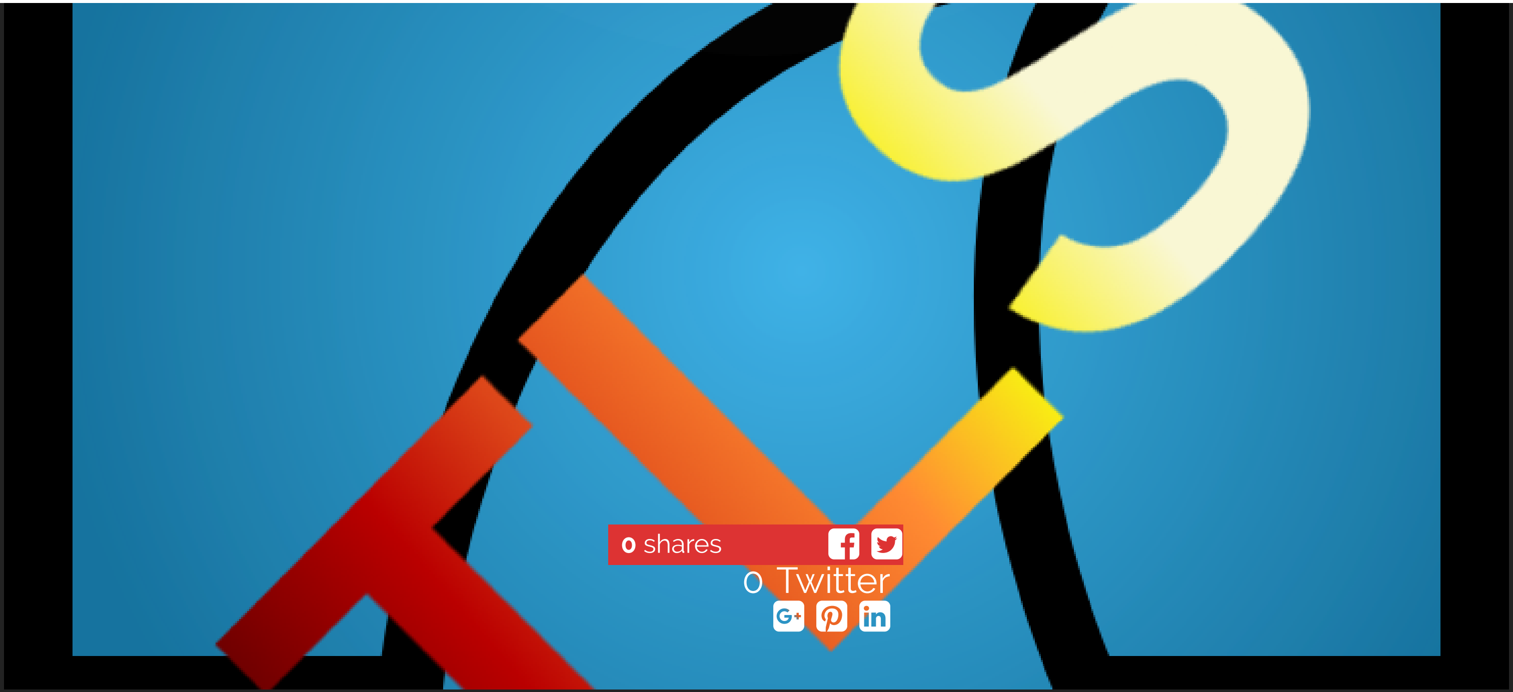


0 shares

0 Twitter

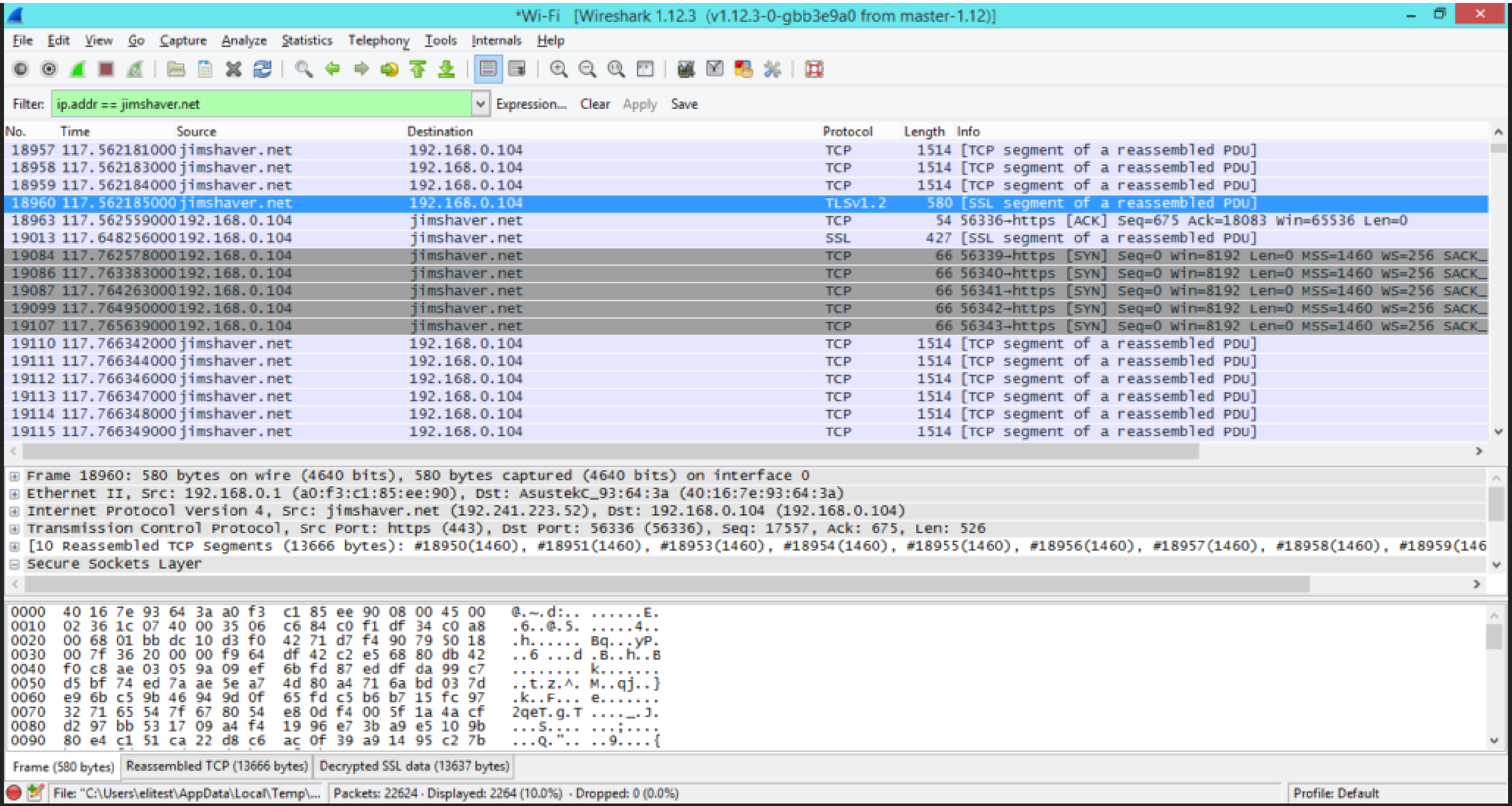
Decrypting TLS Browser Traffic With Wireshark – The Easy Way!

HOW-TO LATEST PEN TESTING SECURITY TOOLS UNCATEGORIZED March 10, 2019 elitest 2

Encryption 1 TLS 1 Wireshark 1

Intro

Most IT people are somewhat familiar with Wireshark. It is a traffic analyzer, that helps you learn how networking works, diagnose problems and much more.


*Wi-Fi [Wireshark 1.12.3 (v1.12.3-0-gbb3e9a0 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save
Frame (580 bytes)
Reassembled TCP (13666 bytes)
Decrypted SSL data (13637 bytes)
Profile: Default

```
Filter: ip.addr == jimshaver.net

No.   Time          Source          Destination      Protocol  Length Info
18957 117.562181000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
18958 117.562183000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
18959 117.562184000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
18960 117.562185000 jimshaver.net   192.168.0.104    TLSv1.2    580 [SSL segment of a reassembled PDU]
18963 117.562559000 192.168.0.104   jimshaver.net    TCP         54 56336→https [ACK] Seq=675 Ack=18083 Win=65536 Len=0
19013 117.648256000 192.168.0.104   jimshaver.net    SSL        427 [SSL segment of a reassembled PDU]
19084 117.762578000 192.168.0.104   jimshaver.net    TCP         66 56339→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19086 117.763383000 192.168.0.104   jimshaver.net    TCP         66 56340→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19087 117.764263000 192.168.0.104   jimshaver.net    TCP         66 56341→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19099 117.764950000 192.168.0.104   jimshaver.net    TCP         66 56342→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19107 117.765639000 192.168.0.104   jimshaver.net    TCP         66 56343→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19110 117.766342000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
19111 117.766344000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
19112 117.766346000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
19113 117.766347000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
19114 117.766348000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]
19115 117.766349000 jimshaver.net   192.168.0.104    TCP       1514 [TCP segment of a reassembled PDU]

⊞ Frame 18960: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface 0
⊞ Ethernet II, Src: 192.168.0.1 (a0:f3:c1:85:ee:90), Dst: AsustekC_93:64:3a (40:16:7e:93:64:3a)
⊞ Internet Protocol Version 4, Src: jimshaver.net (192.241.223.52), Dst: 192.168.0.104 (192.168.0.104)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: 56336 (56336), Seq: 17557, Ack: 675, Len: 526
⊞ [10 Reassembled TCP Segments (13666 bytes): #18950(1460), #18951(1460), #18953(1460), #18954(1460), #18955(1460), #18956(1460), #18957(1460), #18958(1460), #18959(146
⊟ Secure Sockets Layer

0000  40 16 7e 93 64 3a a0 f3  c1 85 ee 90 08 00 45 00   @.~.d:.. ......E.
0010  02 36 1c 07 40 00 35 06  c6 84 c0 f1 df 34 c0 a8   .6..@.5. .....4..
0020  00 68 01 bb dc 10 d3 f0  42 71 d7 f4 90 79 50 18   .h...... Bq...yP.
0030  00 7f 36 20 00 00 f9 64  df 42 c2 e5 68 80 db 42   ..6 ...d .B..h..B
0040  f0 c8 ae 03 05 9a 09 ef  6b fd 87 ed df da 99 c7   ........ k.......
0050  d5 bf 74 ed 7a ae 5e a7  4d 80 a4 71 6a bd 03 7d   ..t.z.^. M..qj..}
0060  e9 6b c5 9b 46 94 9d 0f  65 fd c5 b6 b7 15 fc 97   .k..F... e.......
0070  32 71 65 54 7f 67 80 54  e8 0d f4 00 5f 1a 4a cf   2qeT.g.T ...._.J.
0080  d2 97 bb 53 17 09 a4 f4  19 96 e7 3b a9 e5 10 9b   ...S.... ...;....
0090  80 e4 c1 51 ca 22 d8 c6  ac 0f 39 a9 14 95 c2 7b   ...Q.".. ..9....{

File: "C:\Users\elitest\AppData\Local\Temp\...   Packets: 22624 · Displayed: 2264 (10.0%) · Dropped: 0 (0.0%)
```

Wireshark

One of the problems with the way Wireshark works is that it can't easily analyze encrypted traffic, like **TLS**. It used to be if you had the private key(s) you could feed them into Wireshark and it would decrypt the traffic on the fly, but it only worked when using **RSA** for the key exchange mechanism. As people have started to embrace **forward secrecy** this broke, as having the private key is no longer enough derive the actual session key used to decrypt the data. The other problem with this is that a private key should not or can not leave the client, server, or HSM it is in. This lead me to coming up with very contrived ways of man-in-the-middling myself to decrypt the traffic(e.g. **sslstrip** or **mitmproxy**).

Session Key Logging to the Rescue!

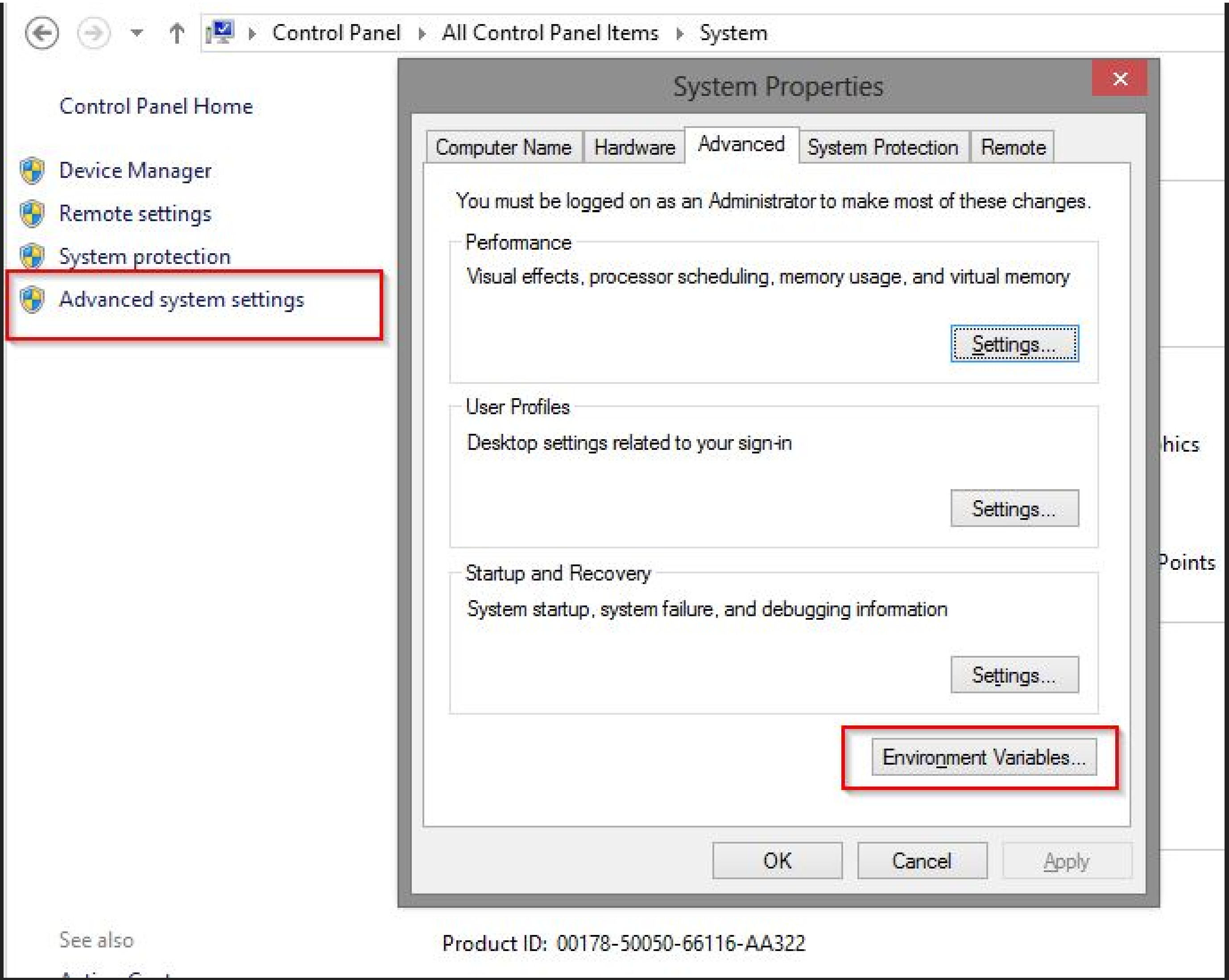
Well my friends I'm here to tell you that there is an easier way! It turns out that Firefox and Chrome both support logging the symmetric session key used to encrypt TLS traffic to a file. You can then point Wireshark at said file and presto! decrypted TLS traffic. Read on to learn how to set this up.

Setting up our Browsers

We need to set an environmental variable.

On Windows:

Go into your computer properties, then click "Advance system settings" then "Environment Variables…"



You must be logged on as an Administrator to make most of these changes.

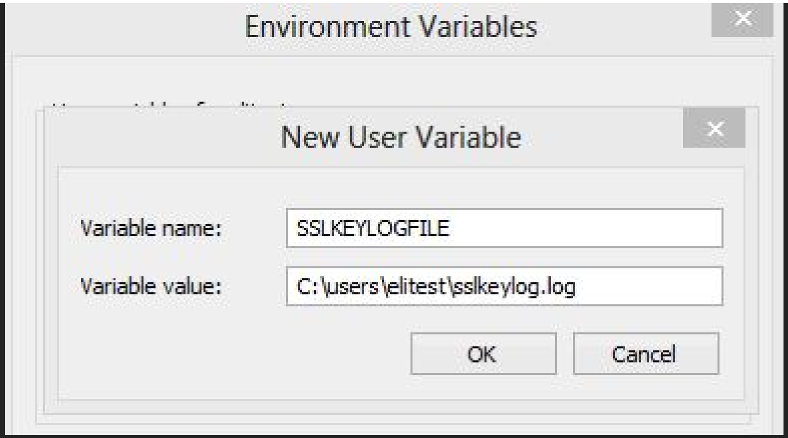
Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your sign-in

System startup, system failure, and debugging information

Product ID: 00178-50050-66116-AA322

Add a new user variable called “SSLKEYLOGFILE” and point it at the location that you want the log file to be located at.



Variable name: SSLKEYLOGFILE

Variable value: C:\users\elitest\sslkeylog.log

On Linux or Mac OS X:

```
$ export SSLKEYLOGFILE=~/path/to/sslkeylog.log
```

You can also add this to the last line of your

```
~/.bashrc
```

on Linux, or

```
~/.MacOSX/environment
```

on OS X so that it is set every time you log in.

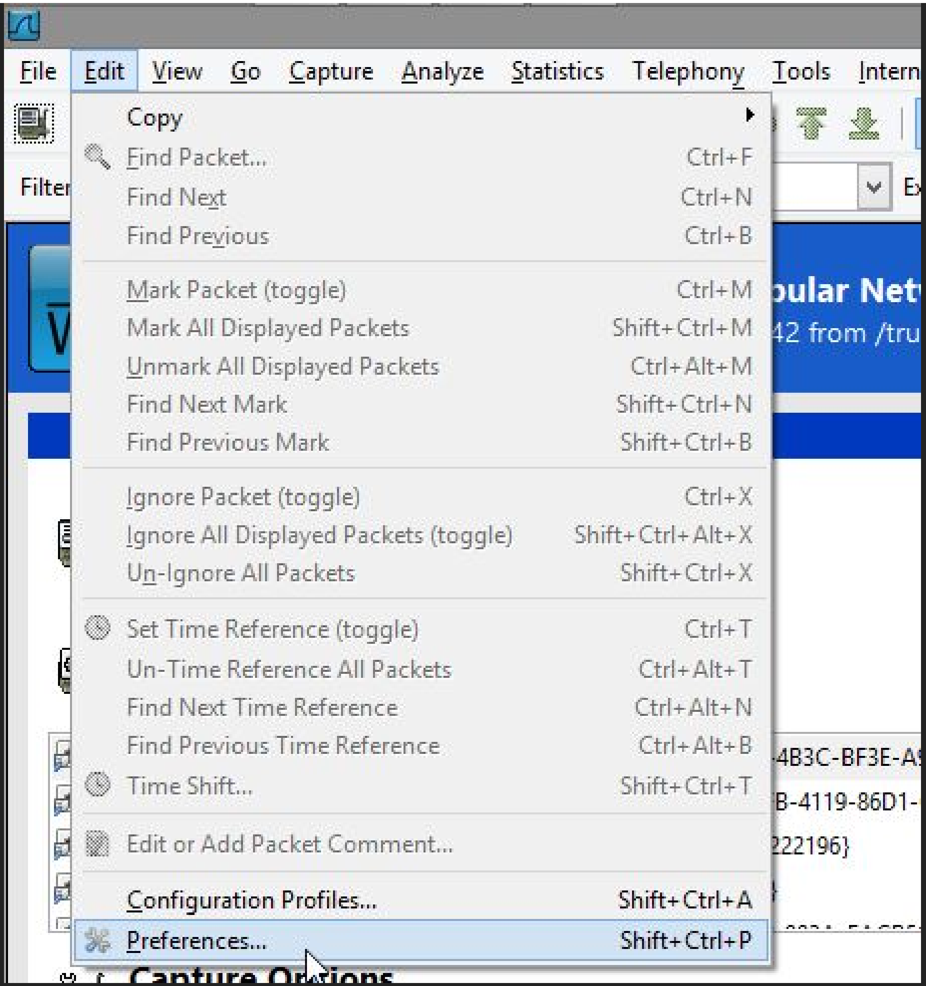
The next time that we launch Firefox or Chrome they will log your TLS keys to this file.

If you are having trouble getting it to work on OS X take a look at the comments below. It seems that Apple has changed how environmental variables work in recent versions of OS X. Try launching firefox and wireshark within the same terminal window with,

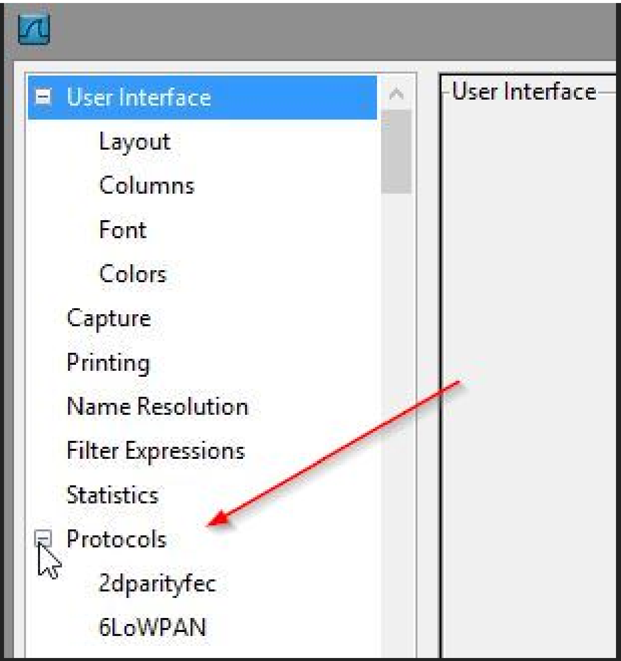
```
$ export SSLKEYLOGFILE=/Users/username/sslkeylogs/output.log
$ open -a firefox
$ wireshark
```

Setting up Wireshark

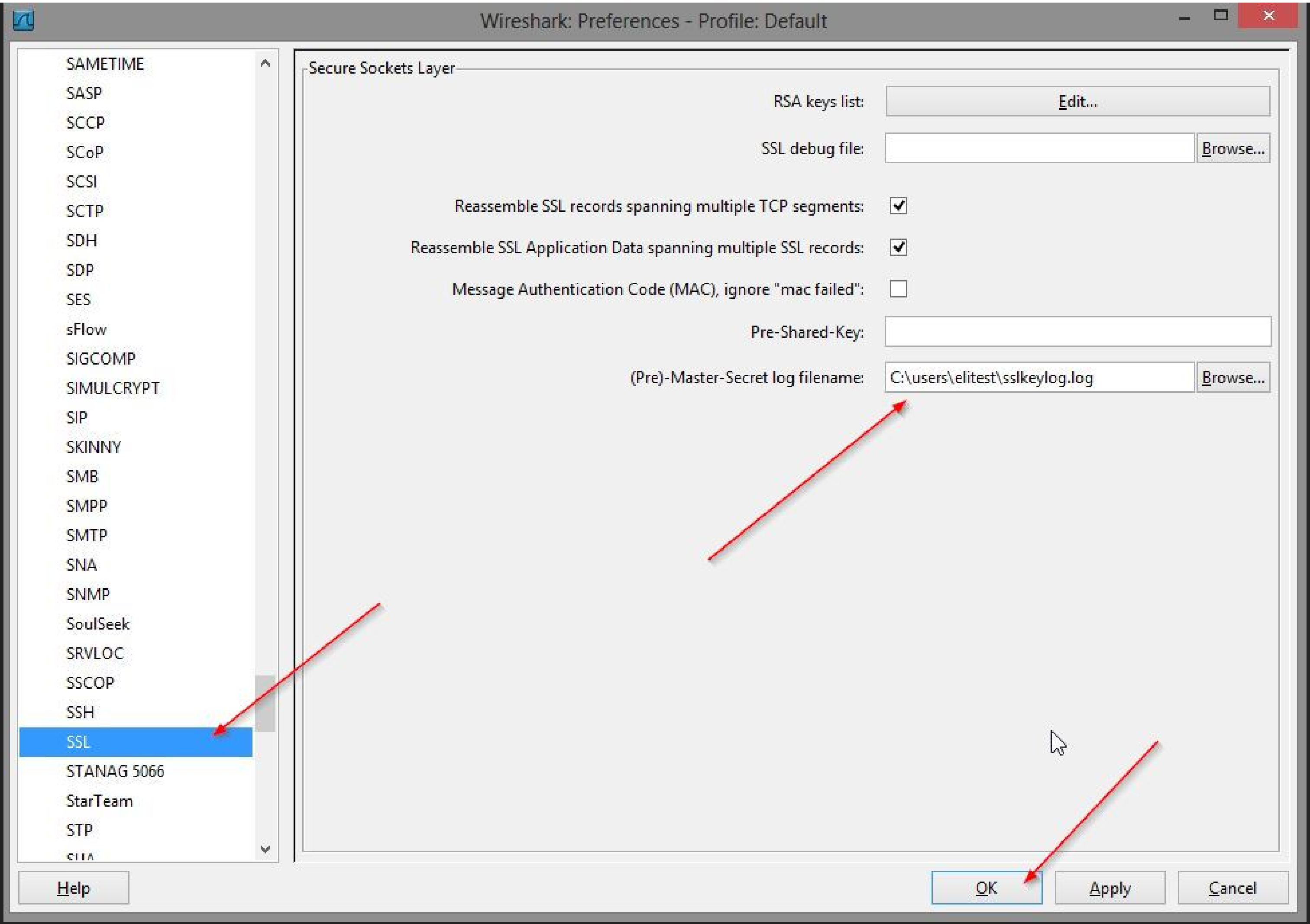
You need at least Wireshark 1.6 for this to work. We simply go into the preferences of Wireshark


File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Tools
Intern
Filter
Copy
Find Packet... Ctrl+F
Find Next Ctrl+N
Find Previous Ctrl+B
Mark Packet (toggle) Ctrl+M
Mark All Displayed Packets Shift+Ctrl+M
Unmark All Displayed Packets Ctrl+Alt+M
Find Next Mark Shift+Ctrl+N
Find Previous Mark Shift+Ctrl+B
Ignore Packet (toggle) Ctrl+X
Ignore All Displayed Packets (toggle) Shift+Ctrl+Alt+X
Un-Ignore All Packets Shift+Ctrl+X
Set Time Reference (toggle) Ctrl+T
Un-Time Reference All Packets Ctrl+Alt+T
Find Next Time Reference Ctrl+Alt+N
Find Previous Time Reference Ctrl+Alt+B
Time Shift... Shift+Ctrl+T
Edit or Add Packet Comment...
Configuration Profiles... Shift+Ctrl+A
Preferences... Shift+Ctrl+P
Capture Options

Expand the protocols section:



Browse to the location of your log file



Secure Sockets Layer

RSA keys list: Edit...

SSL debug file: Browse...

Reassemble SSL records spanning multiple TCP segments: ☑

Reassemble SSL Application Data spanning multiple SSL records: ☑

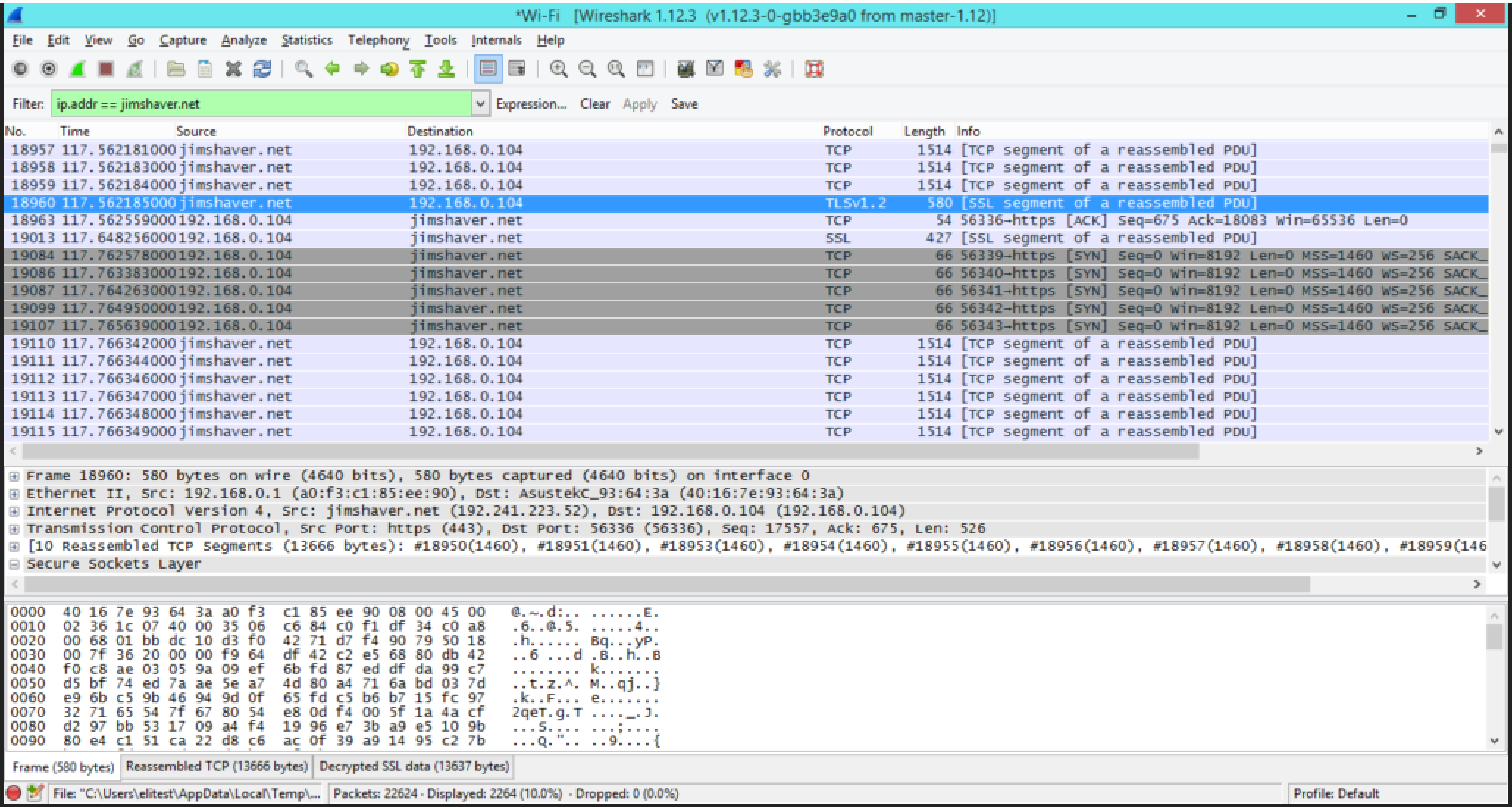
Message Authentication Code (MAC), ignore "mac failed": ☐

Pre-Shared-Key:

(Pre)-Master-Secret log filename: C:\users\elitest\sslkeylog.log Browse...

The Results

This is more along the lines of what we normally see when look at a TLS packet,



*Wi-Fi [Wireshark 1.12.3 (v1.12.3-0-gbb3e9a0 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: ip.addr == jimshaver.net

```
No.   Time        Source          Destination     Protocol  Length Info
18957 117.562181000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
18958 117.562183000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
18959 117.562184000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
18960 117.562185000jimshaver.net  192.168.0.104   TLSv1.2    580 [SSL segment of a reassembled PDU]
18963 117.562559000192.168.0.104  jimshaver.net   TCP         54 56336→https [ACK] Seq=675 Ack=18083 Win=65536 Len=0
19013 117.648256000192.168.0.104  jimshaver.net   SSL        427 [SSL segment of a reassembled PDU]
19084 117.762578000192.168.0.104  jimshaver.net   TCP         66 56339→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19086 117.763383000192.168.0.104  jimshaver.net   TCP         66 56340→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19087 117.764263000192.168.0.104  jimshaver.net   TCP         66 56341→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19099 117.764950000192.168.0.104  jimshaver.net   TCP         66 56342→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19107 117.765639000192.168.0.104  jimshaver.net   TCP         66 56343→https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_
19110 117.766342000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
19111 117.766344000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
19112 117.766346000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
19113 117.766347000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
19114 117.766348000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
19115 117.766349000jimshaver.net  192.168.0.104   TCP       1514 [TCP segment of a reassembled PDU]
```

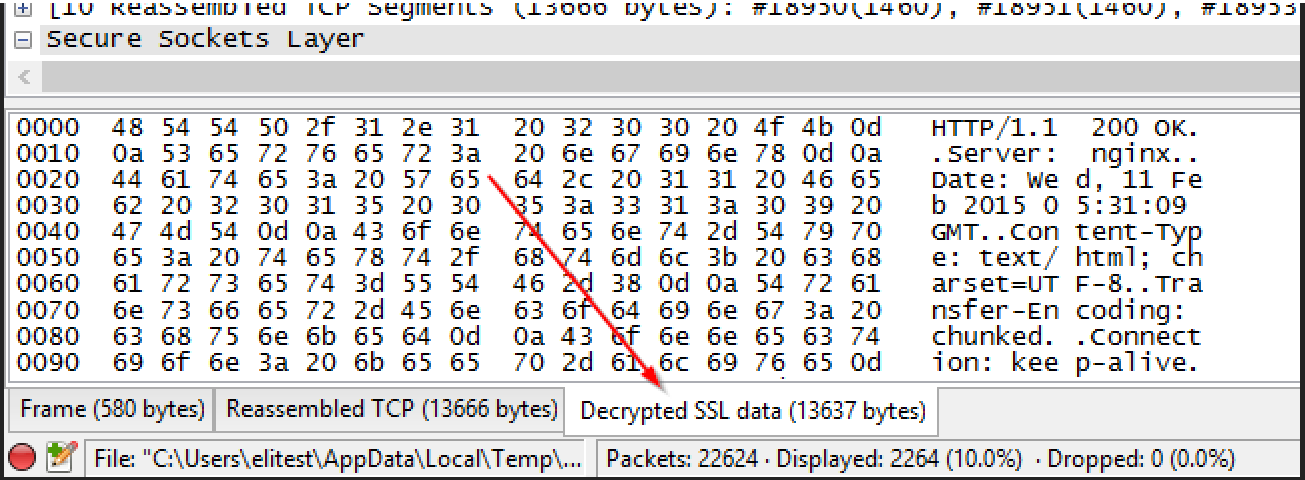
```
⊞ Frame 18960: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface 0
⊞ Ethernet II, Src: 192.168.0.1 (a0:f3:c1:85:ee:90), Dst: AsustekC_93:64:3a (40:16:7e:93:64:3a)
⊞ Internet Protocol Version 4, Src: jimshaver.net (192.241.223.52), Dst: 192.168.0.104 (192.168.0.104)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: 56336 (56336), Seq: 17557, Ack: 675, Len: 526
⊞ [10 Reassembled TCP Segments (13666 bytes): #18950(1460), #18951(1460), #18953(1460), #18954(1460), #18955(1460), #18956(1460), #18957(1460), #18958(1460), #18959(146
⊟ Secure Sockets Layer
```

```
0000  40 16 7e 93 64 3a a0 f3  c1 85 ee 90 08 00 45 00   @.~.d:.. ......E.
0010  02 36 1c 07 40 00 35 06  c6 84 c0 f1 df 34 c0 a8   .6..@.5. .....4..
0020  00 68 01 bb dc 10 d3 f0  42 71 d7 f4 90 79 50 18   .h...... Bq...yP.
0030  00 7f 36 20 00 00 f9 64  df 42 c2 e5 68 80 db 42   ..6 ...d .B..h..B
0040  f0 c8 ae 03 05 9a 09 ef  6b fd 87 ed df da 99 c7   ........ k.......
0050  d5 bf 74 ed 7a ae 5e a7  4d 80 a4 71 6a bd 03 7d   ..t.z.^. M..qj..}
0060  e9 6b c5 9b 46 94 9d 0f  65 fd c5 b6 b7 15 fc 97   .k..F... e.......
0070  32 71 65 54 7f 67 80 54  e8 0d f4 00 5f 1a 4a cf   2qeT.g.T ...._.J.
0080  d2 97 bb 53 17 09 a4 f4  19 96 e7 3b a9 e5 10 9b   ...S.... ...;....
0090  80 e4 c1 51 ca 22 d8 c6  ac 0f 39 a9 14 95 c2 7b   ...Q.".. ..9....{
```

Frame (580 bytes)
Reassembled TCP (13666 bytes)
Decrypted SSL data (13637 bytes)

File: "C:\Users\elitest\AppData\Local\Temp\... Packets: 22624 · Displayed: 2264 (10.0%) · Dropped: 0 (0.0%) Profile: Default

This is what it looks like when you switch to the “Decrypted SSL Data” tab. Note that we can now see the request information in plain-text! Success!



```
[10 Reassembled TCP Segments (13666 bytes): #18950(1460), #18951(1460), #18953
Secure Sockets Layer
```

```
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK.
0010  0a 53 65 72 76 65 72 3a  20 6e 67 69 6e 78 0d 0a   .Server:  nginx..
0020  44 61 74 65 3a 20 57 65  64 2c 20 31 31 20 46 65   Date: We d, 11 Fe
0030  62 20 32 30 31 35 20 30  35 3a 33 31 3a 30 39 20   b 2015 0 5:31:09
0040  47 4d 54 0d 0a 43 6f 6e  74 65 6e 74 2d 54 79 70   GMT..Con tent-Typ
0050  65 3a 20 74 65 78 74 2f  68 74 6d 6c 3b 20 63 68   e: text/ html; ch
0060  61 72 73 65 74 3d 55 54  46 2d 38 0d 0a 54 72 61   arset=UT F-8..Tra
0070  6e 73 66 65 72 2d 45 6e  63 6f 64 69 6e 67 3a 20   nsfer-En coding:
0080  63 68 75 6e 6b 65 64 0d  0a 43 6f 6e 6e 65 63 74   chunked. .Connect
0090  69 6f 6e 3a 20 6b 65 65  70 2d 61 6c 69 76 65 0d   ion: kee p-alive.
```

Frame (580 bytes)
Reassembled TCP (13666 bytes)
Decrypted SSL data (13637 bytes)
File: "C:\Users\elitest\AppData\Local\Temp\...
Packets: 22624 · Displayed: 2264 (10.0%) · Dropped: 0 (0.0%)

Conclusion

I hope you learned something today, this makes capturing TLS communication so much more straightforward. One of the nice things about this setup is that the client/server machine that generates the TLS traffic doesn't have to have Wireshark on it, so you don't have to gum up a clients machine with stuff they won't need, you can either have them dump the log to a network share or copy it off the machine and reunite it with the machine doing the packet capture later. Thanks for stopping by!

References:

Mozilla Wiki

Imperial Violet

jSSLKeyLog

0 0 0

Twitter Share Pin

ABOUT AUTHOR

elitest

COMMENTS

anonymous March 11, 2019 #1

On wireshark 3, moved from SSL protocol to TLS protocol

REPLY

chisight April 3, 2019 #2

On Debian and some Debian based Linuxs, the SSLKEYLOGFILE variable requires recompiling libnss to allow this to work with FireFox. Chromium works as is. ref: https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=842292

REPLY

ADD A COMMENT

Your email address will not be published. Required fields are marked *

Your comment

POST COMMENT

Your name *

E-mail address *

Website

□

Save my name, email, and website in this browser for the next time I comment.

□ Notify me of follow-up comments by email.

□ Notify me of new posts by email.