Wireshark Basics

Contents

- Traffic capture and traffic filtering with Wireshark
- SSL ManInTheMiddle with Wireshark
- WLAN traffic ManInTheMiddle with Wireshark

Wireshark

- Packet analyser / traffic sniffer
- Open-source
- Cross-platform
- Fancy GUI
- https://www.wireshark.org/

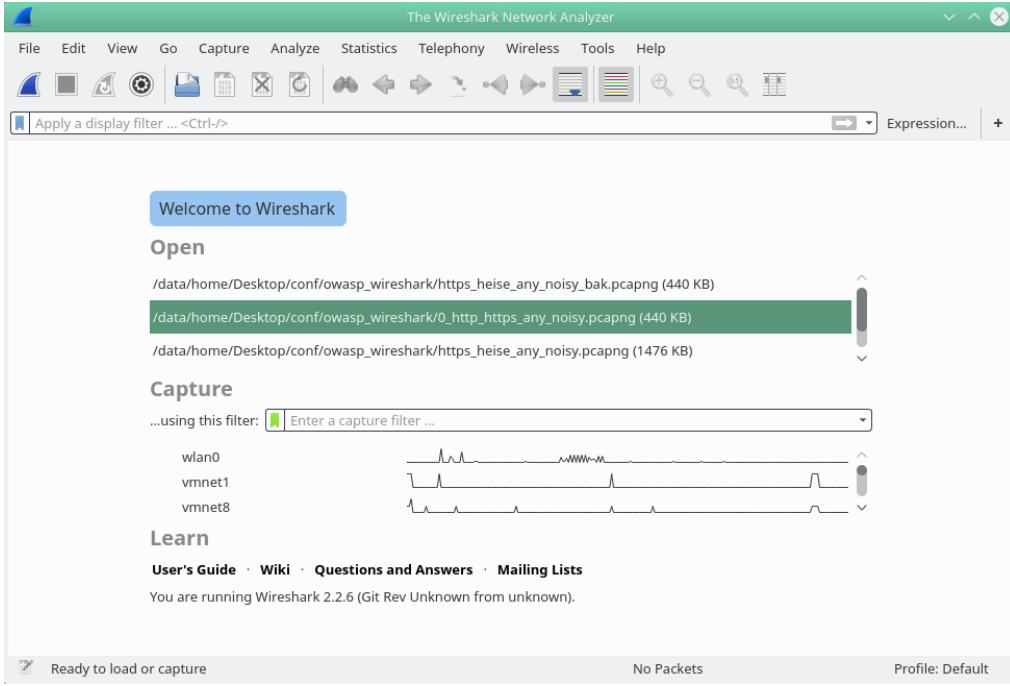
Data packets capturing

To start capturing

- Select a network interface
- Click on the blue shark fin button / press Ctrl + E

To stop capturing

- Click on the red stop button / press Ctrl + E



Data packets capturing

No.	Time	Source	Destination	Protocol	Lengt	Info
13	1.256128834	192.168.178.27	193.70.91.56	HTTP	183	GET / HTTP/1.1
14	1.272047640	193.70.91.56	192.168.178.27	TCP	68	80 → 55590 [ACK] Seq=1 Ack=116 Win=14592 Le
15	1.503106801	192.168.178.27	88.98.79.97	UDPENC…	45	NAT-keepalive
16	2.992854539	192.168.178.27	88.98.79.97	ESP	144	ESP (SPI=0xfc38d29f)
17	3.732746500	193.70.91.56	192.168.178.27	HTTP	345	HTTP/1.1 301 Moved Permanently
18	3.732785793	193.70.91.56	192.168.178.27	TCP	68	80 → 55590 [FIN, ACK] Seq=278 Ack=116 Win=1
19	3.732809890	192.168.178.27	193.70.91.56	TCP	68	55590 → 80 [ACK] Seq=116 Ack=278 Win=30336

```
> Frame 10: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 192.168.178.27, Dst: 193.70.91.56
> Transmission Control Protocol, Src Port: 55590, Dst Port: 80, Seq: 0, Len: 0
```

```
0000  00 04 00 01 00 06 b8 08  cf 58 61 64 00 00 08 00   ........ .Xad....
0010  45 00 00 3c c9 c5 40 00  40 06 e1 b3 c0 a8 b2 1b   E..<..@. @.......
0020  c1 46 5b 38 d9 26 00 50  e6 3e 0e f3 00 00 00 00   .F[8.&.P .>......
0030  a0 02 72 10 41 2c 00 00  02 04 05 b4 04 02 08 0a   ..r.A,.. ........
0040  61 f2 d4 e5 00 00 00 00  01 03 03 07              a....... ....
```

Data packets capturing

Top frame:
Number | Time | Source | Destination | Protocol | Length | Info

Data packets capturing

Top frame:
Number | Time | Source | Destination | Protocol | Length | Info

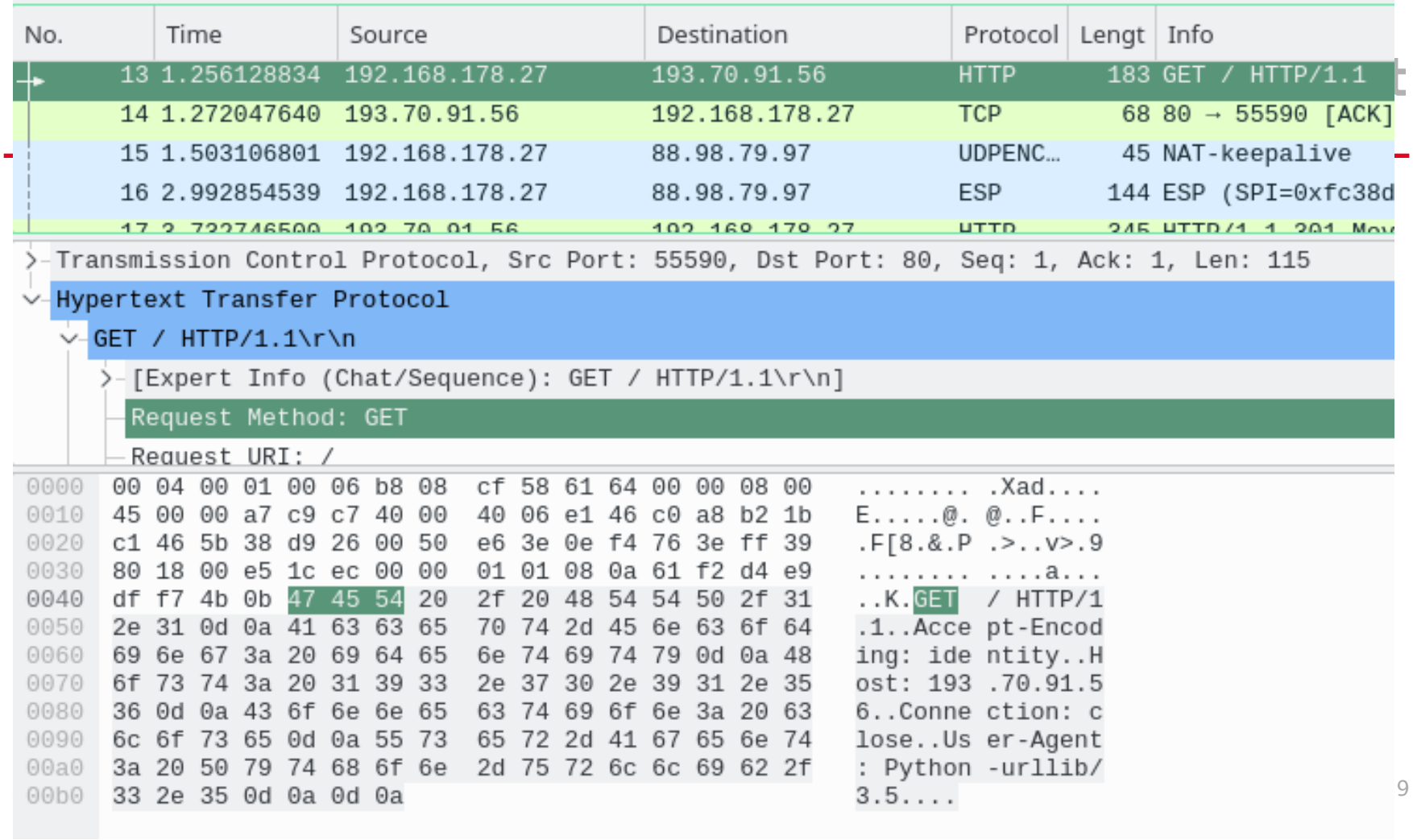
Middle frame example:
> Frame
> Linux cooked capture
> Internet protocol version, source, destination
> Transmission control protocol, src port, dst port, seq, len

Data packets capturing

Top frame:
Number | Time | Source | Destination | Protocol | Length | Info

Middle frame example:
> Frame
> Linux cooked capture
> Internet protocol version, source, destination
> Transmission control protocol, src port, dst port, seq, len

Bottom frame:
Data


No. Time Source Destination Protocol Lengt Info
13 1.256128834 192.168.178.27 193.70.91.56 HTTP 183 GET / HTTP/1.1
14 1.272047640 193.70.91.56 192.168.178.27 TCP 68 80 → 55590 [ACK]
15 1.503106801 192.168.178.27 88.98.79.97 UDPENC… 45 NAT-keepalive
16 2.992854539 192.168.178.27 88.98.79.97 ESP 144 ESP (SPI=0xfc38d
Transmission Control Protocol, Src Port: 55590, Dst Port: 80, Seq: 1, Ack: 1, Len: 115
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
[Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
Request Method: GET
0000 00 04 00 01 00 06 b8 08 cf 58 61 64 00 00 08 00 ..... .Xad....
0010 45 00 00 a7 c9 c7 40 00 40 06 e1 46 c0 a8 b2 1b E.....@. @..F....
0020 c1 46 5b 38 d9 26 00 50 e6 3e 0e f4 76 3e ff 39 .F[8.&.P .>..v>.9
0030 80 18 00 e5 1c ec 00 00 01 01 08 0a 61 f2 d4 e9 ....a...
0040 df f7 4b 0b 47 45 54 20 2f 20 48 54 54 50 2f 31 ..K.GET / HTTP/1
0050 2e 31 0d 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64 .1..Acce pt-Encod
0060 69 6e 67 3a 20 69 64 65 6e 74 69 74 79 0d 0a 48 ing: ide ntity..H
0070 6f 73 74 3a 20 31 39 33 2e 37 30 2e 39 31 2e 35 ost: 193 .70.91.5
0080 36 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6..Conne ction: c
0090 6c 6f 73 65 0d 0a 55 73 65 72 2d 41 67 65 6e 74 lose..Us er-Agent
00a0 3a 20 50 79 74 68 6f 6e 2d 75 72 6c 6c 69 62 2f : Python -urllib/
00b0 33 2e 35 0d 0a 0d 0a 3.5....

Wireshark Filters

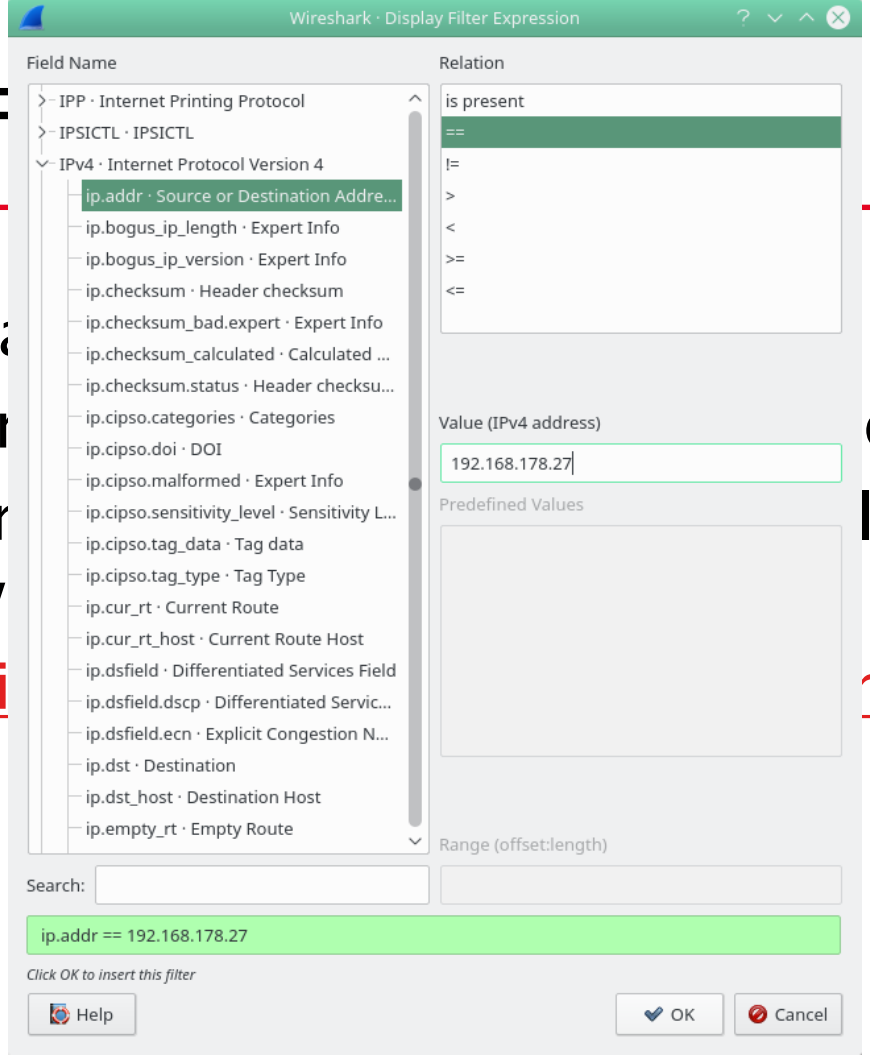
There are 2 ways to filter:

- Build a filter via the fancy GUI (Expression button)
- Type a filter into the “Apply a display filter” entry field (below the toolbar)
- https://wiki.wireshark.org/DisplayFilters

Wireshark F

There are 2 wa

- Build a filter on button)
- Type a filter lter" entry field (below
- https://wiki rs



Wireshark Filters Relations

English	C-like	Description and example
eq	==	Equal. `ip.src==10.0.0.5`
ne	!=	Not equal. `ip.src!=10.0.0.5`
gt	>	Greater than. `frame.len > 10`
lt	<	Less than. `frame.len < 128`
ge	>=	Greater than or equal to. `frame.len ge 0x100`
le	<=	Less than or equal to. `frame.len ⇐ 0x20`
contains		Protocol, field or slice contains a value. `sip.To contains "a1762"`
matches	~	Protocol or text field match Perl regualar expression. `http.host matches "acme\.(org|com|net)"`
bitwise_and	&	Compare bit field value. `tcp.flags & 0x02`

Source: https://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html

Wireshark Combining Expressions

English	C-like	Description and example
and	&&	Logical AND. `ip.src==10.0.0.5 and tcp.flags.fin`
or	||	Logical OR. `ip.scr==10.0.0.5 or ip.src==192.1.1.1`
xor	^^	Logical XOR. `tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29`
not	!	Logical NOT. `not llc`
[…]		Slice Operator. `eth.addr[0:3]==00:06:5B`
in		Membership Operator. `tcp.port in {80 443 8080}`

Source: https://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html
https://wiki.wireshark.org/DisplayFilters

Most common Wireshark filters

tcp.port eq 80
tcp.srcport==443

Filter for HTTP and HTTPS traffic:

tcp.port==443 or tcp.port==80
ssl or http
tcp.port in {80 443 8080}
tcp.port == 80 || tcp.port == 443 || tcp.port == 8080

Most common Wireshark filters

Filter for a protocol:

tcp

udp

dns

IP addresses:

ip.addr == 10.43.54.65

! (ip.addr == 10.43.54.65)

Most common Wireshark filters

Examples for web traffic:

http.request.uri == https://www.wireshark.org/

http.host matches "acme\.(org|com|net)"

http.response.code == 200

http.request.method == "GET“

tcp contains "admin"

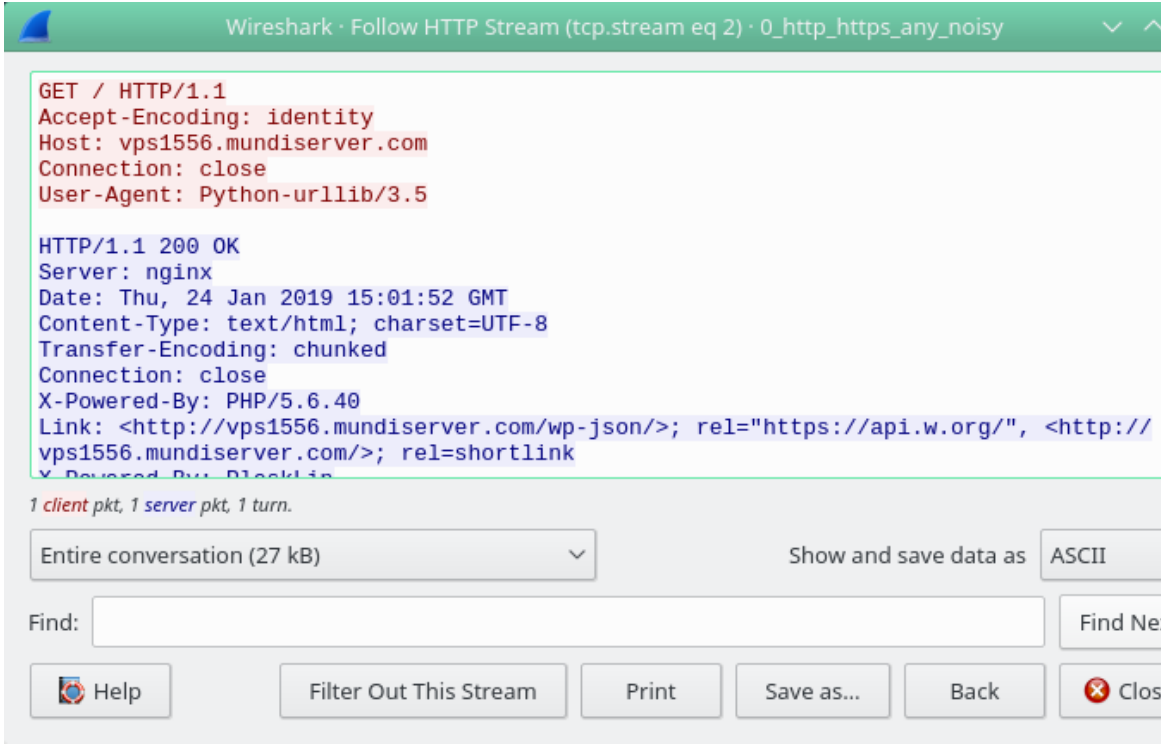
Wireshark filters logic

Filter for http traffic with specific addresses and frame time and not 200 response (e.g. you want to see 301 Moved permanently and 500 Server error packets):

```
http && ( (ip.dst == 192.168.178.27 ) || (ip.dst
== 193.70.91.56  ) ) && frame.time > "2019-01-24
00:01:00.0000" && frame.time < "2019-01-25
15:01:53.0000" && http.response.code !=  200
```

Follow the stream

Select a Packet > Right mouse click > Follow > HTTP Stream



What if I told you…

That you can sort the traffic just by clicking the colomn names…

That you can search for strings in packets using Edit > Find Packet… (Ctrl+F)

Contents

- SSL ManInTheMiddle with Wireshark (Linux edition)

SSL ManInTheMiddle with Wireshark

To test the decryption of SSL traffic with Wireshark:

- Create private keys of the server and the client
- Start a server which uses the certificate with the key and send some test packets
- Configure Wireshark

Create certificates

Create a server certificate

```
# openssl req -new -x509 -out server.crt -nodes
-keyout server.pem -subj /CN=localhost
```

Create a client certificate

```
# openssl req -new -x509 -nodes -out client.crt
-keyout client.key -subj /CN=Moi/O=Foo/C=NL
```

Start a server

Start a server at localhost:4443

```
# openssl s_server -cipher AES256-SHA -accept 4443 -
www -CAfile client.crt -verify 1 -key server.pem -
cert server.crt
```

```
s_server -cipher AES256-SHA -accept 4443 -www -CAfile client.crt -verify 1 -key server.pem -cert server.crt
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:AES256-SHA
---
Ciphers common between both SSL end points:
ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES128-SHA       ECDHE-RSA-AES256-SHA
AES128-GCM-SHA256           AES256-GCM-SHA384          AES128-SHA
AES256-SHA                  DES-CBC3-SHA
Signature Algorithms: ECDSA+SHA256:0x04+0x08:RSA+SHA256:ECDSA+SHA384:0x05+0x08:RSA+SHA384:0x06+0x08:RSA+SHA512:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:RSA+SHA256:ECDSA+SHA384:RSA+SHA384:RSA+SHA512:RSA+SHA1
Supported Elliptic Curves: 0x8A8A:0x001D:P-256:P-384
Shared Elliptic curves: P-256:P-384
---
New, TLSv1/SSLv3, Cipher is AES256-SHA
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : AES256-SHA
    Session-ID:
    Session-ID-ctx: 01000000
    Master-Key: C068D52572ABA77965042CA2E3E4BABA0474F7C45DE563C1226DFC2201AFEDF55BBF500A68FF48D260EE9DCB32BA59CA
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1548434883
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
   0 items in the session cache
   0 client connects (SSL_connect())
```

Send a request with python(3) and stop the capture

```
import urllib.request
import ssl

context = ssl._create_unverified_context()
with
urllib.request.urlopen("https://localhost:4443/",
context=context) as url:
    s = url.read()
    print(s)
```

Traffic captured

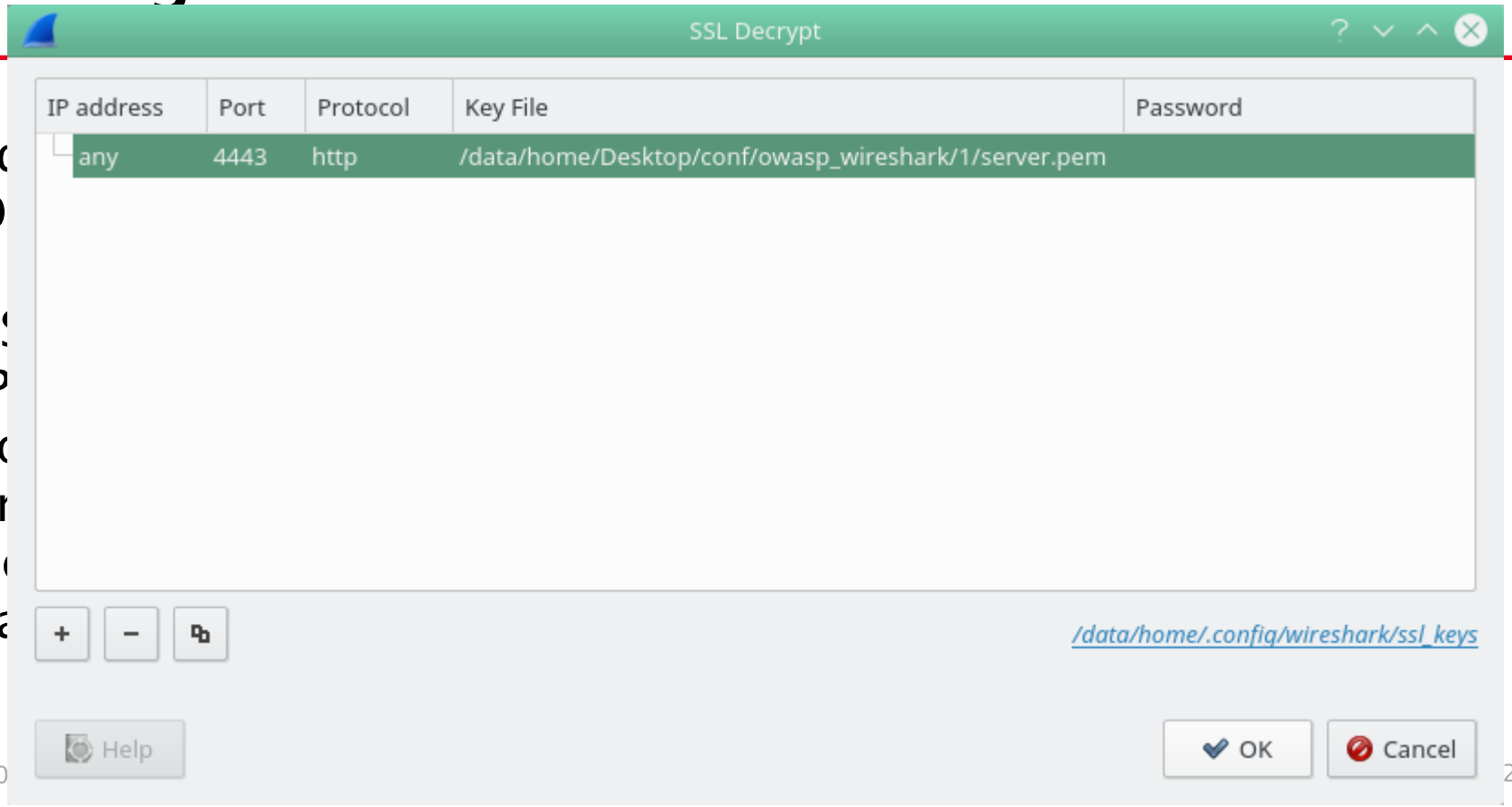
No.	Time	Source	Destination	Protocol	Lengt	Info
22	5.110731826	IntelCor_58:61:64		ARP	44	192.168.178.27 is at b8:08:cf:58:61:64
23	5.362390056	127.0.0.1	127.0.0.1	TCP	76	56052 → 4443 [SYN] Seq=0 Win=43690 Len=0 MSS
24	5.362398184	127.0.0.1	127.0.0.1	TCP	76	4443 → 56052 [SYN, ACK] Seq=0 Ack=1 Win=4369
25	5.362405598	127.0.0.1	127.0.0.1	TCP	68	56052 → 4443 [ACK] Seq=1 Ack=1 Win=43776 Len
26	5.362506833	127.0.0.1	127.0.0.1	TLSv1.2	585	Client Hello
27	5.362518970	127.0.0.1	127.0.0.1	TCP	68	4443 → 56052 [ACK] Seq=1 Ack=518 Win=44800 L
28	5.362599242	127.0.0.1	127.0.0.1	TLSv1.2	1009	Server Hello, Certificate, Certificate Reque
29	5.362606016	127.0.0.1	127.0.0.1	TCP	68	56052 → 4443 [ACK] Seq=518 Ack=942 Win=45696
30	5.362841637	127.0.0.1	127.0.0.1	TLSv1.2	422	Certificate, Client Key Exchange, Change Cip
31	5.363809929	127.0.0.1	127.0.0.1	TLSv1.2	334	New Session Ticket, Change Cipher Spec, Fini
32	5.363964990	127.0.0.1	127.0.0.1	HTTP	233	GET / HTTP/1.1
33	5.364140080	127.0.0.1	127.0.0.1	TLSv1.2	3993	[SSL segment of a reassembled PDU]

Configure wireshark

Edit > Preferences
On the left: Protocols > SSL

RSA keys list: press „Edit...“ and add via „+“
IP address – any
Port – 4443
Protocol – http
Key file – /.../server.pem
Password –

Configure wireshark



Configure wireshark

SSL debug file (file with decrypted output):
/.../wiresharklog.txt

Check „Reassemble SSL records spanning multiple TCP segments“
Check „Reassemble SSL Application Data spanning multiple SSL records“
Don’t check “Message Authentication Code (MAC), ignore “mac failed”

Pre-Shared-Key (left empty):

(Pre)-Master-Secret log filename:
/.../client.key

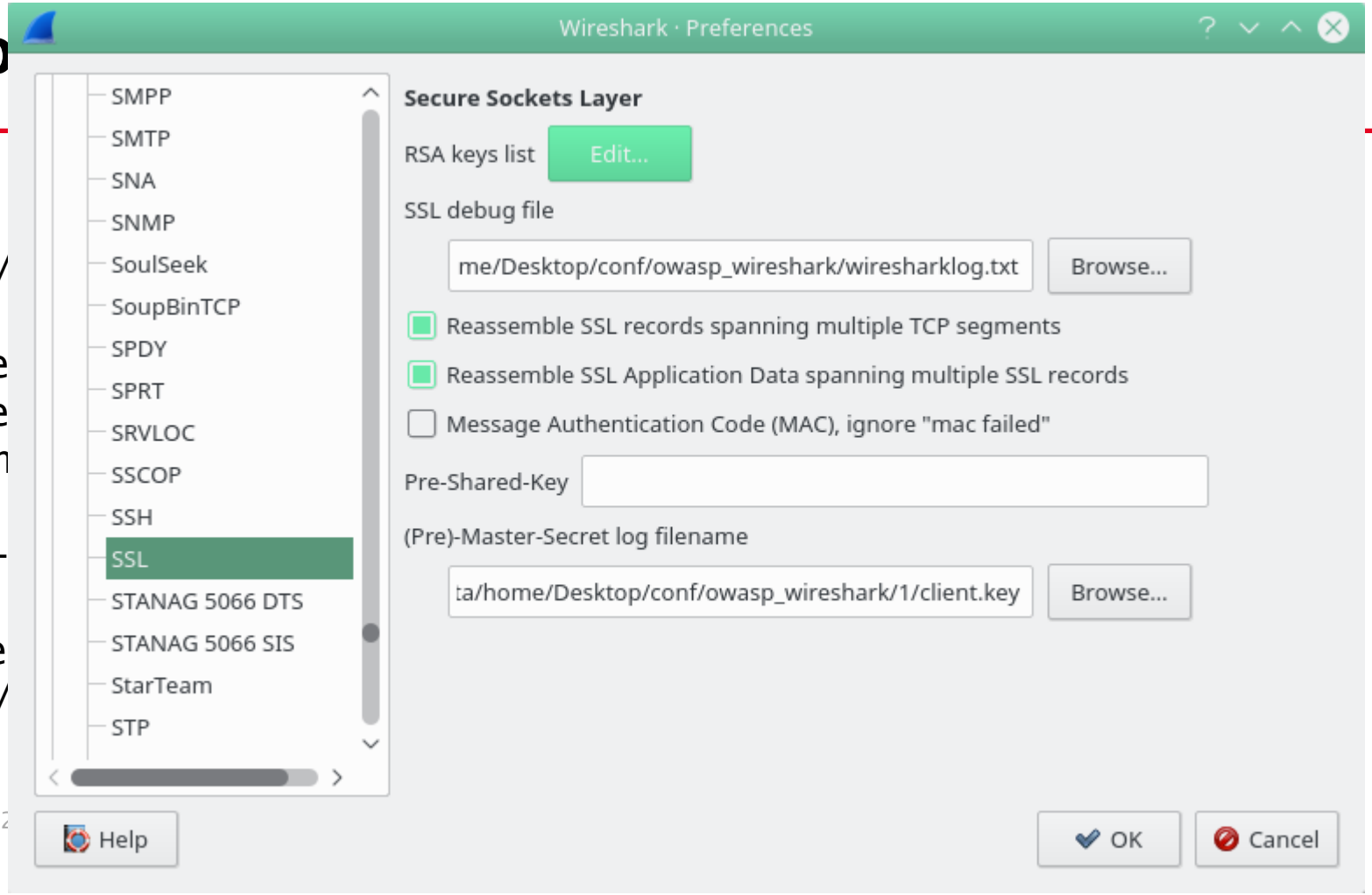
Co

SSL
/…/

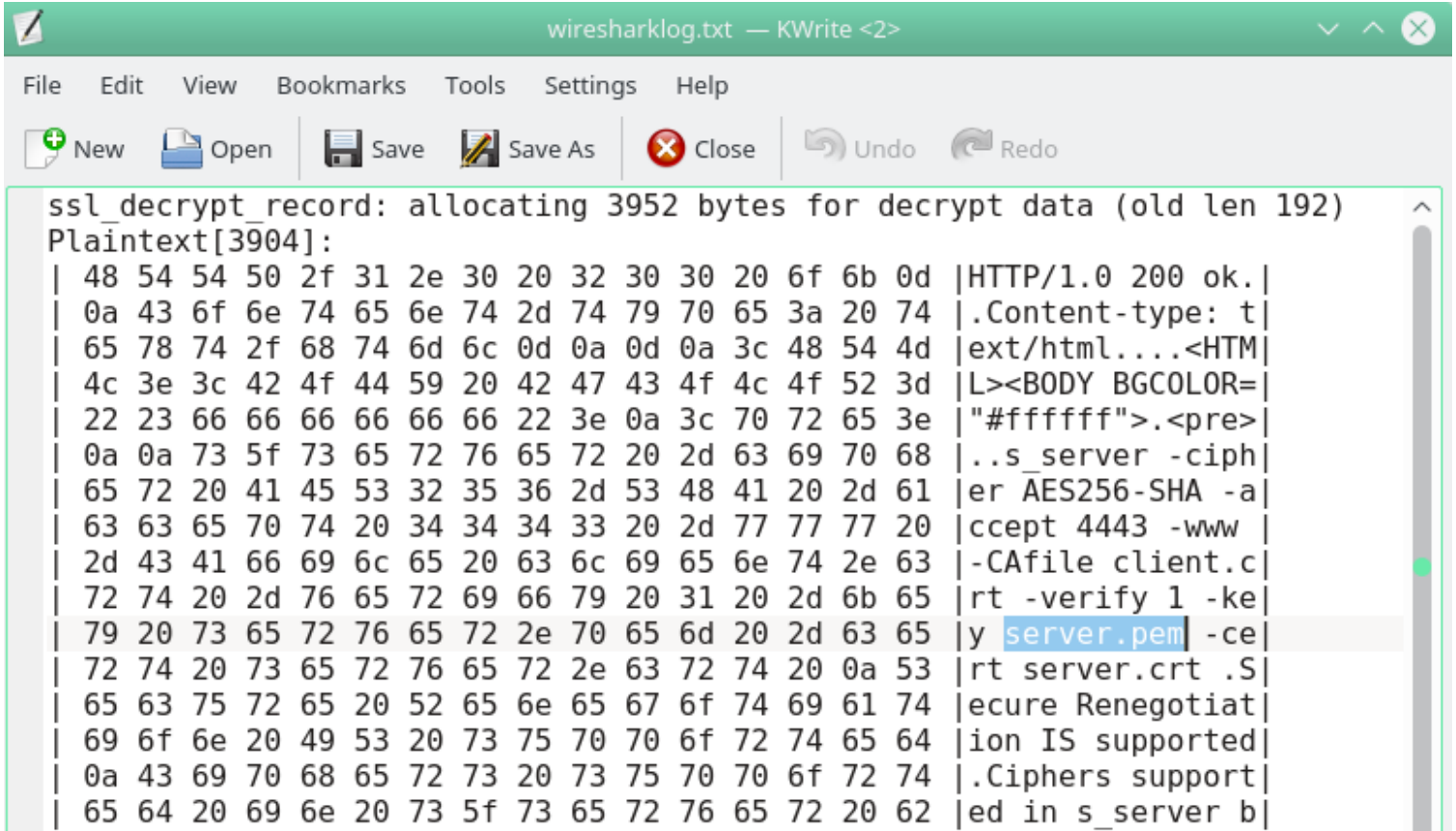
Che
Che
Don

Pre-

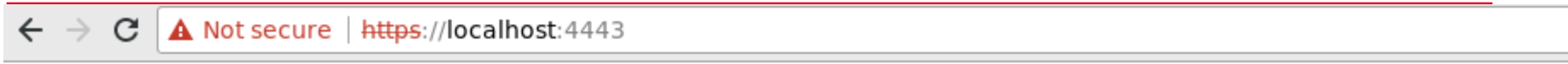
(Pre
/…/



Enjoy the decryption



Enjoy the decryption (proof)




```
s_server -cipher AES256-SHA -accept 4443 -www -CAfile client.crt -verify 1 -key server.pem -cert server.crt
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:AES256-SHA
---
Ciphers common between both SSL end points:
ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES128-SHA       ECDHE-RSA-AES256-SHA
AES128-GCM-SHA256           AES256-GCM-SHA384          AES128-SHA
AES256-SHA                  DES-CBC3-SHA
Signature Algorithms: ECDSA+SHA256:0x04+0x08:RSA+SHA256:ECDSA+SHA384:0x05+0x08:RSA+SHA384:0x06+0x08:RSA+SHA512:RSA+SH
Shared Signature Algorithms: ECDSA+SHA256:RSA+SHA256:ECDSA+SHA384:RSA+SHA384:RSA+SHA512:RSA+SHA1
Supported Elliptic Curves: 0x8A8A:0x001D:P-256:P-384
Shared Elliptic curves: P-256:P-384
---
New, TLSv1/SSLv3, Cipher is AES256-SHA
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : AES256-SHA
    Session-ID:
    Session-ID-ctx: 01000000
    Master-Key: C068D52572ABA77965042CA2E3E4BABA0474F7C45DE563C1226DFC2201AFEDF55BBF500A68FF48D260EE9DCB32BA59CA
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
```

SSL ManInTheMiddle (the easy way)

Set the `SSLKEYLOGFILE` environment variable and enter it's value under `(Pre)-Master-Secret log filename.`

https://jimshaver.net/2015/02/11/decrypting-tls-browser-traffic-with-wireshark-the-easy-way/

Bedtime stories

https://wiki.wireshark.org/SSL

https://www.cellstream.com/reference-reading/tipsandtricks/354-wireshark-ssltls-decryption

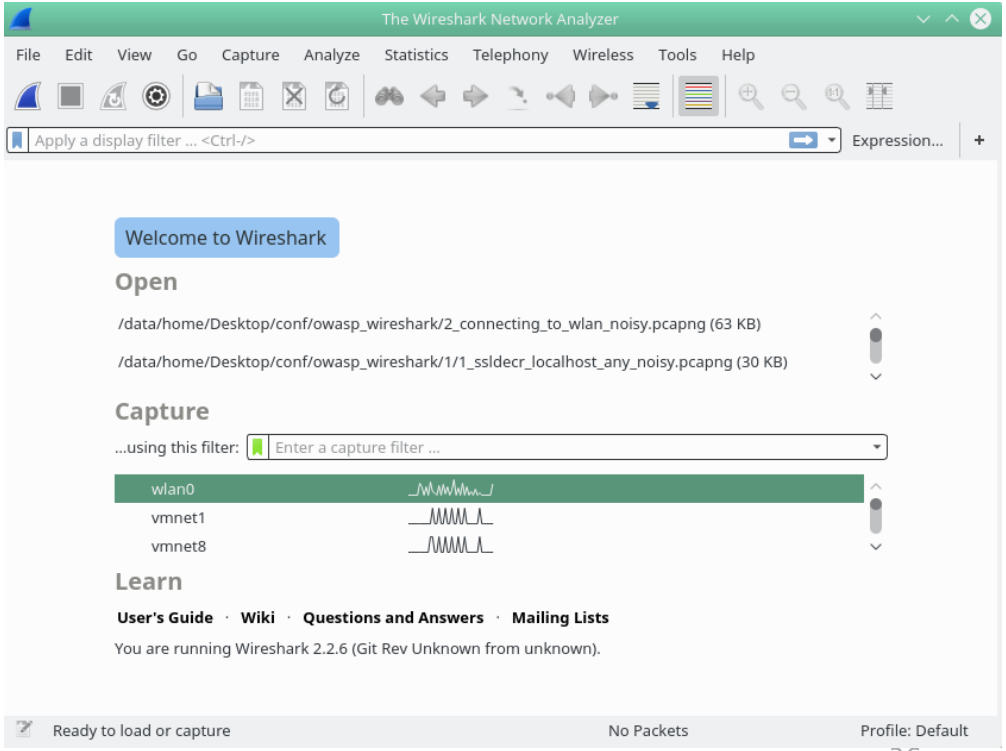
Contents

- WLAN traffic ManInTheMiddle with Wireshark

Data packets capturing

To start capturing

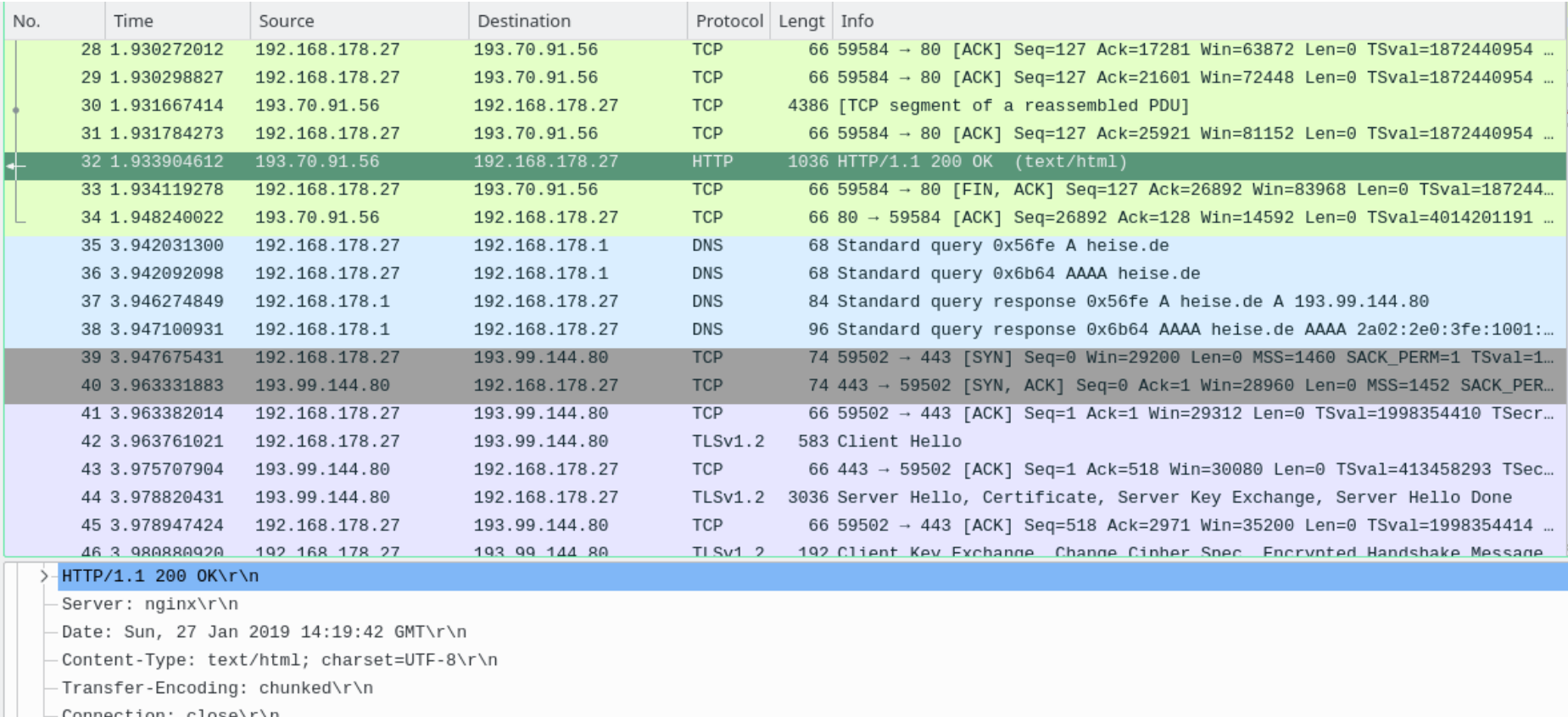
- Select the **WLAN** network interface
- Click on the blue shark fin button / press Ctrl + E



Example: Establishing a WLAN connection

No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000000	Avm_64:00:8f	IntelCor_58:61:64	EAPOL	113	Key (Message 1 of 4)
2	0.001357379	IntelCor_58:61:64	Avm_64:00:8f	EAPOL	135	Key (Message 2 of 4)
3	0.011014769	Avm_64:00:8f	IntelCor_58:61:64	EAPOL	185	Key (Message 3 of 4)
4	0.011116914	IntelCor_58:61:64	Avm_64:00:8f	EAPOL	113	Key (Message 4 of 4)
5	0.031731507	::	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
6	0.047907511	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0x6755
7	0.055516718	192.168.178.1	192.168.178.27	DHCP	590	DHCP ACK - Transaction ID 0x6755
8	0.125767086	IntelCor_58:61:64	Broadcast	ARP	42	Who has 192.168.178.1? Tell 192.168.1
9	0.127793346	Avm_64:00:8d	IntelCor_58:61:64	ARP	42	192.168.178.1 is at 9c:c7:a6:64:00:8d
10	0.127806760	192.168.178.27	192.168.178.1	DNS	74	Standard query 0xeec5 A ntp.ubuntu.co
11	0.127818827	192.168.178.27	192.168.178.1	DNS	74	Standard query 0x78d8 AAAA ntp.ubuntu
12	0.131610265	192.168.178.1	192.168.178.27	DNS	138	Standard query response 0xeec5 A ntp.
13	0.132247067	192.168.178.1	192.168.178.27	DNS	130	Standard query response 0x78d8 AAAA n

Example: HTTP/HTTPS traffic capture on wlan0 interface



No.	Time	Source	Destination	Protocol	Lengt	Info
28	1.930272012	192.168.178.27	193.70.91.56	TCP	66	59584 → 80 [ACK] Seq=127 Ack=17281 Win=63872 Len=0 TSval=1872440954 …
29	1.930298827	192.168.178.27	193.70.91.56	TCP	66	59584 → 80 [ACK] Seq=127 Ack=21601 Win=72448 Len=0 TSval=1872440954 …
30	1.931667414	193.70.91.56	192.168.178.27	TCP	4386	[TCP segment of a reassembled PDU]
31	1.931784273	192.168.178.27	193.70.91.56	TCP	66	59584 → 80 [ACK] Seq=127 Ack=25921 Win=81152 Len=0 TSval=1872440954 …
32	1.933904612	193.70.91.56	192.168.178.27	HTTP	1036	HTTP/1.1 200 OK (text/html)
33	1.934119278	192.168.178.27	193.70.91.56	TCP	66	59584 → 80 [FIN, ACK] Seq=127 Ack=26892 Win=83968 Len=0 TSval=187244…
34	1.948240022	193.70.91.56	192.168.178.27	TCP	66	80 → 59584 [ACK] Seq=26892 Ack=128 Win=14592 Len=0 TSval=4014201191 …
35	3.942031300	192.168.178.27	192.168.178.1	DNS	68	Standard query 0x56fe A heise.de
36	3.942092098	192.168.178.27	192.168.178.1	DNS	68	Standard query 0x6b64 AAAA heise.de
37	3.946274849	192.168.178.1	192.168.178.27	DNS	84	Standard query response 0x56fe A heise.de A 193.99.144.80
38	3.947100931	192.168.178.1	192.168.178.27	DNS	96	Standard query response 0x6b64 AAAA heise.de AAAA 2a02:2e0:3fe:1001:…
39	3.947675431	192.168.178.27	193.99.144.80	TCP	74	59502 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1…
40	3.963331883	193.99.144.80	192.168.178.27	TCP	74	443 → 59502 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1452 SACK_PER…
41	3.963382014	192.168.178.27	193.99.144.80	TCP	66	59502 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1998354410 TSecr…
42	3.963761021	192.168.178.27	193.99.144.80	TLSv1.2	583	Client Hello
43	3.975707904	193.99.144.80	192.168.178.27	TCP	66	443 → 59502 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=413458293 TSec…
44	3.978820431	193.99.144.80	192.168.178.27	TLSv1.2	3036	Server Hello, Certificate, Server Key Exchange, Server Hello Done
45	3.978947424	192.168.178.27	193.99.144.80	TCP	66	59502 → 443 [ACK] Seq=518 Ack=2971 Win=35200 Len=0 TSval=1998354414 …
46	3.980880920	192.168.178.27	193.99.144.80	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

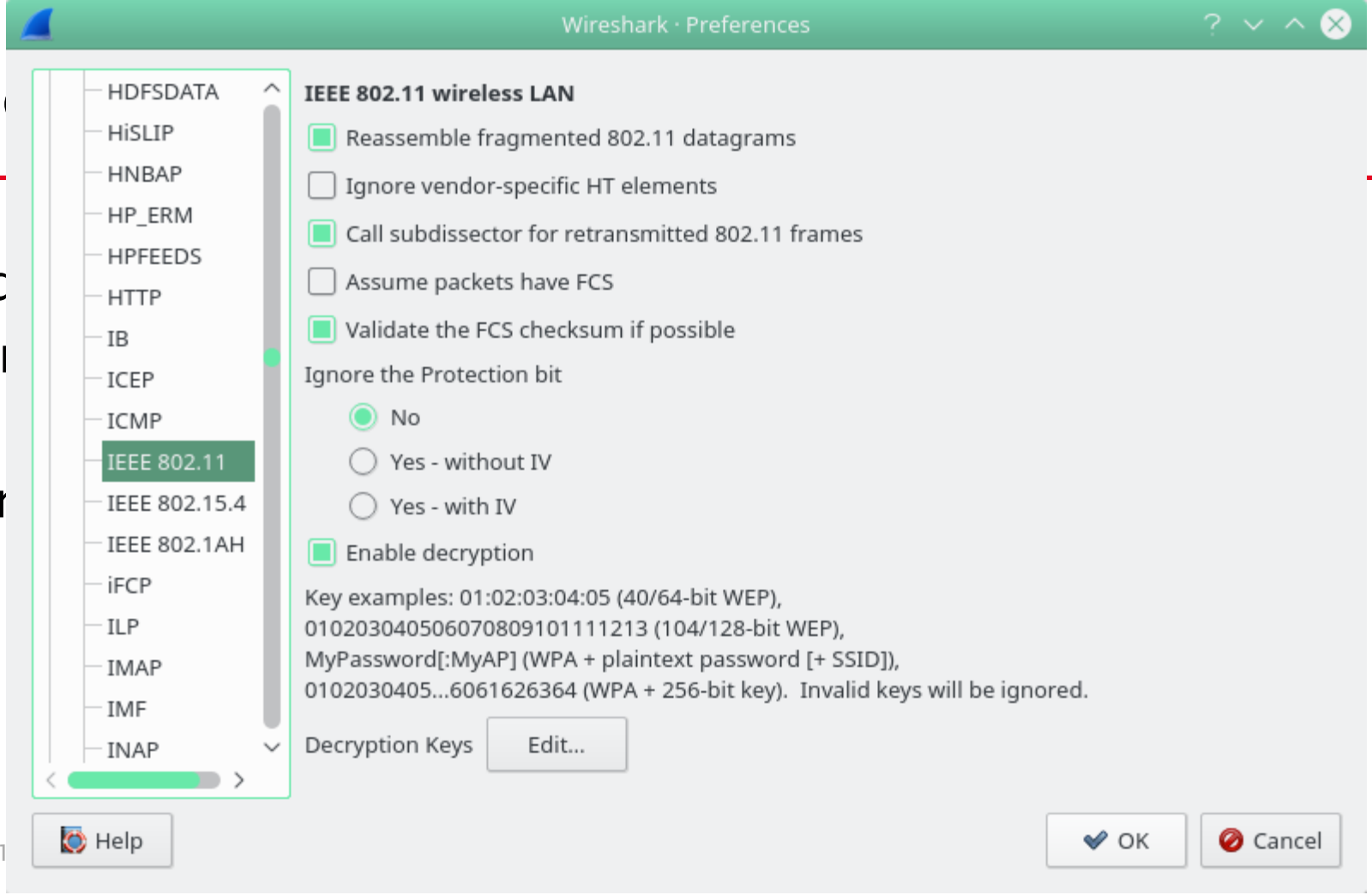
```
HTTP/1.1 200 OK\r\n
Server: nginx\r\n
Date: Sun, 27 Jan 2019 14:19:42 GMT\r\n
Content-Type: text/html; charset=UTF-8\r\n
Transfer-Encoding: chunked\r\n
Connection: close\r\n
```

Decrypt traffic with a known key

Edit > Preferences

On the left: Protocols > IEEE 802.11

And add a decryption key


Wireshark · Preferences
HDFSDATA
HiSLIP
HNBAP
HP_ERM
HPFEEDS
HTTP
IB
ICEP
ICMP
IEEE 802.11
IEEE 802.15.4
IEEE 802.1AH
iFCP
ILP
IMAP
IMF
INAP
IEEE 802.11 wireless LAN
Reassemble fragmented 802.11 datagrams
Ignore vendor-specific HT elements
Call subdissector for retransmitted 802.11 frames
Assume packets have FCS
Validate the FCS checksum if possible
Ignore the Protection bit
No
Yes - without IV
Yes - with IV
Enable decryption
Key examples: 01:02:03:04:05 (40/64-bit WEP),
0102030405060708091011121 3 (104/128-bit WEP),
MyPassword[:MyAP] (WPA + plaintext password [+ SSID]),
0102030405...6061626364 (WPA + 256-bit key). Invalid keys will be ignored.
Decryption Keys
Edit...
Help
OK
Cancel

Further reading

Aircrack-ng
https://www.aircrack-ng.org/
https://tools.kali.org/wireless-attacks/aircrack-ng

EAPOL
https://security.stackexchange.com/questions/66008/how-exactly-does-4-way-handshake-cracking-work

Wireshark Advanced

Contents

- Wireshark parsers (dissectors)

Wireshark dissectors

Disscectors are parsers/custom scripts to analize packets' data.

Can be implemented

- In Lua language
- In C language

Wireshark Lua dissectors

helloworld.lua (saved under /usr/lib/x86_64-linux-gnu/wireshark/plugins/2.2.6/test/helloworld.lua):

```
local splash = TextWindow.new("Hello World!");
```

Wireshark Lua dissectors

The dissector will be executed on Wireshark's start. The script has to be saved in the Plugin directory in this case, e.g.

```
# locate wireshark | grep -iE 'plugins'
/usr/lib/x86_64-linux-gnu/wireshark/plugins/2.2.6
```

Alternatively, you can enforce the execution of a Lua dissector by running the dofile command under Tools > Lua > Evaluate

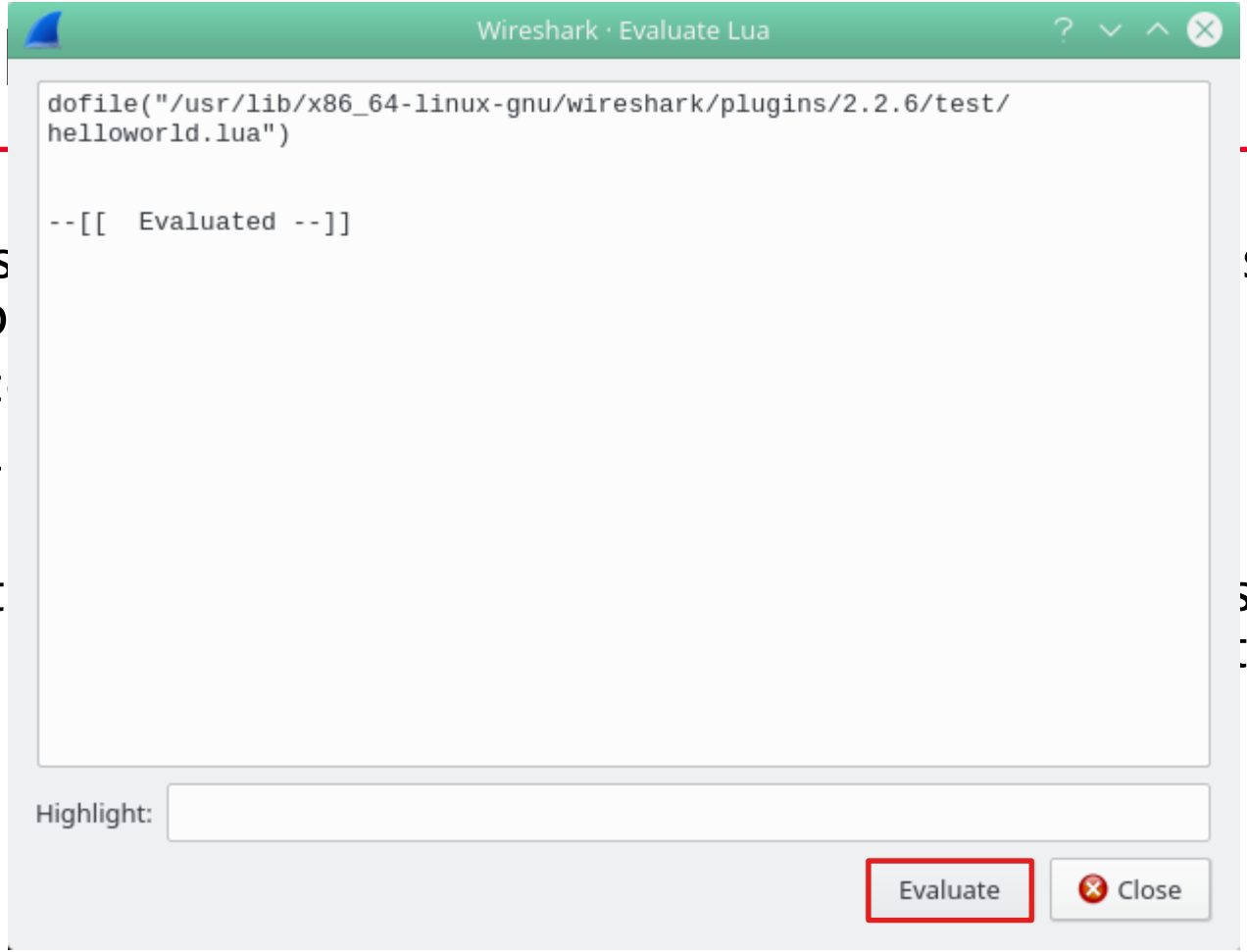
```
dofile("path/to/file.lua")
```

Wires

The diss
has to b
locat
/usr/li

Alternat
running

dofile(



script

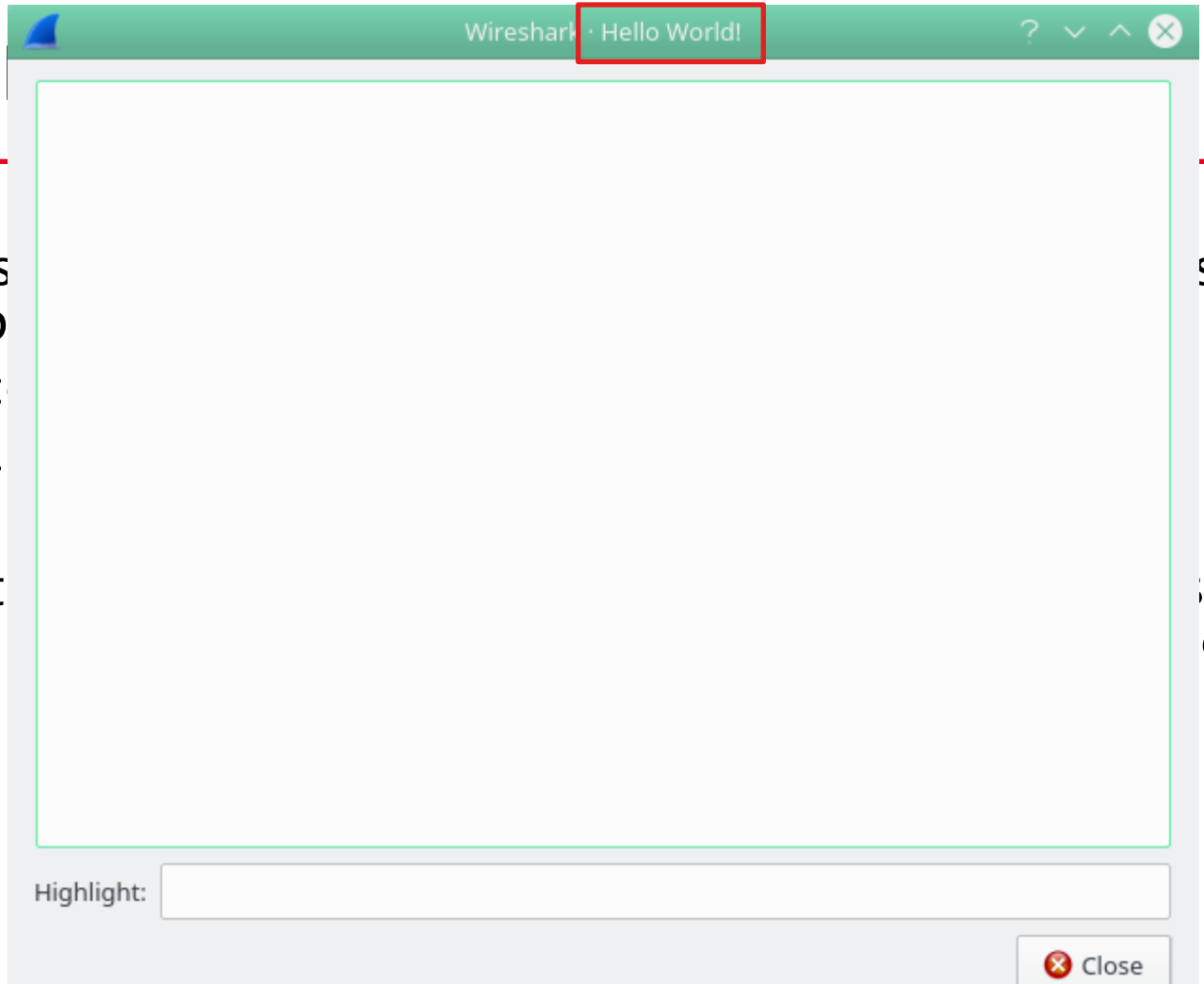
sector by
te

Wires

The diss script
has to b
```
# locat
/usr/li
```

Alternat ector by
running e

```
dofile(
```



Lua basics

- Is a multi-paradigm language (supports procedural style, functional programming, has some object-oriented programming features)
- dynamically typed
- supports atomic data structures such as
  - boolean values,
  - numbers (double-precision floating point and 64-bit integers by default),
  - strings,
  - tables (for arrays/sets/lists)

Lua basics

- -- means comment
- Not equal in conditionals is ~=
- Loops: while, repeat until (similar to a do while loop), for (numeric), for (generic).
- Use i = i + 1 instead of ++ or +=
- nil for null

Lua basics

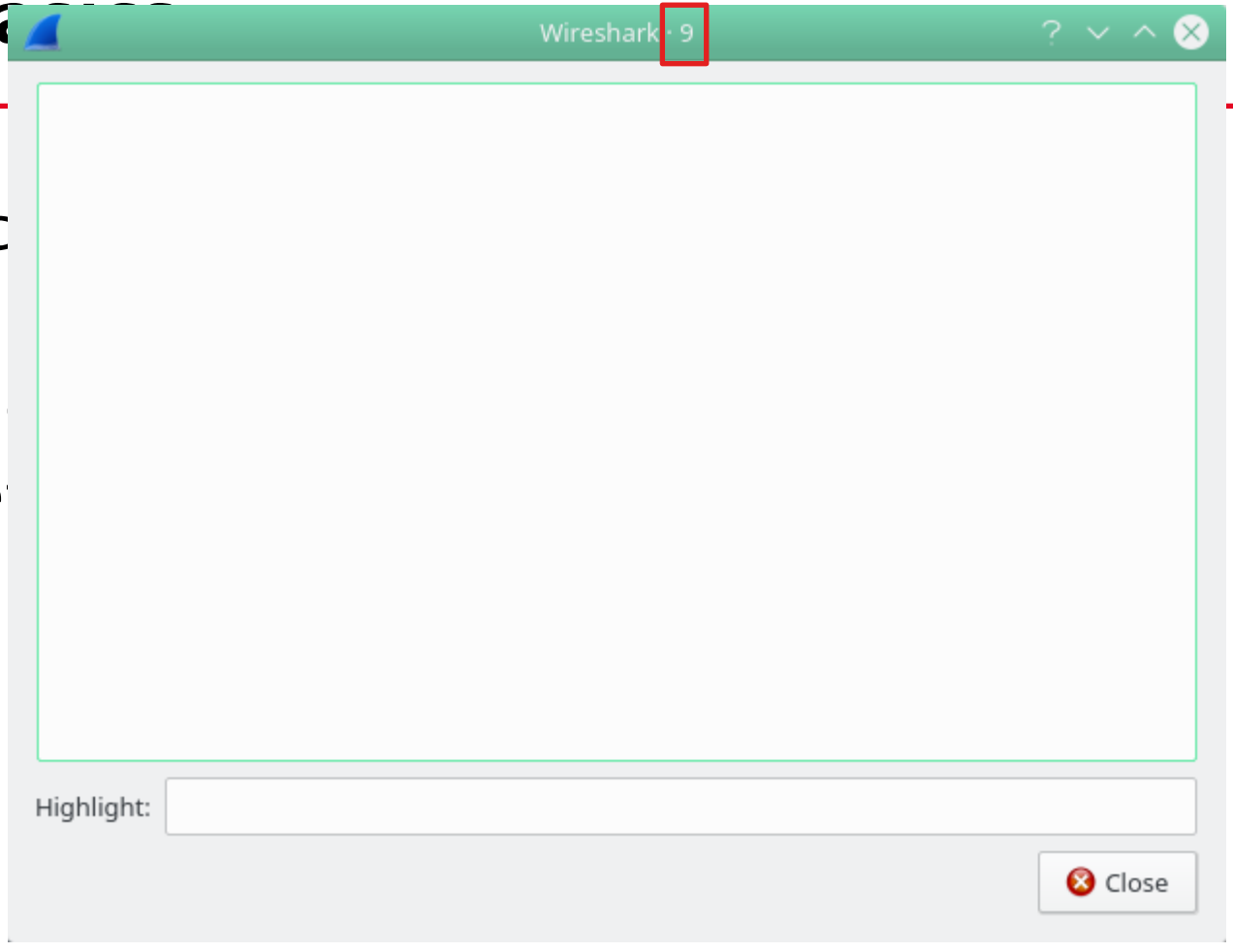
- Function example

```
function add(x, y)
    return x + y
end
local splash = TextWindow.new(add(3,6));
```

Lua ba

- Func

```
functi
    re
end
local
```



Lua basics (function example #2)

```lua
function addto(x)
  -- Return a new function that adds x to the argument
  return function(y)
    --[=[ When we refer to the variable x, which is outside the current
         scope and whose lifetime would be shorter than that of this anonymous
         function, Lua creates a closure.]=]
    return x + y
  end
end
fourplus = addto(4)
--This can also be achieved by calling the function in the following way:
print(addto(4)(3))
local splash = TextWindow.new(fourplus(3));
local splash = TextWindow.new(addto(4)(3));
```


Credits: https://en.wikipedia.org/wiki/Lua_(programming_language)

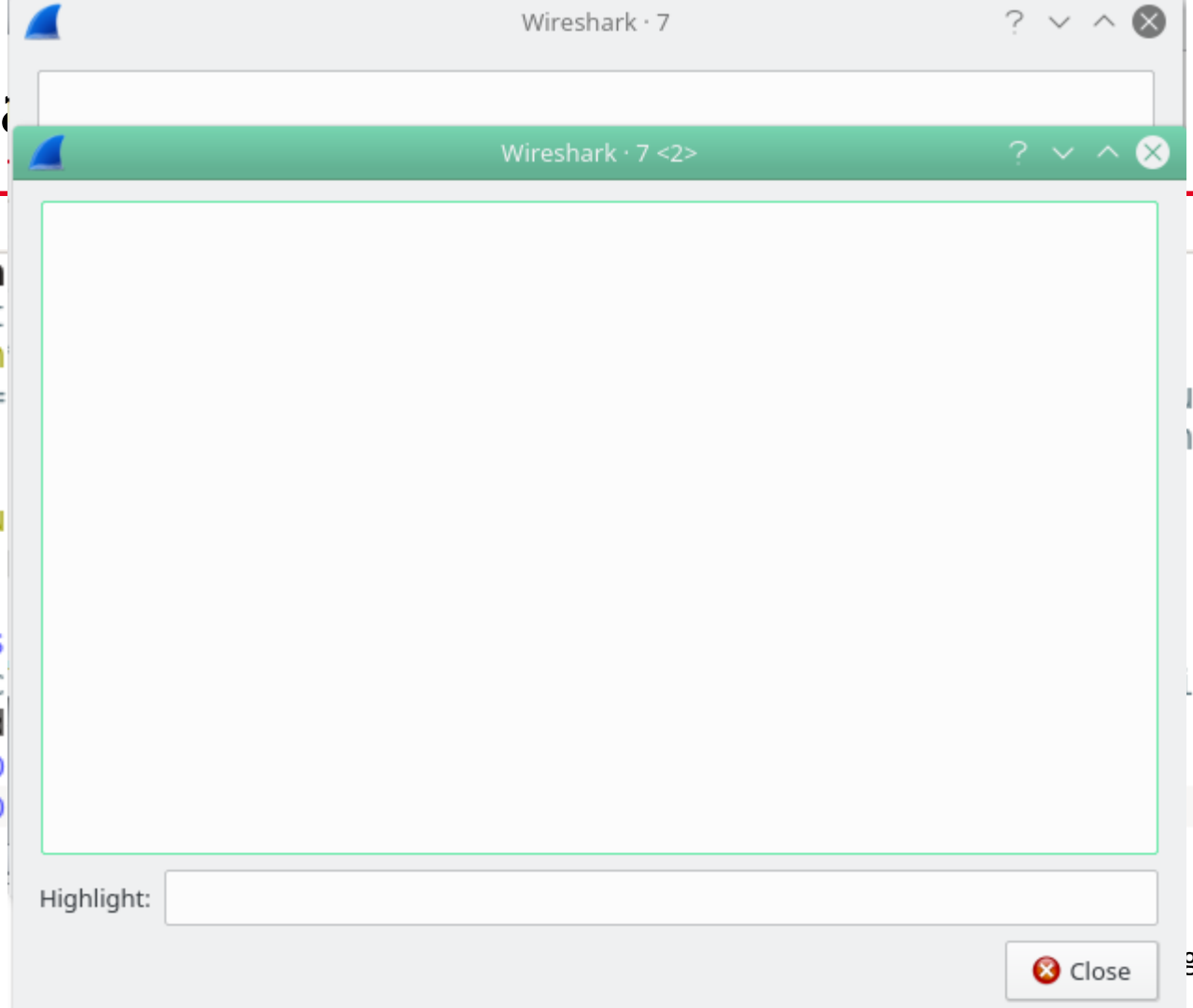
Lua ba

```
function
  -- Ret
  return
    --[=


    retu
  end
end
fourplus
--This c
print(ad
local sp
local sp
```

urrent
his anonymous

ng way:

guage)



Editing columns example

```
-- Append "<dst> -> <src>" to the Info column with a post-dissector.
-- (Taps are not guaranteed to be run at a point when they can set the
-- column text, so they can't be used for this.)

-- create a new protocol so we can register a post-dissector
local myproto = Proto("swapper","Dummy proto to edit info column")

-- the dissector function callback
function myproto.dissector(tvb,pinfo,tree)
    pinfo.cols.info:append(" " .. tostring(pinfo.dst).." -> "..tostring(pinfo.src))
end
-- register our new dummy protocol for post-dissection
register_postdissector(myproto)
```


Source: https://wiki.wireshark.org/Lua/Examples

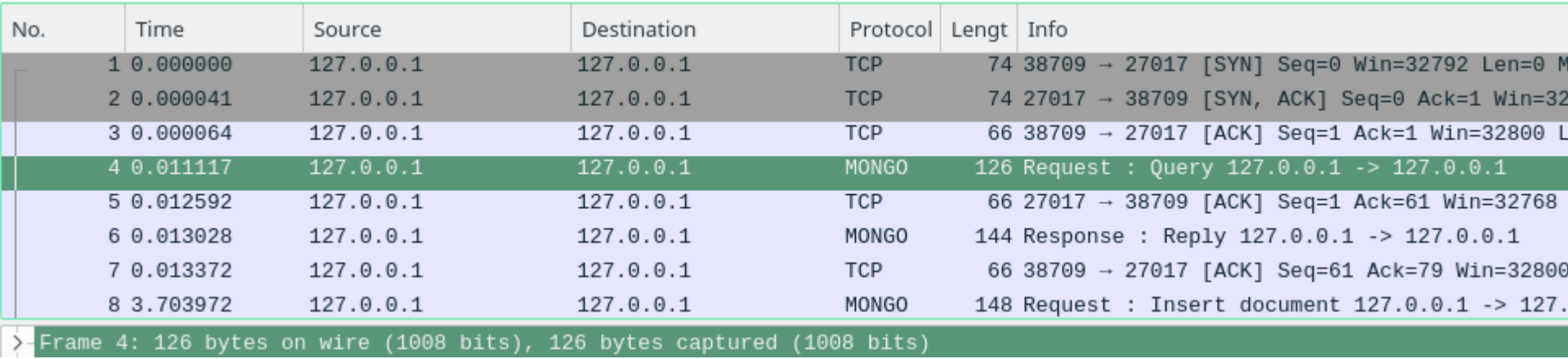
Editing columns example (before lua)

No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38709 → 27017 [SYN] Seq=0 Win=32792
2	0.000041	127.0.0.1	127.0.0.1	TCP	74	27017 → 38709 [SYN, ACK] Seq=0 Ack=
3	0.000064	127.0.0.1	127.0.0.1	TCP	66	38709 → 27017 [ACK] Seq=1 Ack=1 Win
4	0.011117	127.0.0.1	127.0.0.1	MONGO	126	Request : Query
5	0.012592	127.0.0.1	127.0.0.1	TCP	66	27017 → 38709 [ACK] Seq=1 Ack=61 Wi
6	0.013028	127.0.0.1	127.0.0.1	MONGO	144	Response : Reply
7	0.013372	127.0.0.1	127.0.0.1	TCP	66	38709 → 27017 [ACK] Seq=61 Ack=79 W
8	3.703972	127.0.0.1	127.0.0.1	MONGO	148	Request : Insert document

> Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)

Source: https://wiki.wireshark.org/Lua/Examples

Editing columns example (after execution)



No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38709 → 27017 [SYN] Seq=0 Win=32792 Len=0 M
2	0.000041	127.0.0.1	127.0.0.1	TCP	74	27017 → 38709 [SYN, ACK] Seq=0 Ack=1 Win=32
3	0.000064	127.0.0.1	127.0.0.1	TCP	66	38709 → 27017 [ACK] Seq=1 Ack=1 Win=32800 L
4	0.011117	127.0.0.1	127.0.0.1	MONGO	126	Request : Query 127.0.0.1 -> 127.0.0.1
5	0.012592	127.0.0.1	127.0.0.1	TCP	66	27017 → 38709 [ACK] Seq=1 Ack=61 Win=32768
6	0.013028	127.0.0.1	127.0.0.1	MONGO	144	Response : Reply 127.0.0.1 -> 127.0.0.1
7	0.013372	127.0.0.1	127.0.0.1	TCP	66	38709 → 27017 [ACK] Seq=61 Ack=79 Win=32800
8	3.703972	127.0.0.1	127.0.0.1	MONGO	148	Request : Insert document 127.0.0.1 -> 127.

> Frame 4: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)

Note: will only work at Wireshark's start (save the script in the Plugins folder before)

Editing trees example

```lua
local proto_foo = Proto("foo", "Foo Protocol")
proto_foo.fields.bytes = ProtoField.bytes("foo.bytes", "Byte array")
proto_foo.fields.u16 = ProtoField.uint16("foo.u16", "Unsigned short", base.HEX)

function proto_foo.dissector(buf, pinfo, tree)
        -- ignore packets less than 4 bytes long
        if buf:len() < 4 then return end

        -- ###############################################
        -- # Assume buf(0,4) == {0x00, 0x01, 0x00, 0x02}
        -- ###############################################

        local t = tree:add( proto_foo, buf() )

        -- Adds a byte array that shows as: "Byte array: 00010002"
        t:add( proto_foo.fields.bytes, buf(0,4) )

        -- Adds a byte array that shows as "Byte array: 313233"
        -- (the ASCII char code of each character in "123")
        t:add( proto_foo.fields.bytes, buf(0,4), "123" )

        -- Adds a tree item that shows as: "Unsigned short: 0x0001"
        t:add( proto_foo.fields.u16, buf(0,2) )
```


Source: https://wiki.wireshark.org/LuaAPI/TreeItem

Editing trees example

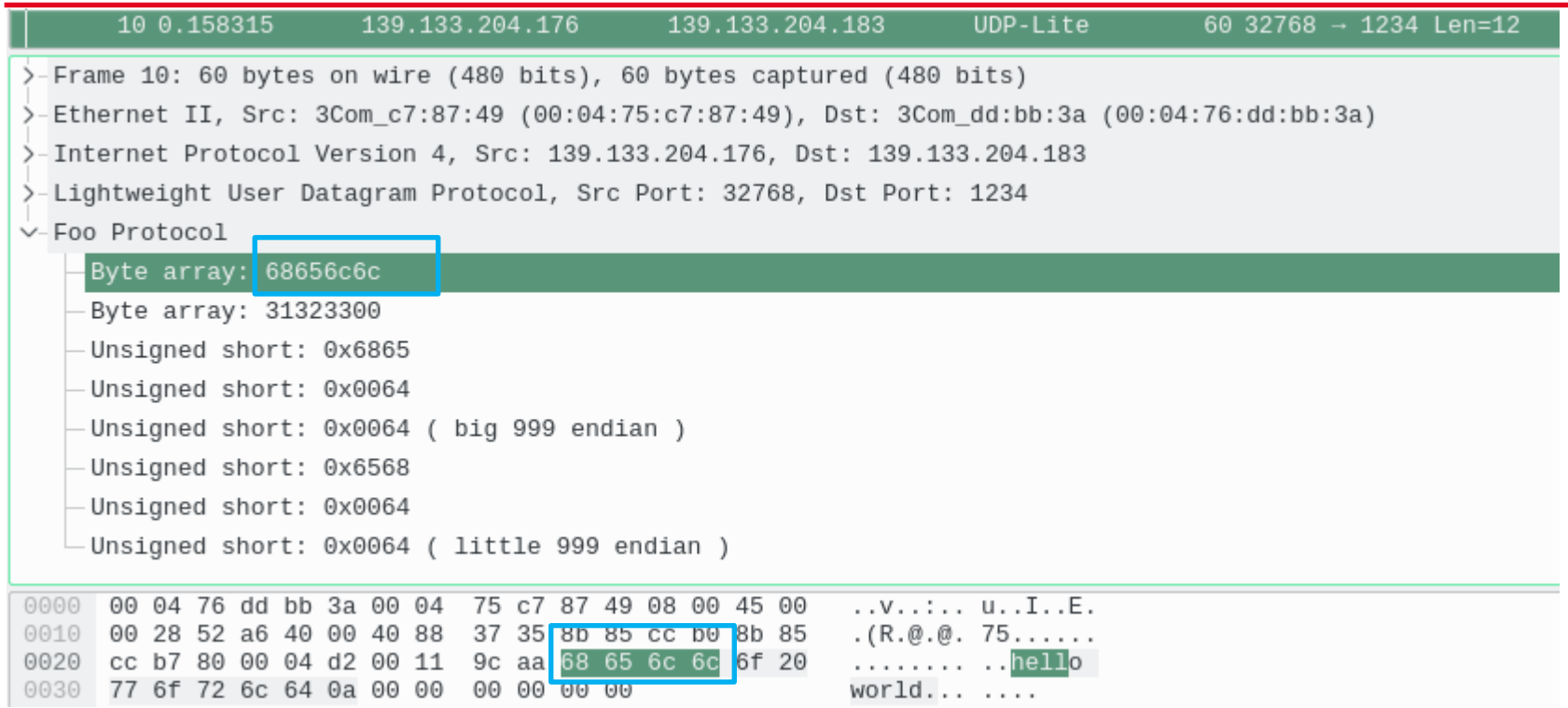
```
        -- Adds a tree item that shows as: "Unsigned short: 0x0064"
        t:add( proto_foo.fields.u16, buf(0,2), 100 )

        -- Adds a tree item that shows as: "Unsigned short: 0x0064 ( big endian )"
        t:add( proto_foo.fields.u16, buf(1,2), 100, nil, "(", nil, "big", 999, nil, "endian", nil, ")" )

        -- LITTLE ENDIAN: Adds a tree item that shows as: "Unsigned short: 0x0100"
        t:add_le( proto_foo.fields.u16, buf(0,2) )

        -- LITTLE ENDIAN: Adds a tree item that shows as: "Unsigned short: 0x6400"
        t:add_le( proto_foo.fields.u16, buf(0,2), 100 )

        -- LITTLE ENDIAN: Adds a tree item that shows as: "Unsigned short: 0x6400 ( little endian )"
        t:add_le( proto_foo.fields.u16, buf(1,2), 100, nil, "(", nil, "little", 999, nil, "endian", nil, ")" )
end

udp_table = DissectorTable.get("udp.port")
udp_table:add(32768, proto_foo)
```


Source: https://wiki.wireshark.org/LuaAPI/TreeItem

Editing trees example

```
    10 0.158315      139.133.204.176      139.133.204.183      UDP-Lite       60 32768 → 1234 Len=12
> Frame 10: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: 3Com_c7:87:49 (00:04:75:c7:87:49), Dst: 3Com_dd:bb:3a (00:04:76:dd:bb:3a)
> Internet Protocol Version 4, Src: 139.133.204.176, Dst: 139.133.204.183
> Lightweight User Datagram Protocol, Src Port: 32768, Dst Port: 1234
v Foo Protocol
    Byte array: 68656c6c
    Byte array: 31323300
    Unsigned short: 0x6865
    Unsigned short: 0x0064
    Unsigned short: 0x0064 ( big 999 endian )
    Unsigned short: 0x6568
    Unsigned short: 0x0064
    Unsigned short: 0x0064 ( little 999 endian )

0000  00 04 76 dd bb 3a 00 04  75 c7 87 49 08 00 45 00   ..v..:.. u..I..E.
0010  00 28 52 a6 40 00 40 88  37 35 8b 85 cc b0 8b 85   .(R.@.@. 75......
0020  cc b7 80 00 04 d2 00 11  9c aa 68 65 6c 6c 6f 20   ........ ..hello
0030  77 6f 72 6c 64 0a 00 00  00 00 00 00              world... ....
```

Editing trees example



HTTP Example

Edit the script so that

- it works for HTTP protocol on port 80

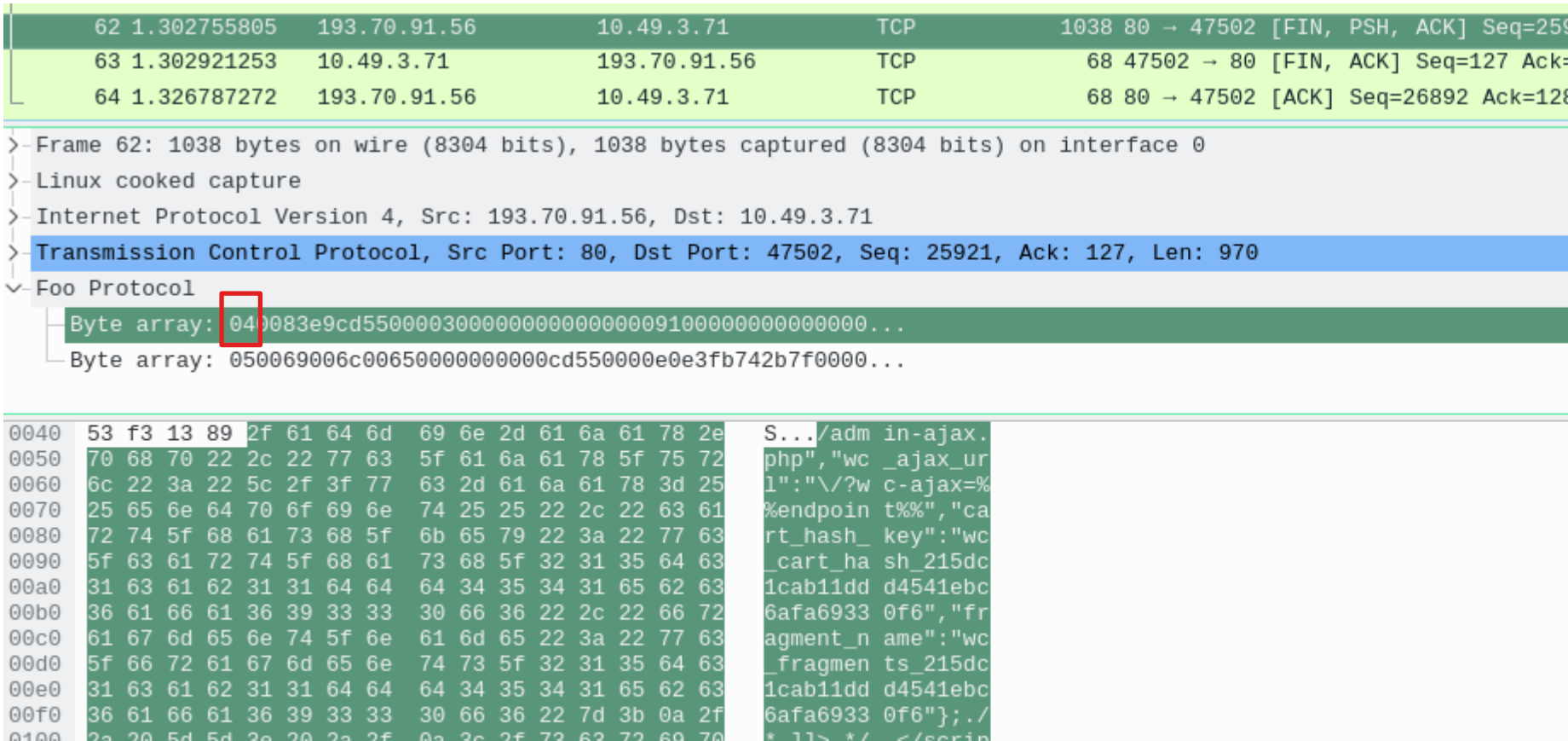
Add a function

- e.g. addition of 2 values
- output the result in a tree field

HTTP Example

```
local proto_foo = Proto("foo", "Foo Protocol")
proto_foo.fields.bytes = ProtoField.bytes("foo.bytes", "Byte array")
proto_foo.fields.u16 = ProtoField.uint16("foo.u16", "Unsigned short", base.HEX)

function addxy(x, y)
    return x + y
end

function proto_foo.dissector(buf, pinfo, tree)
        -- ignore packets less than 1 bytes long
        if buf:len() < 1 then return end

        local t = tree:add( proto_foo, buf() )

        t:add( proto_foo.fields.bytes, buf(), string.char(addxy(2,2)) )
        t:add( proto_foo.fields.bytes, buf(), string.char(addxy(2,3)) )
end

tcp_table = DissectorTable.get("tcp.port")
tcp_table:add(80, proto_foo)
```

HTTP Example



Exercise

- Develop a dissector to encode the request body of a HTTP packet into the Base64 format
  (if you know how to encode it, you will probably be able to decode it;))

Exercise

- Develop a dissector to encode the request body of a HTTP packet into the Base64 format
  (if you know how to encode it, you will probably be able to decode it;))

Hints
https://wiki.wireshark.org/LuaAPI/TreeItem
https://github.com/toastdriven/lua-base64/blob/master/base64.lua

Exercise (solution)

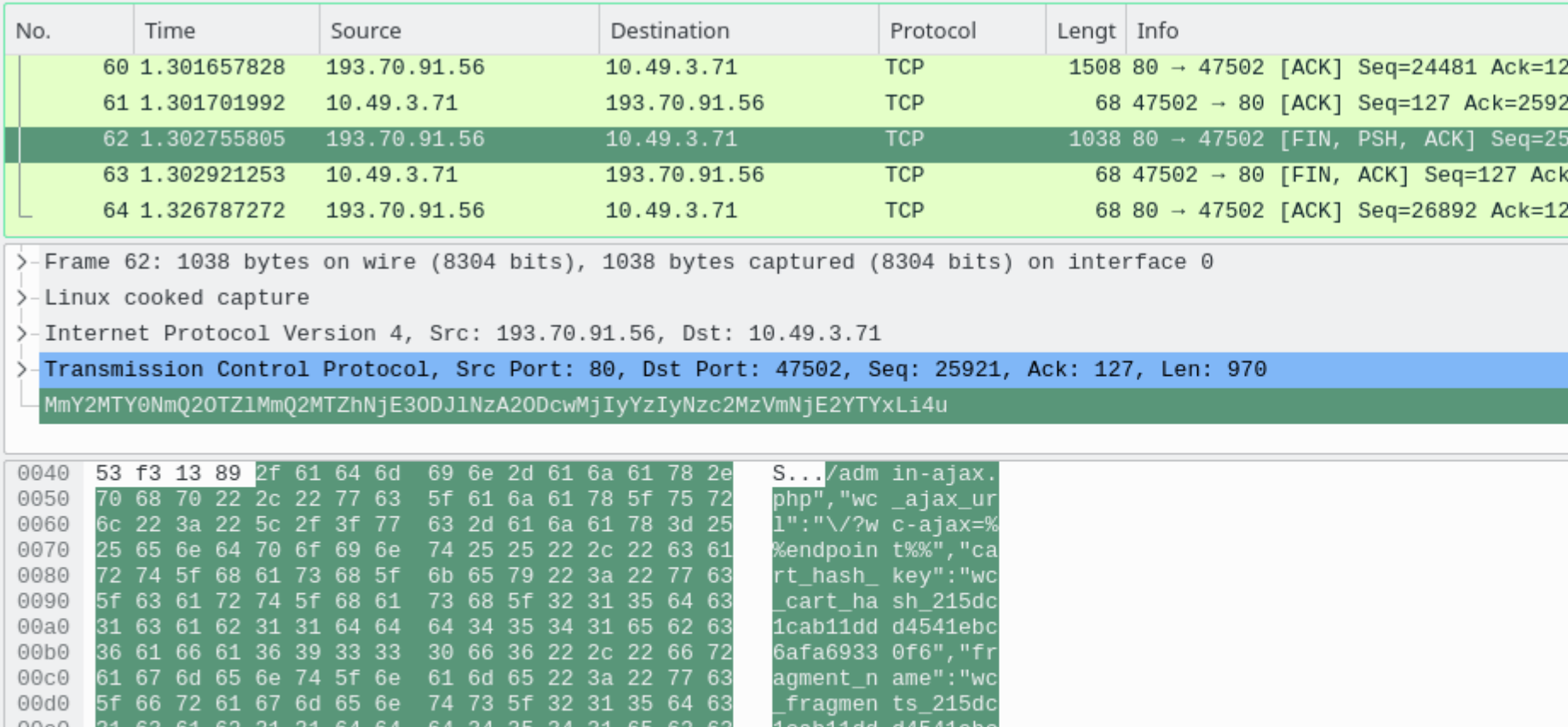
- Replace the `addxy` function with the code from https://github.com/toastdriven/lua-base64/blob/master/base64.lua

- Edit `proto_foo.dissector` as follows:

Exercise (solution)

```
function proto_foo.dissector(buf, pinfo, tree)
        -- ignore packets less than 1 bytes long
        if buf:len() < 1 then return end


        local t = tree:add( proto_foo, buf() )

        t:set_text( to_base64(tostring(buf())) )
        --t:set_text( from_base64(to_base64(tostring(buf()))) )

        --test with 'ab' string
        --t:set_text( to_base64('ab') )
end

tcp_table = DissectorTable.get("tcp.port")
tcp_table:add(80, proto_foo)
```

Exercise (solution)



No.	Time	Source	Destination	Protocol	Lengt	Info
60	1.301657828	193.70.91.56	10.49.3.71	TCP	1508	80 → 47502 [ACK] Seq=24481 Ack=12
61	1.301701992	10.49.3.71	193.70.91.56	TCP	68	47502 → 80 [ACK] Seq=127 Ack=2592
62	1.302755805	193.70.91.56	10.49.3.71	TCP	1038	80 → 47502 [FIN, PSH, ACK] Seq=25
63	1.302921253	10.49.3.71	193.70.91.56	TCP	68	47502 → 80 [FIN, ACK] Seq=127 Ack
64	1.326787272	193.70.91.56	10.49.3.71	TCP	68	80 → 47502 [ACK] Seq=26892 Ack=12

```
> Frame 62: 1038 bytes on wire (8304 bits), 1038 bytes captured (8304 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 193.70.91.56, Dst: 10.49.3.71
> Transmission Control Protocol, Src Port: 80, Dst Port: 47502, Seq: 25921, Ack: 127, Len: 970
  MmY2MTY0NmQ2OTZlMmQ2MTZhNjE3ODJlNzA2ODcwMjIyYzIyNzc2MzVmNjE2YTYxLi4u
```

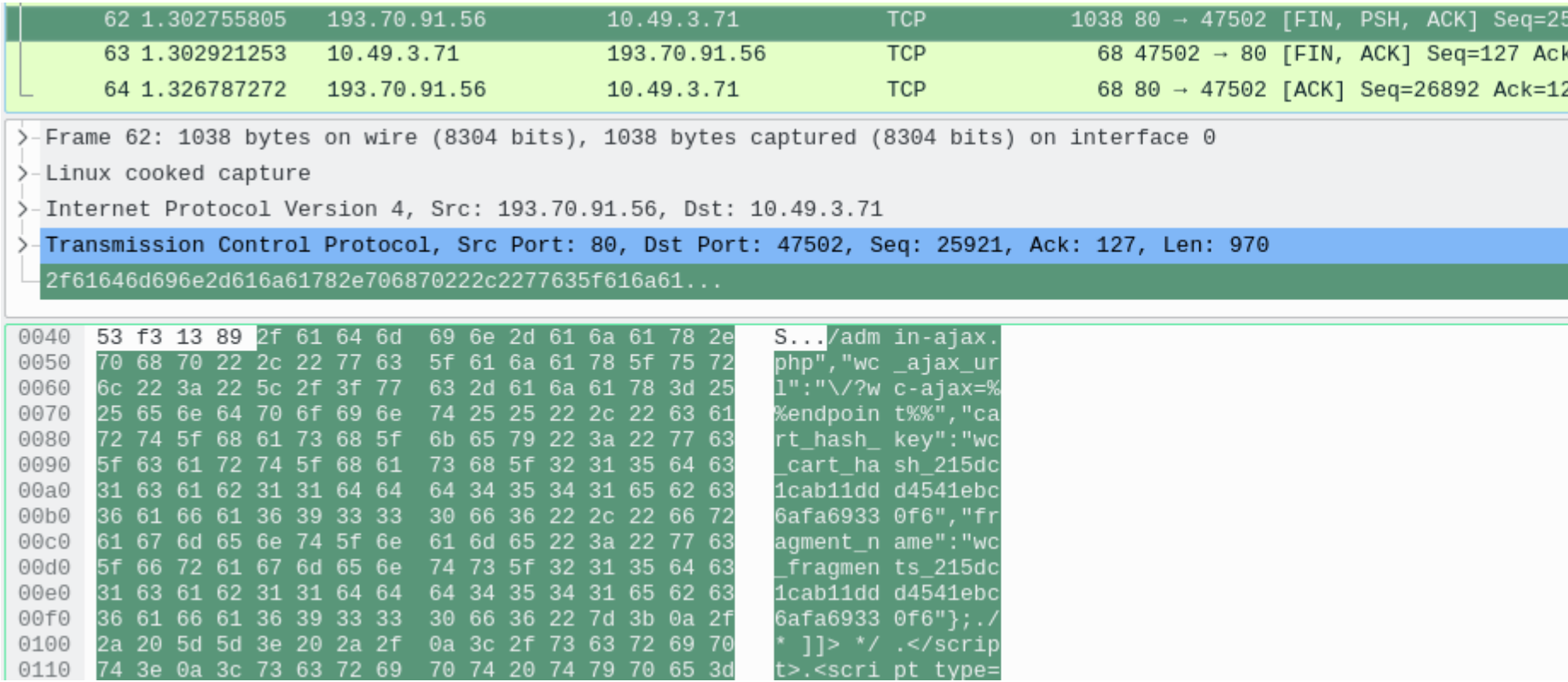
```
0040  53 f3 13 89 2f 61 64 6d  69 6e 2d 61 6a 61 78 2e   S.../adm in-ajax.
0050  70 68 70 22 2c 22 77 63  5f 61 6a 61 78 5f 75 72   php","wc _ajax_ur
0060  6c 22 3a 22 5c 2f 3f 77  63 2d 61 6a 61 78 3d 25   l":"\/?w c-ajax=%
0070  25 65 6e 64 70 6f 69 6e  74 25 25 22 2c 22 63 61   %endpoin t%%","ca
0080  72 74 5f 68 61 73 68 5f  6b 65 79 22 3a 22 77 63   rt_hash_ key":"wc
0090  5f 63 61 72 74 5f 68 61  73 68 5f 32 31 35 64 63   _cart_ha sh_215dc
00a0  31 63 61 62 31 31 64 64  64 34 35 34 31 65 62 63   1cab11dd d4541ebc
00b0  36 61 66 61 36 39 33 33  30 66 36 22 2c 22 66 72   6afa6933 0f6","fr
00c0  61 67 6d 65 6e 74 5f 6e  61 6d 65 22 3a 22 77 63   agment_n ame":"wc
00d0  5f 66 72 61 67 6d 65 6e  74 73 5f 32 31 35 64 63   _fragmen ts_215dc
```

Exercise (solution, proof)

- Decode as a proof

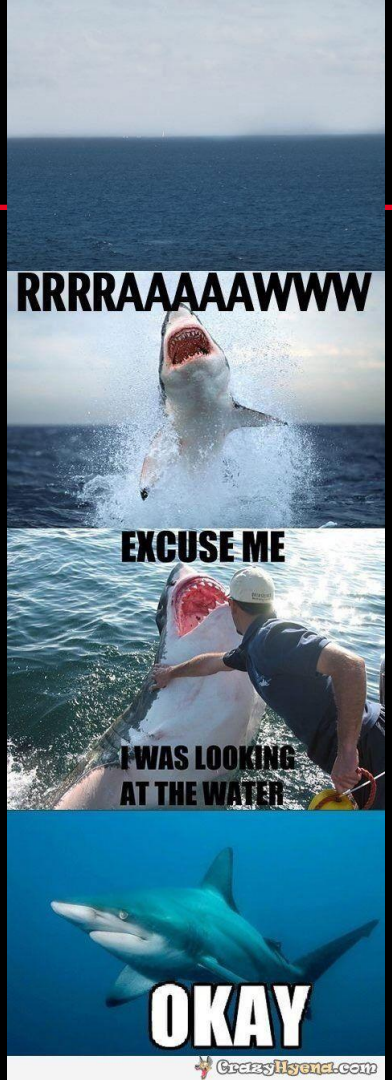
```
t:set_text(
from_base64(to_base64(tostring(buf()))) )
```

Exercise (solution)



What’s next?

What's next?


RRRRAAAAAWWW
EXCUSE ME
I WAS LOOKING
AT THE WATER
OKAY
CrazyHyena.com

What's next?

Modify/resend packets?

> use Burp, OWASP ZAP etc. proxy for HTTP/HTTPS,
> tcpreplay, tcprewrite, tcpreplay-edit
> Canape (if you dare)

Thanks!