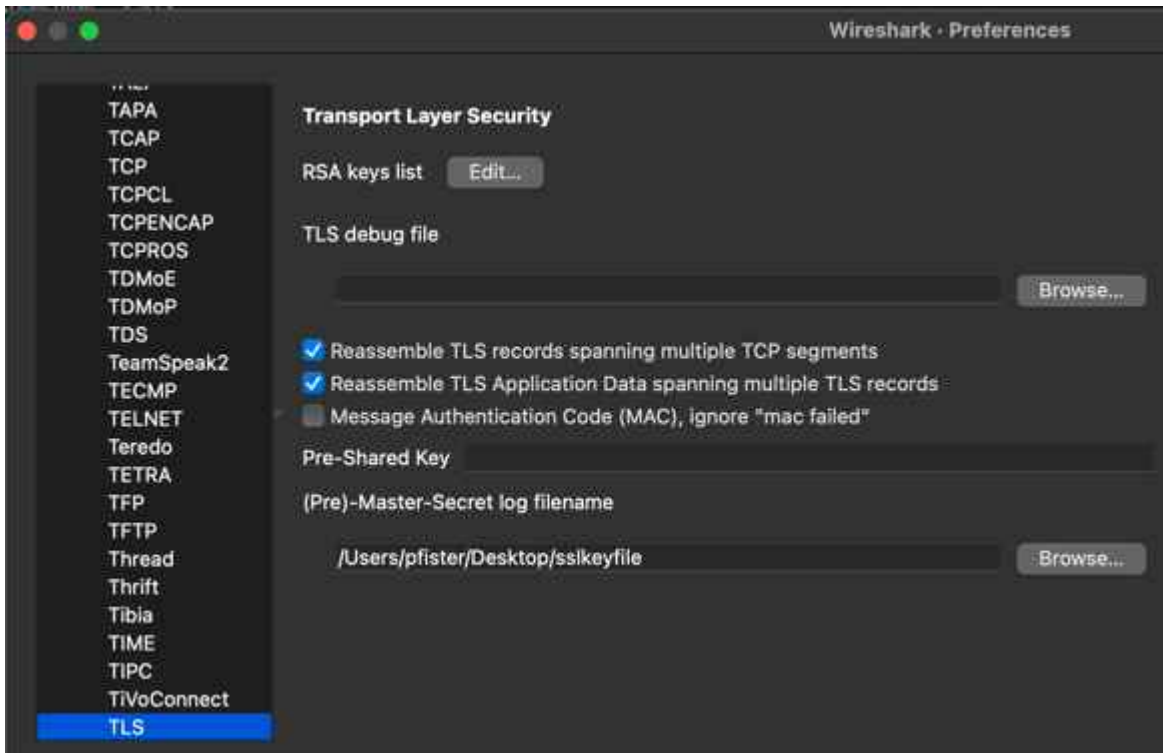
Die Bibliotheken schreiben den Pre-Master Key dann in die in der Umgebungsvariablen referenzierte Datei. Der Client generiert diesen in der Client Exchange Phase des TLS Handshake. Der Export kann auf Client- oder Serverseite stattfinden. Ein Mitlesen auf dem Transportweg ist somit nicht möglich. Wireshark 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 sie die Bibliothek NSS für den Schreibvorgang zuständig war (siehe Abbildung 2).

```
pfister@dwic ~ % cat Desktop/sslkeyfile
# SSL/TLS secrets log file, generated by NSS
CLIENT_HANDSHAKE_TRAFFIC_SECRET d9f54f9b831c8a29ee5438f4a80e6277f8463ba25f32bd0d8e46ce82d10480 75bcb0fe4e4338ef7d2a23c39e98c45a77f8a6c58627138b1d880fca7e5V
4e8
SERVER_HANDSHAKE_TRAFFIC_SECRET d9f54f9b831c8a29ee5438f4a80e6277f8463ba25f32bd0d8e46ce82d10480 19d7275d9ee9fff77c615228e3873a8ac29930c2138d67a3aa3788183c89e6
13a
CLIENT_TRAFFIC_SECRET_0 d9f54f9b831c8a29ee5438f4a80e6277f8463ba25f32bd0d8e46ce82d10480 e7c8432787a3d941c813eaa338d137adfe781f8e21885e81a9c869804a41619
SERVER_TRAFFIC_SECRET_0 d9f54f9b831c8a29ee5438f4a80e6277f8463ba25f32bd0d8e46ce82d10480 80966a81e54e71a2e6a37a8b6732c2ac4be8085199b3644a973e7284392d65b
EXPORTER_SECRET d9f54f9b831c8a29ee5438f4a80e6277f8463ba25f32bd0d8e46ce82d10480 b29eb770a35e3a18546c6ccfa2251b4e2cb3618c46e5222886af1272f8bfc
CLIENT_HANDSHAKE_TRAFFIC_SECRET 548a1296b77b22a080c5970184b13910a0ef7b5f2be5a6e3fa368038589a9c5 c378c843e22a801eadf8c2638e42942d535a12d046f84722823cc7456
4ed
SERVER_HANDSHAKE_TRAFFIC_SECRET 548a1296b77b22a080c5970184b13910a0ef7b5f2be5a6e3fa368038589a9c5 c8216553efbc3780ec52b1956f37efc62847664e9e95667a4d8689e6c63
e37
CLIENT_TRAFFIC_SECRET_0 548a1296b77b22a080c5970184b13910a0ef7b5f2be5a6e3fa368038589a9c5 815c380ea95e98673b78205eb4323e7344b96eb2ef86c6d99038085162a8e8
SERVER_TRAFFIC_SECRET_0 548a1296b77b22a080c5970184b13910a0ef7b5f2be5a6e3fa368038589a9c5 9e77688ba98a3bc38a87c558c12f8a11de7e977a418e1805d3d379aebdfca73e
EXPORTER_SECRET 548a1296b77b22a080c5970184b13910a0ef7b5f2be5a6e3fa368038589a9c5 2cd14865a798df1c72b82efdb7648anc3e21153e7e3ba1d6cc9089f8e871c662b
CLIENT_HANDSHAKE_TRAFFIC_SECRET 997a334266ba1a91e0b7ae4e69ea72a7842f77e672a8d7ea833bc783314744 9c096efad18edd3b3fcc4d7d9a669f9c1748ad2c2199dd3de208f8132f8e6
81d
SERVER_HANDSHAKE_TRAFFIC_SECRET 997a334266ba1a91e0b7ae4e69ea72a7842f77e672a8d7ea833bc783314744 f91b3841d91ce886f719813b5739bf8fe75f8a3a9f7d1caa473115e0e08
f4e
CLIENT_HANDSHAKE_TRAFFIC_SECRET 51f196bffa2893a59c6fe7ebccac84b3da2589594e4687d38a796c6446356799 bf5865c31a8aaa498a79ba053fc8cd52756bcb97c48c4299791a053cdc4
fad
SERVER_HANDSHAKE_TRAFFIC_SECRET 51f196bffa2893a59c6fe7ebccac84b3da2589594e4687d38a796c6446356799 dfed5309a9c8cc0837737aaef5e5fc8ebc2325e84863878e7144aa9f54
82e
CLIENT_TRAFFIC_SECRET_0 997a334266ba1a91e0b7ae4e69ea72a7842f77e672a8d7ea833bc783314744 bf18a113231e2d85ba8ff09d6078285de1dc218db79da1bcd77b94c351c
SERVER_TRAFFIC_SECRET_0 997a334266ba1a91e0b7ae4e69ea72a7842f77e672a8d7ea833bc783314744 d22e98e1f2a1de1f27a7ccdf1fd4ed1c88399978b08c9e25e02a4b718f1b8
EXPORTER_SECRET 997a334266ba1a91e0b7ae4e69ea72a7842f77e672a8d7ea833bc783314744 9c3e8ea9a108a92091652f632baf5d67m80c538e73ec1ae257cc1f75ac8061d
CLIENT_TRAFFIC_SECRET_0 51f196bffa2893a59c6fe7ebccac84b3da2589594e4687d38a796c6446356799 1a5cf8b3ac436ba73cc92ad599e0887f284877379f833f233479b38640d88d
SERVER_TRAFFIC_SECRET_0 51f196bffa2893a59c6fe7ebccac84b3da2589594e4687d38a796c6446356799 172baad411f22512d8380936cc08f473d8d421b56c47ad518498847897f03c
EXPORTER_SECRET 51f196bffa2893a59c6fe7ebccac84b3da2589594e4687d38a796c6446356799 9c2c08e74485aa13762f11aa3237aa349291179aac288a986728e3708f88a
CLIENT_RANDOM c3bd69c79b5599349897e2879ea9a9a982a50f78a42e98ad27127f0fcd15 b6f81a198dd7ab8c635991ee5748aa37de6e7574d59978227d6435aa7cfcfb72766441f7549e
488dd84f9324543
```

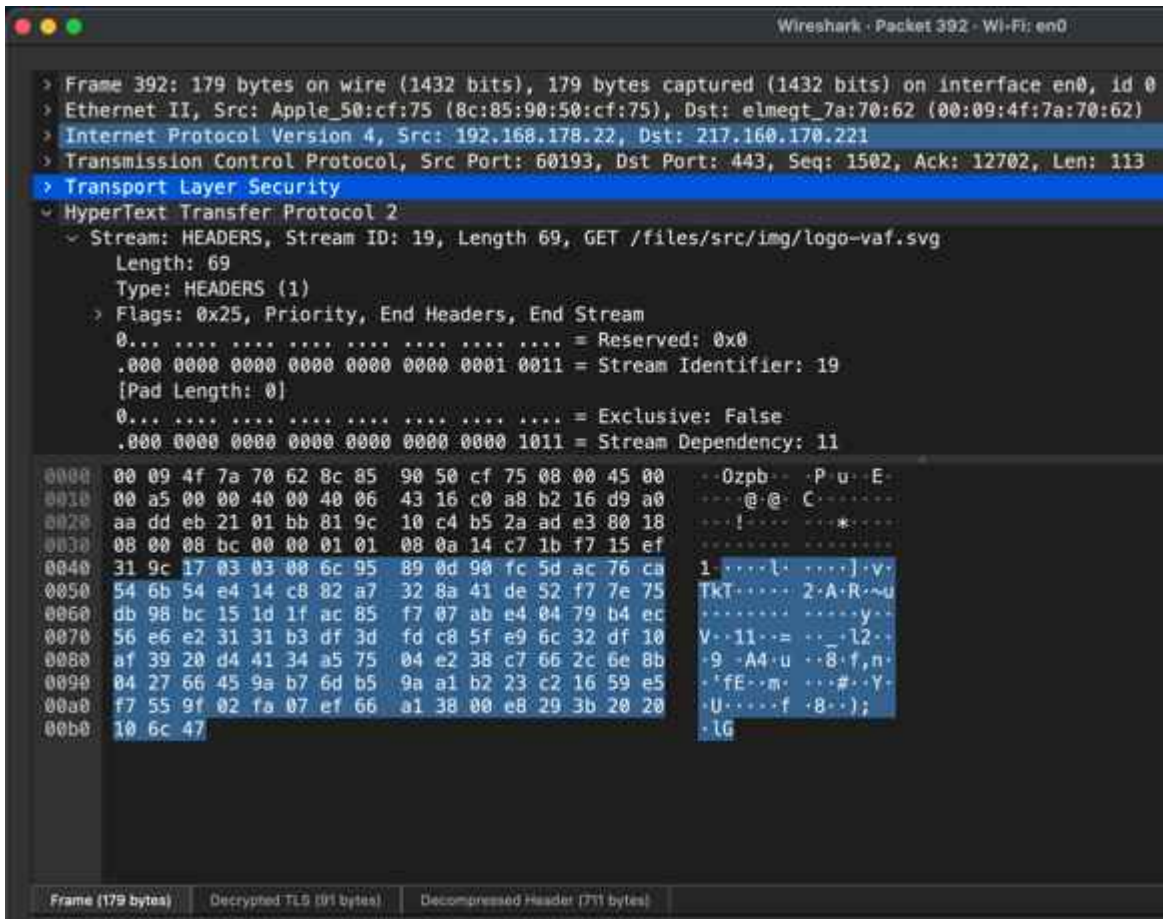
Nach dem Laden der ersten TLS-verschlüsselten Daten zeigt sich am SSLKEYLOG, ob die Schlüsseldatei korrekt gespeichert wurde (Abb. 2).

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)