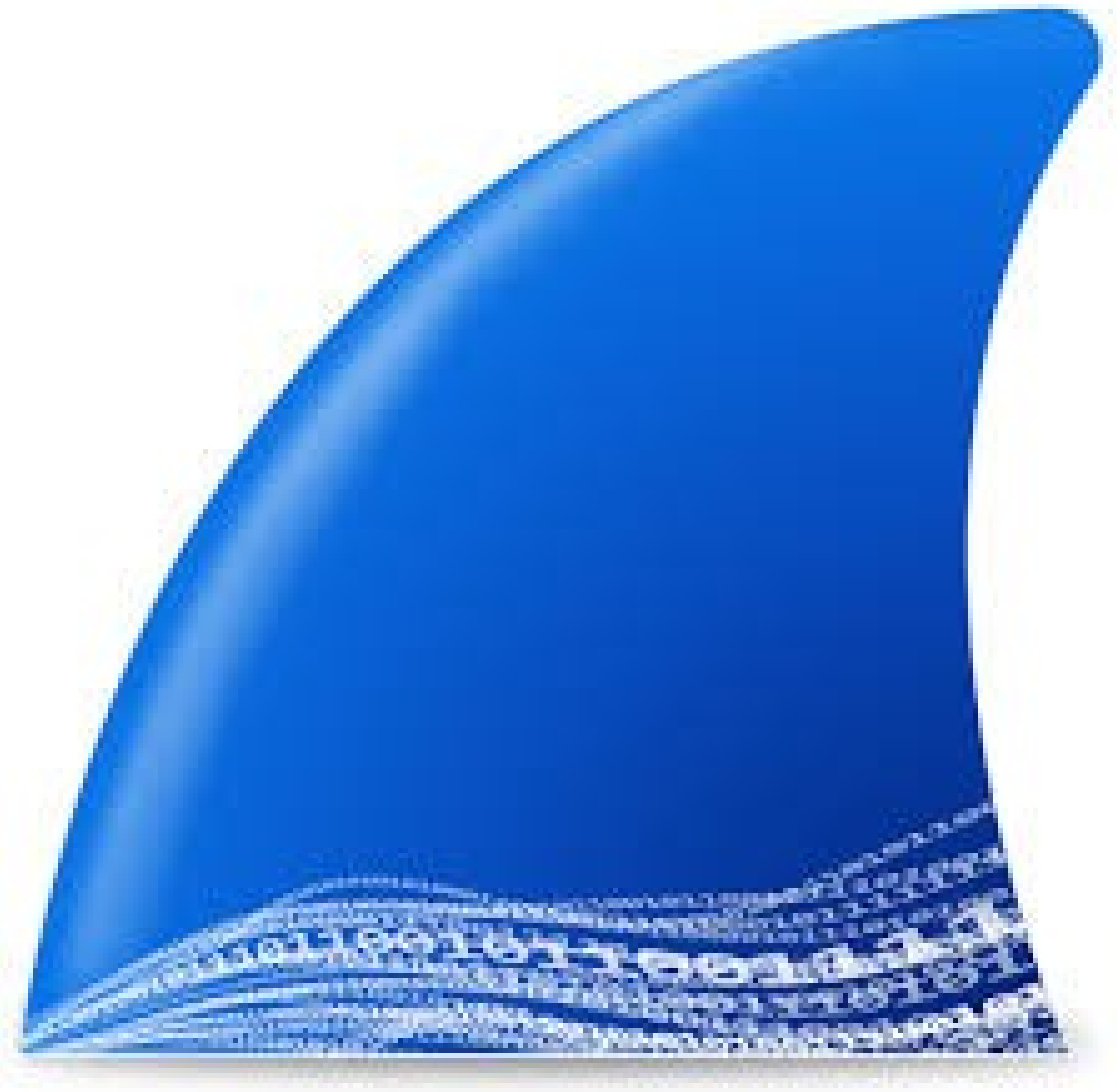
Dissecting the Client Hello with Pyshark

Katherine Leese
@diebestimmerin

Let me introduce myself

Twitter: diebestimmerin
LinkedIn: katleese

How did I get to this topic?



Overview

Pyshark

TLS

Client Hello

Backwards Compatibility

Cipher Suites

Conditional processing

Slicing and dicing

Access to python libraries

Use capture, display filters, lua dissectors (limited)

Pass t-shark arguments

Pyshark vs Tshark

T-Shark: Predefined Access (fixed extraction)

Elements must be specified upfront in the command:

tshark -r file.pcap -T fields -e tcp.srcport

No conditional access during extraction:

Any filtering or conditional logic has to happen after you've extracted the elements.

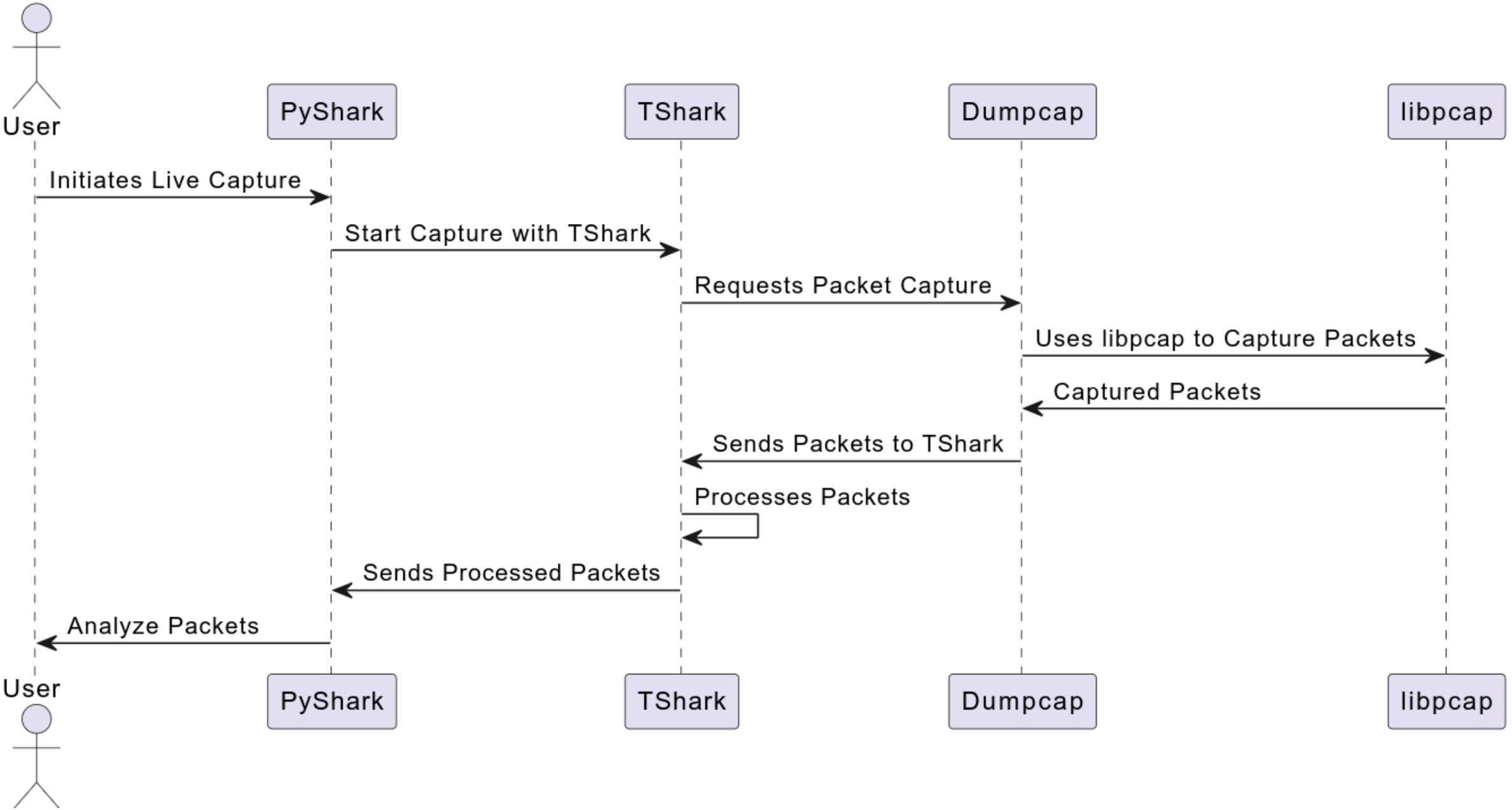
Pyshark: Dynamic Access (flexible extraction)

Access elements at any point during processing:

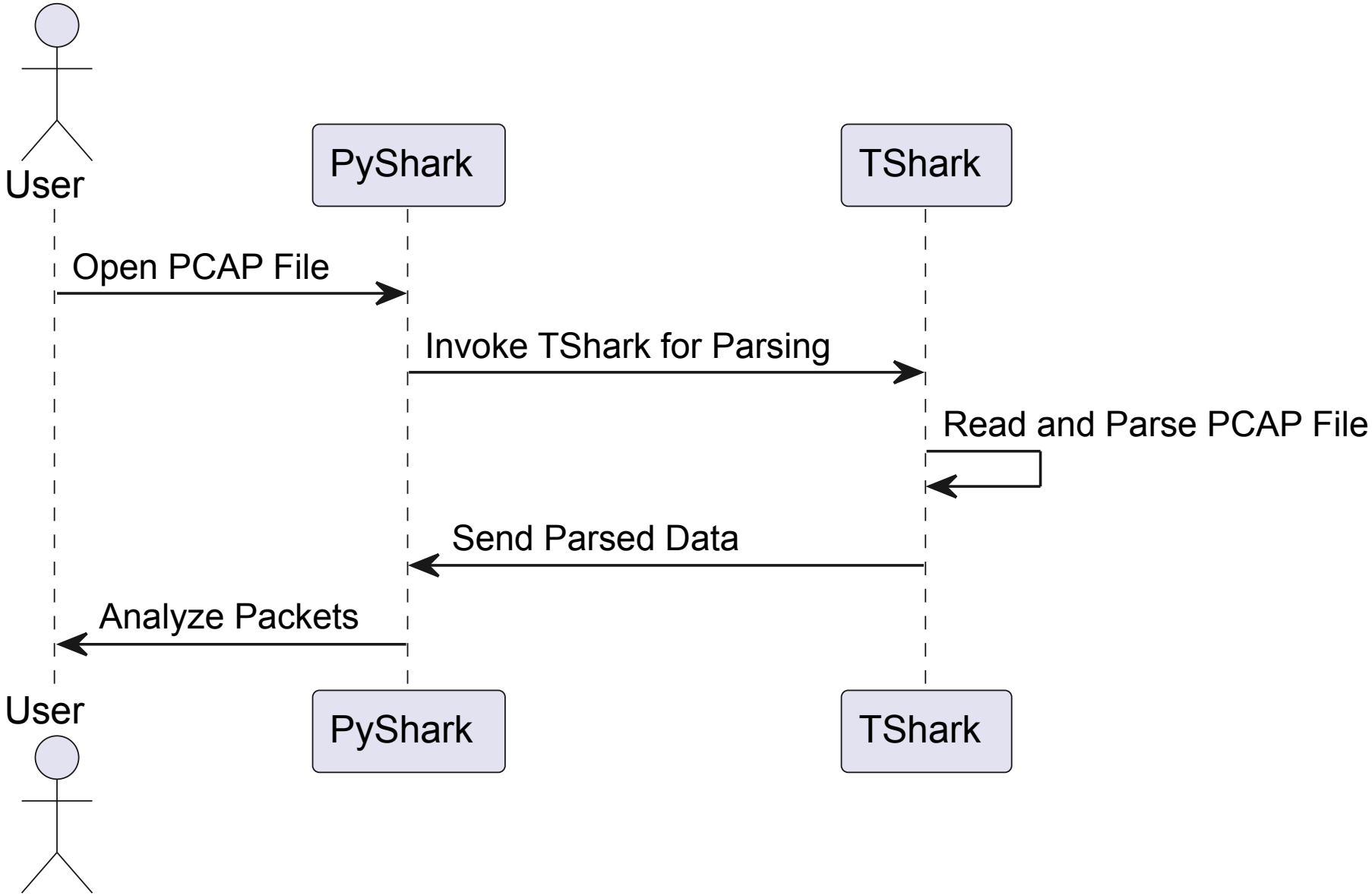
Example: First filter packets, then retrieve the source port:

```
for pkt in capture:
    if hasattr(packet, 'tcp'): print(pkt.tcp.srcport)
```

Live Capture



File Capture (read pcap)



Lack of!

Confusing

python 2

Read the code (well commented)

I was learning new things right through this entire process

Searching for information in the repo issues

Printing a packet with use_json=True fails #660

Open **landoncrabtree** opened this issue on Jul 28, 2023 · 0 comments

landoncrabtree commented on Jul 28, 2023

Describe the bug
Printing a packet with use_json=True fails

To Reproduce
Steps to reproduce the behavior:

```
pcap = pyshark.FileCapture(test.pcap, include_raw=True, use_json=True)
for p in pcap:
    print(pcap) # or print(pcap.ip)
```

```
Traceback (most recent call last):
  File "/home/landon/Downloads/hash_extractor.py", line 26, in <module>
    print(packet.ospf)
```

Assignees
No one assigned

Labels
bug

Projects
None yet

Milestone
No milestone

Development

What is happening

Apologies, I have made a wrong conclusion on the error logs. The real error was because I wrapped the above code in a `while True` loop. This forced my program to repeatedly start a new capture. Removing the loop fixed the issue.

Hey, why did you close this issue? Did you find a fix?

> My mistake it's other layer

NVM, I further dug into the code

Capturing and reading

Live Capture

```
#specify the interface
        capture = pyshark.LiveCapture(interface='eth0')
# then start it:
        capture.sniff(packet_count=100)
```

File Capture

```
#specify the pcap
        capture = pyshark.FileCapture('example.pcap')
```

Filters

Live Capture – BPF, display and lua

File Capture – display and lua

capture = (interface='eth0', display_filter ='tls.handshake.type==1')

bpf_filter only possible on LiveCapture, not FileCapture

tcp port 443 and ((tcp[((tcp[12:1] & 0xf0) >> 2):1] = 0x16) and

(tcp[((tcp[12:1] & 0xf0) >> 2) + 5:1] = 0x01))

Parameters

Interface

Filters

Only summaries – problematic with large files

Snaplen – capture only the first part of each packet

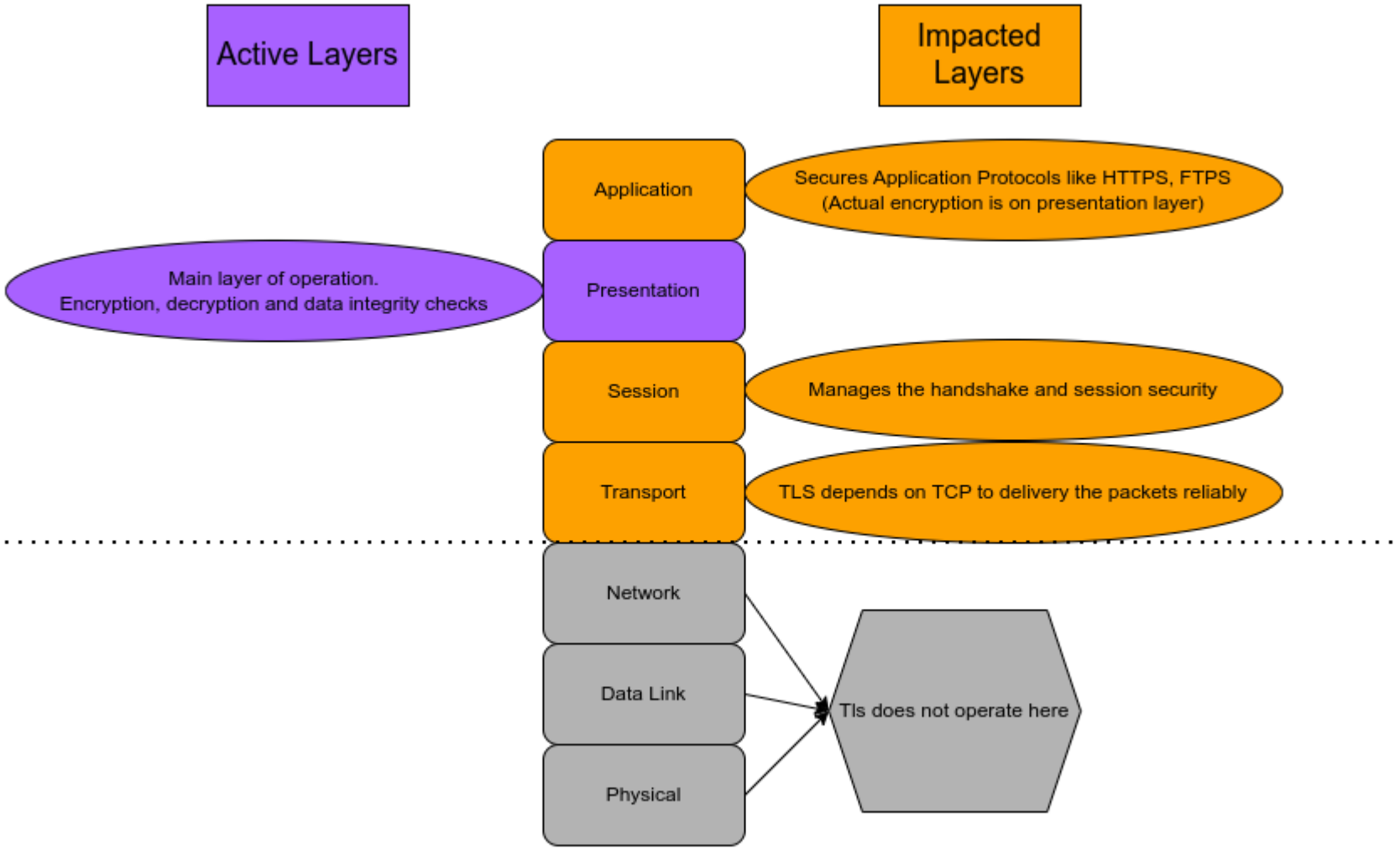
Use_pcapng – capture file will be pcap by default

tshark_args – paths, buffer size (global settings)

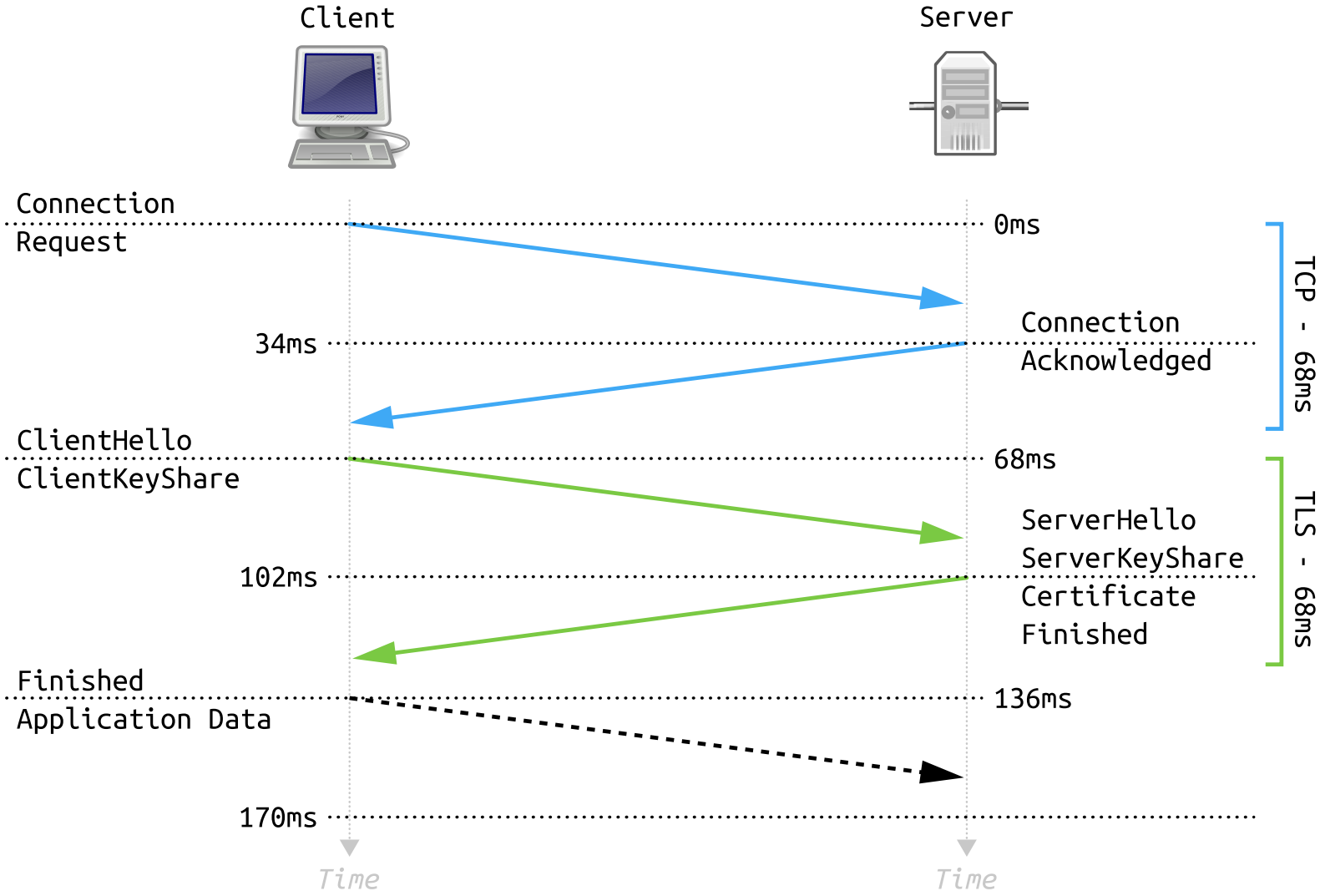
custom_parameters – duplicate packets, keylogfile

['-o', 'tcp.duplicate_ack_frame_filter:true']

TLS in the Osi Model



TLS Basics


Client
Server
Connection Request
0ms
Connection Acknowledged
34ms
TCP - 68ms
ClientHello
ClientKeyShare
68ms
ServerHello
ServerKeyShare
Certificate
Finished
102ms
TLS - 68ms
Finished
Application Data
136ms
170ms
Time
Time

Layers

```
for packet in cap:
    print(packet.layers)
```

[<ETH Layer>, <IP Layer>, <TCP Layer>, <TLS Layer>]

```
pkt[3]
pkt['tls']
pkt.tls
```

packet.eth.src
packet.eth.dst
packet.ip.src
packet.ip.dst
packet.tcp.srcport
packet.tcp.dstport

print(cap[0][0]) – ethernet layer of first packet

```
Frame 8: 298 bytes on wire (2384 bits), 298 bytes captured (2384 bits) on interface veth0, id 0
Ethernet II, Src: 32:08:c3:51:ce:fe (32:08:c3:51:ce:fe), Dst: 82:d3:f7:5e:29:17 (82:d3:f7:5e:29:17)
Internet Protocol Version 4, Src: 10.9.0.2 (10.9.0.2), Dst: tls13.crypto.mozilla.org (52.32.149.186)
Transmission Control Protocol, Src Port: 36964 (36964), Dst Port: https (443), Seq: 1, Ack: 1, Len: 232
Transport Layer Security
```

Getting the Client Hello Packets

```
cap = pyshark.FileCapture('testcap.pcapng',
display_filter='tls.handshake.type == 1 && not quic')

pkt=(cap[0])
Print(pkt.tls)
```

Which TLS version is it?

```
▾ Transport Layer Security
  ▾ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 227
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 223
        Version: TLS 1.2 (0x0303)
        Random: e0dc60c5176032733f74c616e754b83a56b2538eb3467c914eeee89e10b2e816
        Session ID Length: 0
        Cipher Suites Length: 64
      ▸ Cipher Suites (32 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 118
      ▸ Extension: renegotiation_info (len=1)
      ▸ Extension: extended_master_secret (len=0)
      ▸ Extension: session_ticket (len=0)
      ▸ Extension: signature_algorithms (len=20)
      ▸ Extension: ec_point_formats (len=2)
      ▸ Extension: Reserved (key_share) (len=38) x25519
      ▸ Extension: psk_key_exchange_modes (len=2)
      ▾ Extension: supported_versions (len=11) TLS 1.3 (draft 18), TLS 1.2, TLS 1.1, TLS 1.0, SSL 3.0
          Type: supported_versions (43)
          Length: 11
          Supported Versions length: 10
          Supported Version: TLS 1.3 (draft 18) (0x7f12)
          Supported Version: TLS 1.2 (0x0303)
          Supported Version: TLS 1.1 (0x0302)
          Supported Version: TLS 1.0 (0x0301)
          Supported Version: SSL 3.0 (0x0300)
      ▸ Extension: supported_groups (len=8)
        [JA4: t00i320900_be6088d3a083_c5dd8abecdb0]
        [JA4_r [truncated]: t00i320900_000a,002f,0033,0035,0039,003c,003d,0067,006b,009c,009d,009e,009
        [JA3 Fullstring: 771,4865-4866-4867-49195-49199-158-49196-49200-159-52393-52392-52244-52243-49
```

TLS Summary

- Record Layer – High level indicator of support for TLS1.3
- Detected TLS version – (detected by Wireshark)
- we can't extract it!

- Record Layer Version – Usually 1.0, can be 1.2, for backward compatibility
- pkt.tls.record_version – 0301 because TLS 1.0 is SSL 3.1

RFC5246 TLS1.2

Earlier versions of the TLS specification were not fully clear on what the record layer version number should contain when sending ClientHello (i.e., before it is known which version of the protocol will be employed). Thus, TLS servers compliant with this specification MUST accept any value {03,XX} as the record layer version number for ClientHello.

No single value will guarantee interoperability with all old servers, but this is a complex topic beyond the scope of this document.

RFC 8446 TLS1.3

Prior versions of TLS used the record layer version number (TLSPlaintext.legacy_record_version and TLSCiphertext.legacy_record_version) for various purposes. As of TLS 1.3, this field is deprecated.

Using dir()

dir(cap[0].tls)
'handshake', 'handshake_cipher_suites_length', 'handshake_ciphersuite', 'handshake_ciphersuites', 'handshake_comp_method', 'handshake_comp_methods', 'handshake_comp_methods_length', 'handshake_extension_len', 'handshake_extension_type', 'handshake_extensions_ec_point_format', 'handshake_extensions_ec_point_formats', 'handshake_extensions_ec_point_formats_length', 'handshake_extensions_key_share_client_length', 'handshake_extensions_key_share_group', 'handshake_extensions_key_share_key_exchange', 'handshake_extensions_key_share_key_exchange_length', 'handshake_extensions_length', 'handshake_extensions_server_name', 'handshake_extensions_server_name_len', 'handshake_extensions_server_name_list_len', 'handshake_extensions_server_name_type', 'handshake_extensions_session_ticket', 'handshake_extensions_supported_group', 'handshake_extensions_supported_groups', 'handshake_extensions_supported_groups_length', 'handshake_extensions_supported_version', 'handshake_extensions_supported_versions_len', 'handshake_ja3', 'handshake_ja3_full', 'handshake_ja4', 'handshake_ja4_r', 'handshake_length', 'handshake_random', 'handshake_random_bytes', 'handshake_random_time', 'handshake_session_id', 'handshake_session_id_length', 'handshake_sig_hash_alg', 'handshake_sig_hash_alg_len', 'handshake_sig_hash_algs', 'handshake_sig_hash_hash', 'handshake_sig_hash_sig', 'handshake_type', 'handshake_version', 'has_field', 'layer_name', 'pretty_print', 'raw_mode', 'record', 'record_content_type', 'record_length', 'record_version']

Handshake Protocol Version – Usually 1.2

suggested TLS version for TLS1.2 and lower

pkt.tls.handshake_version

It gets better…

03 03

Client Version

A protocol version of "3,3" (meaning TLS 1.2) is given. Because middleboxes have been created and widely deployed that do not allow protocol versions that they do not recognize, the TLS 1.3 session must be disguised as a TLS 1.2 session. This field is no longer used for version negotiation and is hardcoded to the 1.2 version. Instead, version negotiation is performed using the "Supported Versions" extension below.

The unusual version number ("3,3" representing TLS 1.2) is due to TLS 1.0 being a minor revision of the SSL 3.0 protocol. Therefore TLS 1.0 is represented by "3,1", TLS 1.1 is "3,2", and so on.

Supported Versions

supported_versions = packet.tls.handshake_extensions_supported_versions

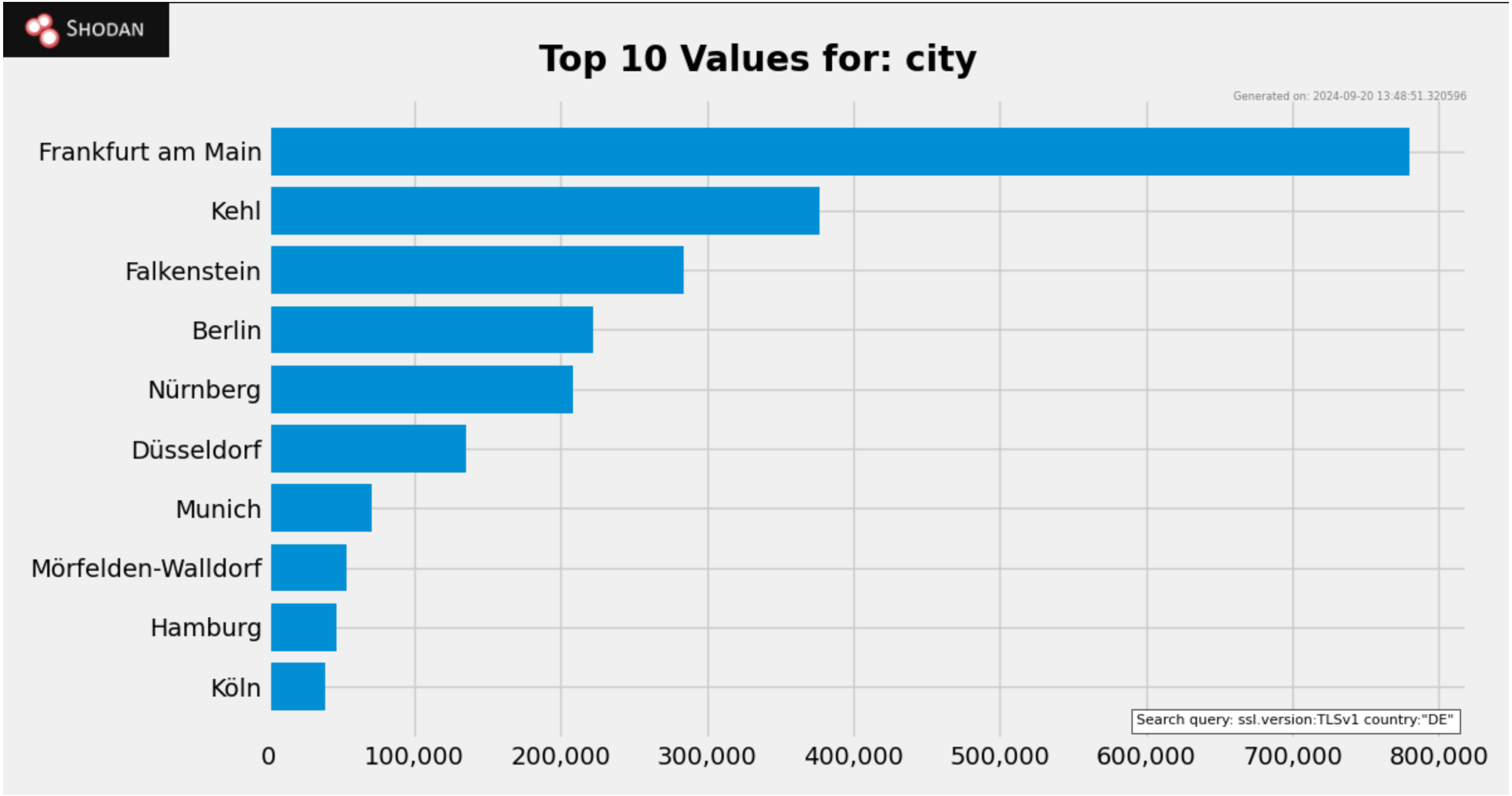
gives us only the first version

We need to loop over them to get all

```
 for packet in cap: if hasattr(packet.tls, 'handshake_extensions_supported_version’):
    supported_versions = packet.tls.handshake_extensions_supported_version
    print("Supported TLS Versions:")
    for version in supported_versions.split(',’):
        print(version.strip())
```

We can’t update the whole internet at once

The Rocky Road to TLS 1.3 and better Internet Encryption
- hanno


SHODAN
Top 10 Values for: city
Generated on: 2024-09-20 13:48:51.320596
Frankfurt am Main
Kehl
Falkenstein
Berlin
Nürnberg
Düsseldorf
Munich
Mörfelden-Walldorf
Hamburg
Köln
Search query: ssl.version:TLSv1 country:"DE"
0
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000

Which TLS version it is really?

Protocol Version

```
$ tshark -r test/captures/tls13-rfc8446.pcap -c2
1   0.000000     10.9.0.1 → 10.9.0.2     TLSv1 304 Client Hello
2   0.002634     10.9.0.2 → 10.9.0.1     TLSv1.3 658 Server Hello…..
```

```
$ tshark -r test/captures/tls13-rfc8446.pcap -c2 -2
1   0.000000     10.9.0.1 → 10.9.0.2     TLSv1.3 304 Client Hello
2   0.002634     10.9.0.2 → 10.9.0.1     TLSv1.3 658 Server Hello…….
```

C2 limits to the first two packets in the capture

-2 makes Tshark process the file twice!!!!

TLS Summary

Version that is not actually the version

TLS 1.3 disguised as 1.2 to maintain backwards compatibility

(Handshake Protocol Version)

Misleading field

older version for compatibility

Record Layer Version

Actual timetravelling!!!!

What you see is not always what you get

Getting the Cipher Suites

Layer TLS:

TLSv1 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 227

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 223

Version: TLS 1.2 (0x0303)

Random: e0dc60c5176032733f74c616e754b83a56b2538eb3467c914eeee89e10b2e816

GMT Unix Time: Jul 18, 2089 17:31:17.000000000 CEST

Random Bytes: 176032733f74c616e754b83a56b2538eb3467c914eeee89e10b2e816

Session ID Length: 0

Cipher Suites Length: 64

Cipher Suites (32 suites)

Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)

Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)

Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)

Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)

etc….

Getting the cipher suites manually

```
tshark -r tls13.pcapng -Y 'tls.handshake.type == 1' -V | grep 'Cipher Suite'
for packet in cap:
    for line in str(packet).split('\n'):
        if 'Cipher Suite:' in line:
            print(line)
```

```
Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
Etc…..
```

```
packet=cap[0]
print(dir(packet.tls))
['"handshake', 'handshake_cipher_suites_length', 'handshake_ciphersuite',
'handshake_ciphersuites', 'handshake_comp_method', 'handshake_comp_methods',
'handshake_comp_methods_length', 'handshake_extension_len',
'handshake_extension_type', 'handshake_extensions_ec_point_format',
'handshake_extensions_ec_point_formats',
'handshake_extensions_ec_point_formats_length',
'handshake_extensions_key_share_client_length',
'handshake_extensions_key_share_group',
'handshake_extensions_key_share_key_exchange',
'handshake_extensions_key_share_key_exchange_length', 'handshake_extensions_length',
'handshake_extensions_reneg_info_len', 'handshake_extensions_session_ticket',
'handshake_extensions_supported_group', 'handshake_extensions_supported_groups',
'handshake_extensions_supported_groups_length',
'handshake_extensions_supported_version',
'handshake_extensions_supported_versions_len', 'handshake_ja3', 'handshake_ja3_full',
'handshake_ja4', 'handshake_ja4_r', 'handshake_length', 'handshake_random',
'handshake_random_bytes', 'handshake_random_time', 'handshake_session_id_length',
'handshake_sig_hash_alg', 'handshake_sig_hash_alg_len', 'handshake_sig_hash_algs',
'handshake_sig_hash_hash', 'handshake_sig_hash_sig', 'handshake_type',
'handshake_version']
```

```
import pyshark
cap = pyshark.FileCapture('tls13.pcapng',
display_filter='tls.handshake.type == 1' && not quic)

for packet in cap:
    try:
        if hasattr(packet.tls, 'handshake_ciphersuite'):
            print('Handshake Cipher Suite:')
            print(packet.tls.handshake_ciphersuite)

        if hasattr(packet.tls, 'handshake_ciphersuites'):
            print('Handshake Cipher Suites:')
            print(packet.tls.handshake_ciphersuites)
```

```
Handshake Cipher Suite:
0x1301
```

```
Handshake Cipher Suites:
Cipher Suites (32 suites)
```

Examining handshake.ciphersuite

```
pkt=cap[0]
dir(pkt.tls.handshake_ciphersuite)
['__add__', '__class__', '__class__', '__contains__', '__delattr__', '__delattr__',
'__dict__', '__dir__', '__dir__', '__doc__', '__doc__', '__eq__', '__eq__', '__format__',
'__format__', '__ge__', '__ge__', '__getattr__', '__getattribute__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getstate__', '__getstate__', '__gt__', '__gt__',
'__hash__', '__hash__', '__init__', '__init__', '__init_subclass__', '__init_subclass__',
'__iter__', '__le__', '__le__', '__len__', '__lt__', '__lt__', '__mod__', '__module__',
'__module__', '__mul__', '__ne__', '__ne__', '__new__', '__new__', '__reduce__',
'__reduce__', '__reduce_ex__', '__reduce_ex__', '__repr__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__setattr__', '__setstate__', '__setstate__', '__sizeof__',
'__sizeof__', '__slots__', '__str__', '__str__', '__subclasshook__', '__subclasshook__',
'__weakref__', 'add_field', 'all_fields', 'alternate_fields', 'base16_value', 'binary_value',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'fields', 'find',
'format', 'format_map', 'get_default_value', 'hex_value', 'hide', 'index', 'int_value',
'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'main_field',
'maketrans', 'name', 'partition', 'pos', 'raw_value', 'removeprefix', 'removesuffix',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'show', 'showname',
'showname_key', 'showname_value', 'size', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'unmaskedvalue', 'upper', 'zfill']
```

print(pkt.tls.handshake_ciphersuite.int_value)

ValueError: invalid literal for int() with base 10: 'aaaa'

print(pkt.tls.handshake_ciphersuite.hex_value)

43690

hex_values = [hex(field.hex_value) for field in pkt.tls.handshake_ciphersuite.all_fields]

print(hex_values)

['0xaaaa', '0x1301', '0x1302', '0x1303', '0xc02b', '0xc02f', '0xc02c', '0xc030', '0xcca9', '0xcca8', '0xc013', '0xc014', '0x9c', '0x9d', '0x2f', '0x35']

print(pkt.tls.handshake_ciphersuite.raw_value)

aaaa

Trying with all_fields

```
print(pkt.tls.handshake_ciphersuite.all_fields)
[<LayerField tls.handshake.ciphersuite: 0xaaaa>, <LayerField
tls.handshake.ciphersuite: 0x1301>, <LayerField
tls.handshake.ciphersuite: 0x1302>, <LayerField
tls.handshake.ciphersuite: 0x1303>, <LayerField
tls.handshake.ciphersuite: 0xc02b>, <LayerField
tls.handshake.ciphersuite: 0xc02f>, <LayerField
tls.handshake.ciphersuite: 0xc02c>, <LayerField
tls.handshake.ciphersuite: 0xc030>, <LayerField
tls.handshake.ciphersuite: 0xcca9>, <LayerField etc…
```

```
>>> dir(pkt.tls.handshake_ciphersuite.all_fields)
['__add__', '__class__', '__class_getitem__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__',
'__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

```
pkt=cap[0]

raw_values = [field.raw_value for field in
pkt.tls.handshake_ciphersuite.all_fields]

print(raw_values)
```

```
['aaaa', '1301', '1302', '1303', 'c02b', 'c02f', 'c02c', 'c030', 'cca9',
'cca8', 'c013', 'c014', '009c', '009d', '002f', '0035']
```

We need to map the cipher suites

```
cipher_suites = ['(1302)', '(1303)', '(1301)', '(1304)', '(c02c)', '(cca9)']
cipher_suites_names = {
    '(1302)': 'TLS_AES_128_GCM_SHA256',
    '(1303)': 'TLS_AES_256_GCM_SHA384',
    '(1301)': 'TLS_CHACHA20_POLY1305_SHA256',
    '(1304)': 'TLS_AES_128_CCM_SHA256',
    '(c02c)':
'TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384',
}
```

```
import pyshark
cipher_suite_mapping = {
    '1301': 'TLS_CHACHA20_POLY1305_SHA256',
    '1302': 'TLS_AES_128_GCM_SHA256',
    '1303': 'TLS_AES_256_GCM_SHA384',
    'c02b': 'TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256',
Etc.…
}
cap = pyshark.FileCapture('tls_handshake.pcapng', display_filter='tls.handshake.type == 1')

for packet in cap:
    if hasattr(packet.tls, 'handshake_ciphersuite'):
        raw_values = [field.raw_value for field in packet.tls.handshake_ciphersuite.all_fields]

        print("Cipher Suites:")
        for cipher in raw_values:
            cipher = cipher.lower()  # Ensure lowercase for matching
            cipher_name = cipher_suite_mapping.get(cipher, f"Unknown Cipher Suite: 0x{cipher}")
            print(cipher_name)
cap.close()
```

Cipher Suites:

TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Unknown Cipher Suite:

0xcca8 Unknown Cipher Suite: 0x9c

Special Use Cipher Suites:

TLS_EMPTY_RENEGOTIATION_INFO_SCSV: Used to prevent renegotiation attacks by signaling that the client does not support renegotiation.
TLS_FALLBACK_SCSV: Used to indicate that the client is attempting a lower version fallback to prevent version downgrade attacks.

GREASE (Generate Random Extensions And Sustain Extensibility) Values:

GREASE_TLS_DHE_DHE_0A0A: Reserved for testing and to ensure that servers handle unknown values gracefully.
GREASE_TLS_ECDHE_ECDSA_0B0B: Reserved for testing and to ensure that servers handle unknown values gracefully.
GREASE_TLS_ECDHE_RSA_1A1A: Reserved for testing and to ensure that servers handle unknown values gracefully.

GREASE_TLS_PSK_PSK_2A2A: Reserved for testing and to ensure that servers handle unknown values gracefully.

if '00ff' not in line:

```
print(dir(cap[0].tls.handshake_extensions_supported_version.all_fields))
```

['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

```
>>> print(dir(cap[0].tls.handshake_extensions_supported_version))
```

['__add__', '__class__', '__class__', '__contains__', '__delattr__', '__delattr__', '__dict__', '__dir__', '__dir__', '__doc__', '__doc__', '__eq__', '__eq__', '__format__', '__format__', '__ge__', '__ge__', '__getattr__', '__getattribute__', '__getattribute__', '__getitem__', '__getnewargs__', '__getstate__', '__getstate__', '__gt__', '__gt__', '__hash__', '__hash__', '__init__', '__init__', '__init_subclass__', '__init_subclass__', '__iter__', '__le__', '__le__', '__len__', '__lt__', '__lt__', '__mod__', '__module__', '__module__', '__mul__', '__ne__', '__ne__', '__new__', '__new__', '__reduce__', '__reduce__', '__reduce_ex__', '__reduce_ex__', '__repr__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__setattr__', '__setstate__', '__setstate__', '__sizeof__', '__sizeof__', '__slots__', '__str__', '__str__', '__subclasshook__', '__subclasshook__', '__weakref__', 'add_field', 'all_fields', 'alternate_fields', 'base16_value', 'binary_value', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'fields', 'find', 'format', 'format_map', 'get_default_value', 'hex_value', 'hide', 'index', 'int_value', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'main_field', 'maketrans', 'name', 'partition', 'pos', 'raw_value', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'show', 'showname', 'showname_key', 'showname_value', 'size', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'unmaskedvalue', 'upper', 'zfill']

```
supported_versions_field =
packet.tls.handshake_extensions_supported_version.all_fields
[<LayerField tls.handshake.extensions.supported_version:
0x0304>, <LayerField tls.handshake.extensions.supported_version:
0x0303>]

for version in supported_versions_field:
    print(version.raw_value)

0304
0303
```

Taking it to the next level – pyshark and lua

- automated testing scenarios, eg

- test how servers react (eg SSL downgrading)

Or low TLS versions

Intrusion Detection Systems

Detecting null ciphers

PyShark and Wireshark – Similar on the Surface, Different in Depth

Graphical vs programmatic

Quick checks vs automation

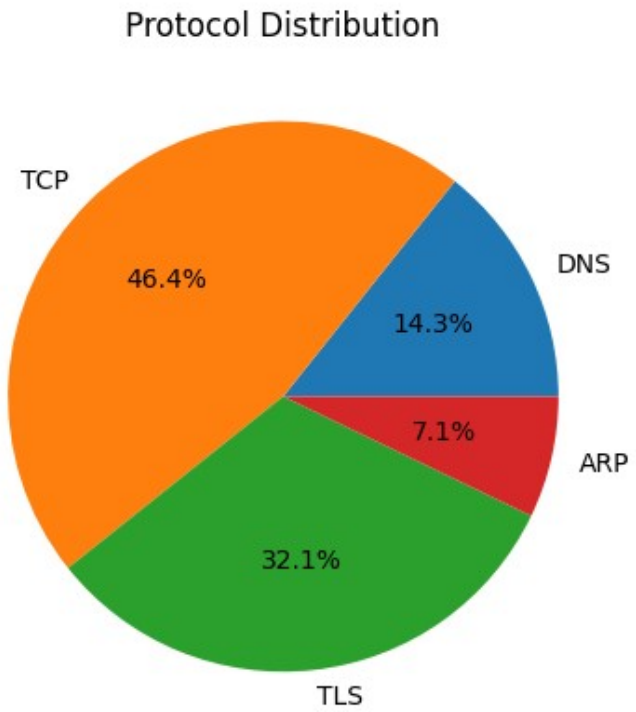
Pyshark – good for large amounts of data

Standalone tool vs access to python libraries

Biggest lesson learned – read the code!!!!

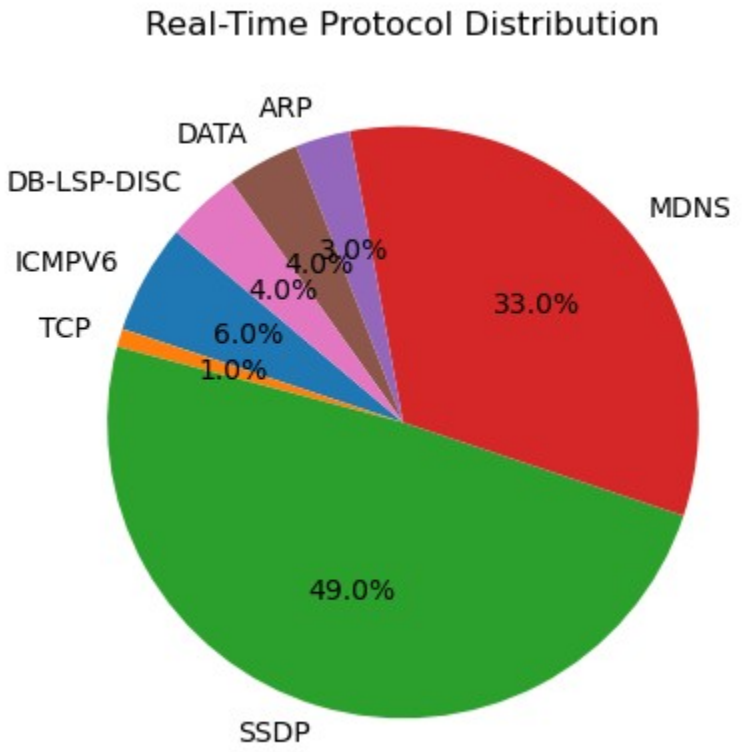
Pcap Visual Analysis

```
import pyshark
import matplotlib.pyplot as plt

cap = pyshark.FileCapture('tls13.pcapng')

protocol_counts = {}
for packet in cap:
    if hasattr(packet, 'highest_layer'):
        protocol = packet.highest_layer
        protocol_counts[protocol] = protocol_counts.get(protocol, 0) + 1

# Pie chart
protocols = list(protocol_counts.keys())
counts = list(protocol_counts.values())

plt.pie(counts, labels=protocols, autopct='%1.1f%%')
plt.title('Protocol Distribution')
plt.show()
```



Dynamic visual protocol display


```
import pyshark
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


cap = pyshark.LiveCapture(interface='wlan0')
cap.sniff(timeout=1)
protocol_counts = {}
def update(frame):
    global protocol_counts
    plt.clf()
for packet in cap.sniff_continuously(packet_count=5):  # Capture packets in small batches
        if hasattr(packet, 'highest_layer'):
            protocol = packet.highest_layer
            protocol_counts[protocol] = protocol_counts.get(protocol, 0) + 1
protocols = list(protocol_counts.keys())
    counts = list(protocol_counts.values())
plt.pie(counts, labels=protocols, autopct='%1.1f%%', startangle=140)
    plt.title('Real-Time Protocol Distribution')
fig = plt.figure()
ani = FuncAnimation(fig, update, interval=1000)  # Update the pie chart every second
plt.show()
```



Feedback