#sf21veu

Automate your analysis

TShark, the Swiss army knife

André Luyer
Rabobank

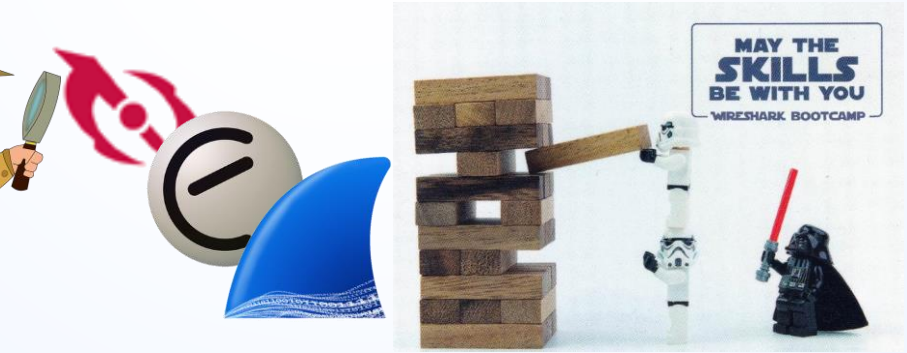
#sf21veu

Hello!

Rabobank

I am André Luyer

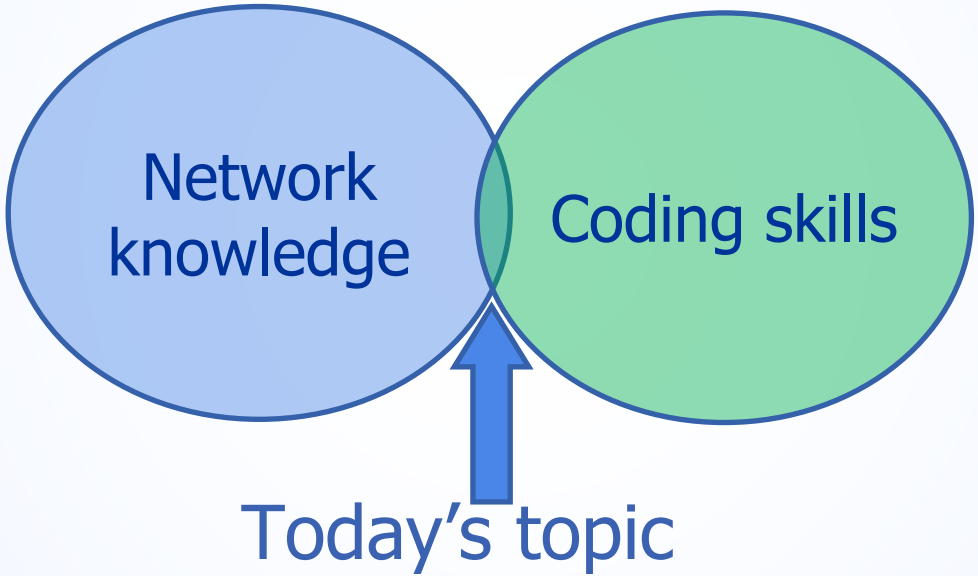
You can find me at @AndreLuyer



Why automate your analysis

- When troubleshooting
  - Quickly scan for usual suspects
- Scan for issues and anomalies
  - E.g. send alert for SSL/TLS versions used deemed unsafe or certificates that are about to expire
- For statistics
  - Trends and performance
    E.g. application turns per stream, large overhead
- Part of CI pipeline

Area of expertise

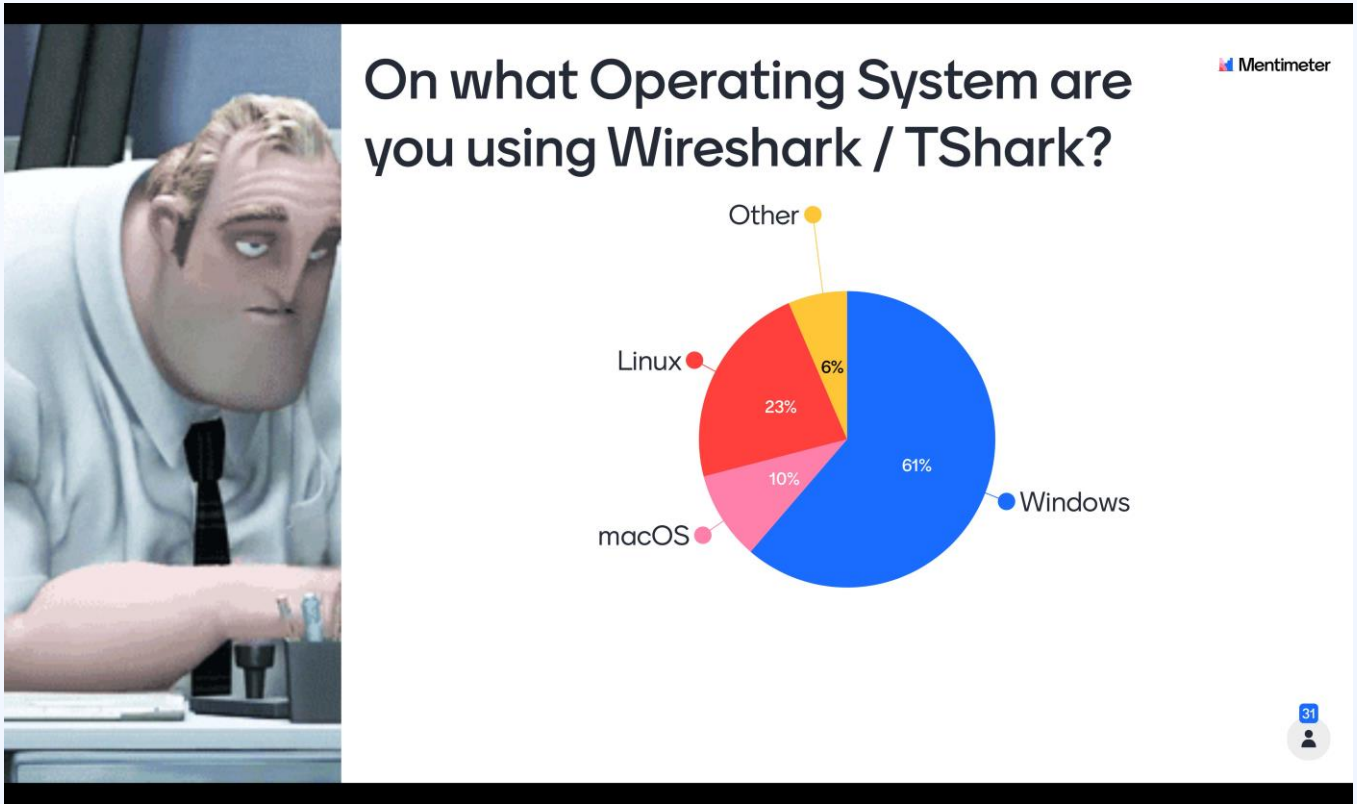


TShark versus Wireshark

- Demo

```
tshark -r sf21demo.pcapng -Y tls -c 20
```

About you

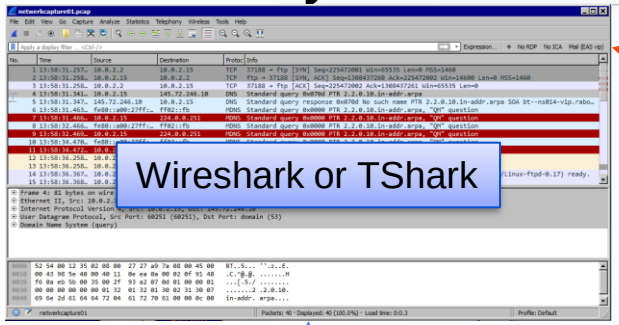

On what Operating System are you using Wireshark / TShark?
Mentimeter
Other
Linux
macOS
Windows
6%
23%
10%
61%

Other command line tools

capinfos	shows properties of a pcap
captype	prints the types of capture files
dumpcap	capture live traffic
editcap	filter on time, split, snap, convert, deduplicate, inject secrets
mergecap	merge into one
rawshark	dump and analyze raw pcap data
reordercap	sort on timestamp
text2pcap	convert hexdump into pcap

Capturing network packets locally

#sf21veu



Display filter

Read filter

File

Application

Checksum offloading

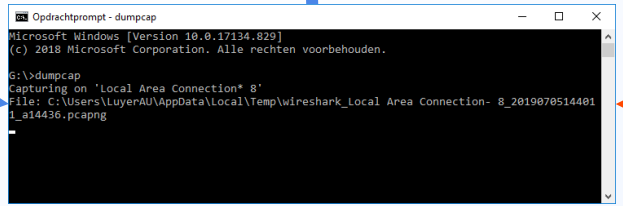
Protocol stack

Npcap driver

Firewall

NIC

Network



Capture filter (BPF)

Dumpcap

Windows PATH

- PATH not set by Wireshark installer
  - Command line tools ‘not found’
- Temporary (or in script):

```
CMD: set PATH=%PATH%;%PROGRAMFILES%\Wireshark
PS:  $env:PATH+=";$env:PROGRAMFILES\Wireshark"
```

- Permanent: Control Panel or

```
[System.Environment]::SetEnvironmentVariable("Path","…"
```

- Use full path:

```
"C:\Program Files\Wireshark\tshark.exe" -v
```

#sf21veu

TShark output examples

Command line option	As shown in Wireshark
-P | --print	Packet list pane
-V	Packets details pane, all expanded
-O <protocols>	Packets details pane, only <protocols> expanded
-x	Packet Bytes pane (hexdump) Including Decompressed, Decrypted, Reassembled
-z <statistics>	Statistics menu + Expert

Example

```
-z 'io,stat,0,BYTES,COUNT(frame)frame.len>frame.cap_len'
```

#sf21veu

TShark output -T option

-T option	Outputs
ek	‘single line’ JSON
json	‘multi line’ (pretty print) JSON
jsonraw	As json above, but without field names
pdml	XML output, packet details
psml	XML, the summary information only
text	Human readable (like -P or -V)
tabs	As text above but delimited with tabs instead of spaces
ps	Postscript
fields	Output only fields specified with -e

TShark -T fields issue

```
tshark -r tls.pcapng -O tls -2R tls.handshake.type==11 -c 1 |
   grep -A3 notBefore
                notBefore: utcTime (0)
                    utcTime: 2016-07-25 07:54:21 (UTC)
                notAfter: generalizedTime (1)
                    generalizedTime: 2050-06-02 07:54:21 (UTC)

                notBefore: utcTime (0)
                    utcTime: 2017-12-09 13:54:53 (UTC)
                notAfter: utcTime (0)
                    utcTime: 2021-12-09 14:04:53 (UTC)
```

Certificate

x509af.utcTime

x509af.generalizedTime

Which is before and after?

```
-T fields -e x509af.utcTime -e x509af.generalizedTime
```

#sf21veu

Generate hosts file from pcap

Nowadays reverse DNS lookups may *not* provide useful hostnames because:

- Website is hosted in the cloud
- Dynamic-DNS used and DNS name has been changed between the time of capturing and analyzing
- Multiple websites hosted using one IP-address

Solution:

Get the hostnames actually used out of the pcap file *itself, and use*:

- tshark ‘-H’ option
- store in pcapng, ‘-H –W n’
- save as personal hosts in configuration

#sf21veu

Generate hosts file from pcap

```
tshark -r filename.pcap -Tfields -e ipv6.dst -e ip.dst
  -e http.host
  -e tls.handshake.extensions_server_name
  -e gquic.tag.sni
  -e dhcp.fqdn.name
  -Y 'http.host||tls.handshake.extensions_server_name
       ||gquic.tag.sni||(dhcp.fqdn.name&&
       !(ip.dst==255.255.255.255))' -n |
sort -ui > hosts.txt
tshark -r filename.pcap -H hosts.txt
   -w file-with-hosts.pcapng -F pcapng -W n
```

Filter complete TCP streams (1)

For better statistical analysis, filter out incomplete streams

- Typically at begin or end of capture

- Example 1: start of TCP stream not captured
  - TCP-Syn absent

Filter complete TCP streams (2)

To filter TCP streams that were captured from their beginning:

```
output=$(tshark -r sf21tcp.pcapng -T fields -e tcp.stream
   -Y "tcp.flags.syn==1" -n | sort -u)
tshark -n -r sf21tcp.pcapng -w new.pcapng -Y "tcp.stream
   in { $output }"
```

Filter complete TCP streams (3)

- Next step: captured from start till end
  - Both TCP-Syn and TCP-Fin or TCP-Reset present

Logic:

1. Use stream number as index into (associative) array
2. Record Syn present
3. Record Fin or Reset present
4. Output list that matches criteria

no sorting required! Bit-wise OR maybe faster

#sf21veu

Filter complete TCP streams (3)

- Next step: captured from start till end
  - Both TCP-SYN and TCP-FIN or TCP-RST present

```
output=$(tshark -r sf21tcp.pcapng -T fields -e tcp.stream
   -e tcp.flags.syn -n -Y "tcp.flags & 7" |
   awk -F "\t" '{ if ($2) arr[$1] = 1;
   else if (arr[$1] == 1) arr[$1] = 2 }
   END { for (i in arr) if (arr[i] == 2) print i }')
tshark -n -r sf21tcp.pcapng -w new.pcapng -Y "tcp.stream
   in { $output }"
```

Fin/Syn/Reset

Command line length limitation

Linux: About 2 Mbyte

to get the exact value use the command:

```
echo $(( $(getconf ARG_MAX) - $(env | wc -c) - $(env | wc -l) * 8 - 8 )) # = ARG_MAX – environment array size
```

More info: `man execve`

Mac OS: About 256 kbyte, same method as Linux

Windows:

CreateProcess & PowerShell: 32767 (wide) characters

CMD: 8191 wchars (before Windows XP: 2047 wchars)

Compiled capture filter (Berkeley Packet Filter) limit: 1 page / 4 kbyte

Filter complete TCP streams (4)

Optimize by using range operator:

e.g.: `tcp.stream in { 3..99 123 }`

Logic:

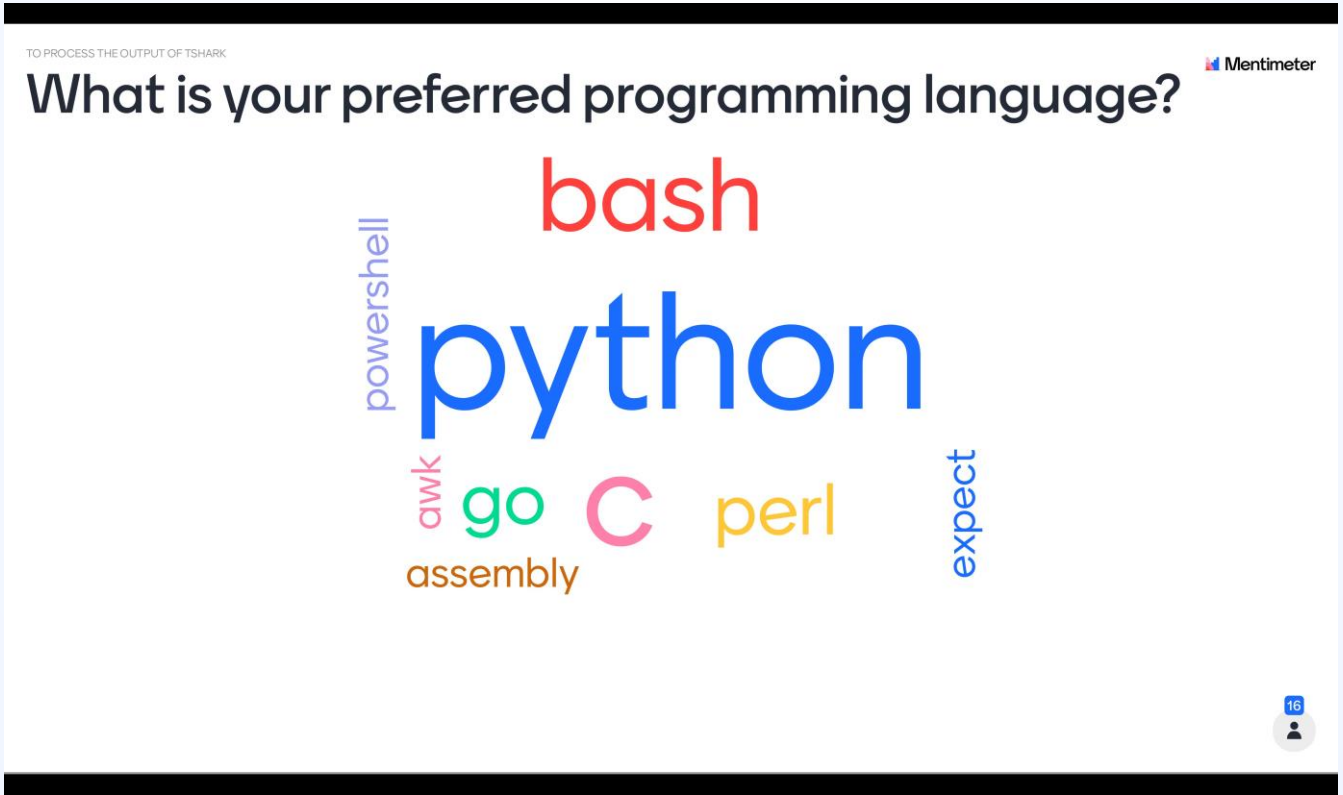
1. Loop through array
2. For every match determine range (or first gap)
3. If length range ≥ 3 use range operator
4. Else add individual number

#sf21veu

Filter complete TCP streams (5)

```
output=$(tshark -r bigfile.pcapng -T fields -e tcp.stream
  -e tcp.flags.syn -n -Y "tcp.flags & 7" |
  awk -F "\t" '{ if ($2) arr[$1] = 1
    else if (arr[$1] == 1) arr[$1] = 2;
    if ($1 > max) max = $1 }
  END { filter = ""; for (i = 0; i <= max; i++)
  if (arr[i] == 2) { # if complete TCP stream found
    j = i + 1; while (arr[j] == 2) j++ # look for range
    j-- # now range is from i to j inclusive
    if (i+2 <= j) { filter = filter " " i ".." j; i = j }
    else filter = filter " " i }
  print filter }'
```

About you


TO PROCESS THE OUTPUT OF TSHARK
What is your preferred programming language?
Mentimeter
bash
powershell
python
awk
go
C
perl
expect
assembly
16

Using pipelines

- Normally use pipeline to read TShark's output

Language	Call subprocess with pipeline
C, C++	fp = popen("tshark ...", "r");
Golang	cmd := exec.Command("tshark", "...")
Perl	open(my $fh, "-|", ('tshark', '...'))
Python	proc = subprocess.Popen('tshark ...', stdout=subprocess.PIPE)
Java	Runtime r = Runtime.getRuntime(); Process p = r.exec("tshark ...");

Process stream by stream

To simplify your analysis logic, process data stream by stream; sort by stream & time/frame

```
tshark -n -r filename.pcap -T fields -e tcp.stream
   -e frame.time_epoch  (or -e frame.number)
   -e ip.src -e _ws.col.Info -Y tcp |
sort -k 1,1n -k 2,2n
```

```
0   1528205116.256929000    172.18.224.71    56293 → 80 [SYN, ECN, C
0   1528205116.258545000    172.17.184.32    80 → 56293 [SYN, ACK] S
0   1528205116.258588000    172.18.224.71    56293 → 80 [ACK] Seq=1
0   1528205116.258769000    172.18.224.71    POST /demo/Sharkfest
0   1528205116.258825000    172.18.224.71    56293 → 80 [ACK] Seq=35
```

TShark in Perl example

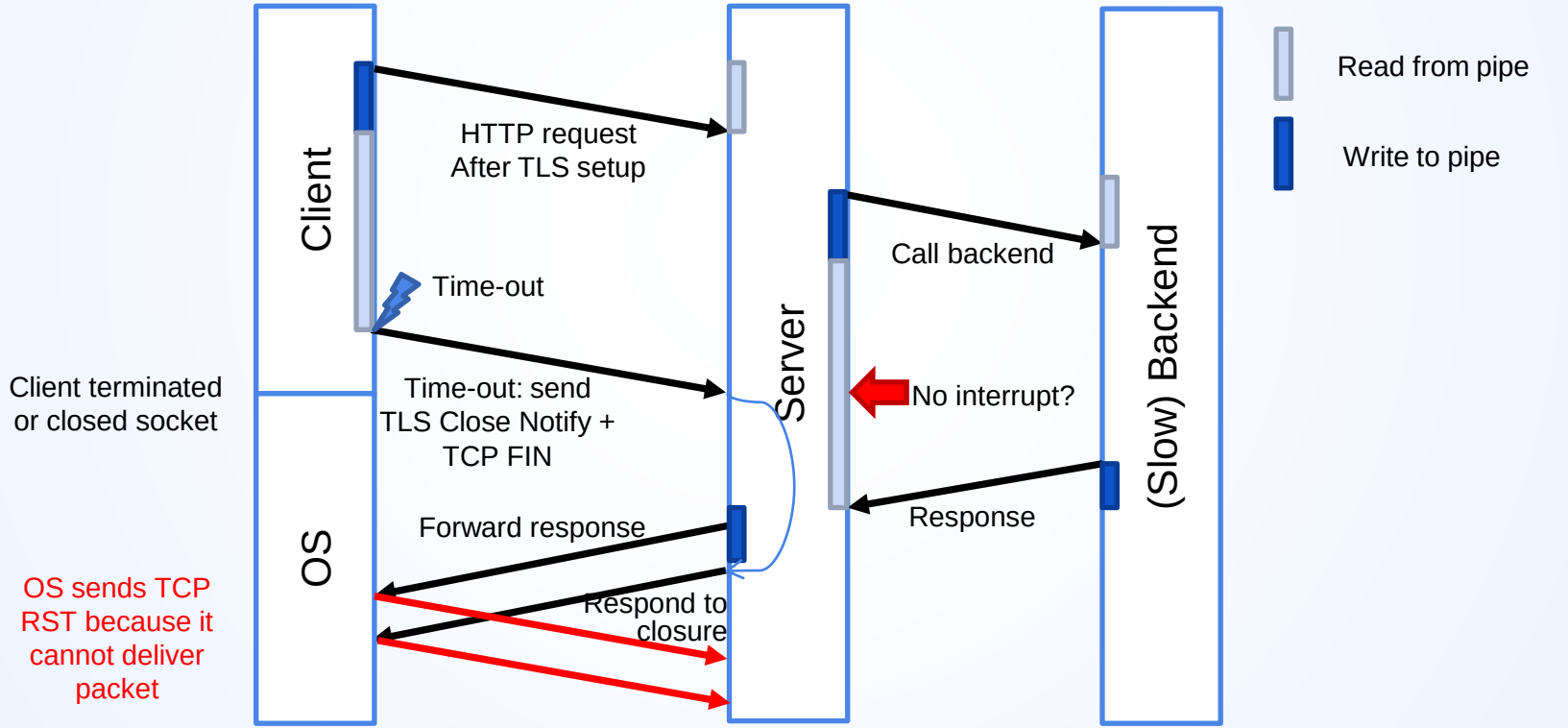
```
my $command = "tshark -r '$file' -n -T fields"
   . " -e tcp.stream -e frame.time_epoch"
   . " -e ip.src -e _ws.col.Info -Y tcp"
   . "| sort -k 1,1n -k 2,2n"; # sort by column 1 & 2 numeric
my $tcpnr = -1; # to detect new TCP stream
open(my $fh, "-|", $command) or die "Command failed $!";
while (<$fh>) {
   my ($tcp, $time, $ipsrc, $info) = split "\t";
   if ($tcp != $tcpnr) { new_tcp_stream(); $tcpnr = $tcp; }
   # process fields here…
}
close($fh);
new_tcp_stream() if $tcpnr >= 0;
```

Data after Fin or Reset

- TCP protocol allows half open connections
- Not useful for applications
  - So TCP data after Fin or Reset indicates a problem
    - Exception: TLS Alert (Close Notify)

Data after Fin or Reset example

#sf21veu



Data after Fin or Reset

Logic:

For every packet do:

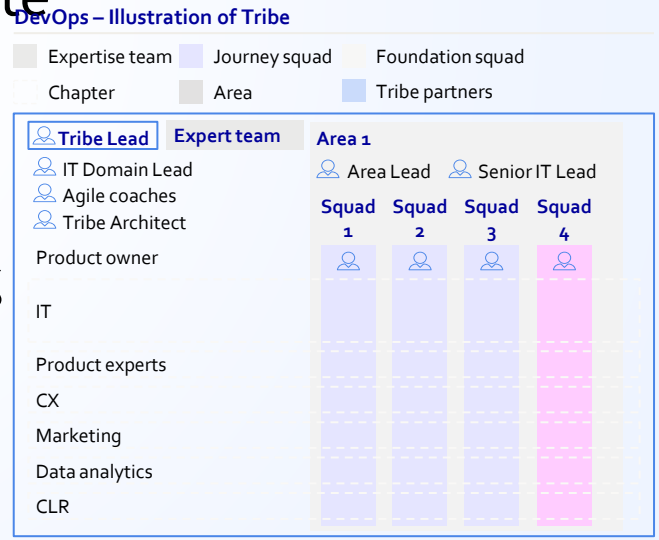
1. If Fin or Reset mark Close_Started
2. If tcp.len > 0 and Close_Started and not TLS Alert then mark Data-after-Fin/Reset
   - Optional: and time delta > RTT
3. On TCP stream change or last packet do issue warning (Fin) or critical (Reset)

Our implementation

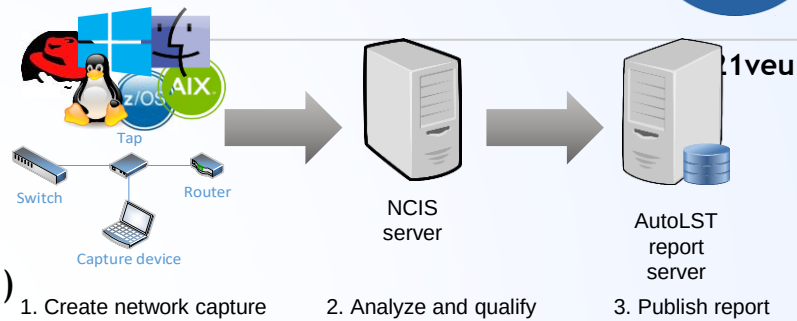
- From ‘handy scripts’ to fully automated tool
  - Novice users can upload the pcap file
  - Report generated
  - Also used as part of performance load tests

Why did *we* automate

- Need for fast feedback on development activities on stability and performance
  - Application monitoring incomplete
- DevOps – many small teams
  - Need: fall back to expert team
- Repetitive work
  - Part of performance load testing checking if issues were fixed
- Shift towards HTTP/TLS
  - Micro services & cloud based



How did *we* automate



Steps:

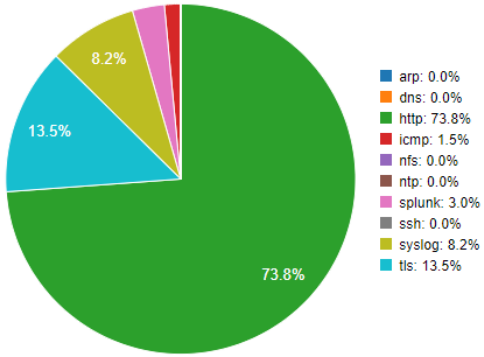
1. Upload pcap (can be automated)
2. Sanity check
   - Reject: corrupted files, too many snapped packets, too low traffic volume, high dropped packet rate
3. Split in known protocols/applications – simplifies analysis
4. Process individual protocols/applications
   - Filter out incomplete streams, process stream by stream
5. Send results to report server (JSON format)

Demo

#sf21veu

Global (all combined) summary

Traffic by type graph:



SHOW DETAILED INFORMATION | SHOW TYPE DESCRIPTIONS

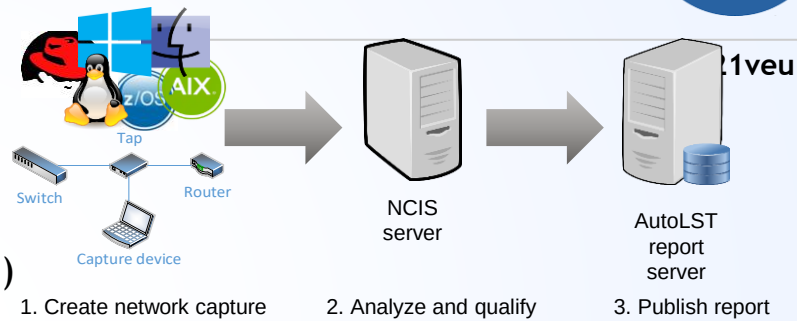
Result summary

	positive	minor	major	critical	undetermined	none
Occurrence	3	5	3	5	0	0

TShark -z io,stat from Perl


```
my @command = ('tshark', '-r', $file, '-q', '-n'
   , '-z', 'io,stat,0' # generate IO stats, totals only
   . ',FRAMES,BYTES'   # frame & byte totals
   . ',COUNT(frame)frame.time_delta<-0.0005'
      # negative delta is a sign of dropped packets in kernel
   . ',COUNT(frame)frame.len>frame.cap_len'); # snapped!
open(my $fh, "-|", @command) or die "Command failed $!";
while (<$fh>) {
   next if not /<>/; # ignore ‘human readable’ output
   tr/|<>\r\n//d;    # remove formatting
   my @columns = split ' '; # split AWK style
   print join("\t", @columns), "\n"; # process fields…
} close($fh);
```

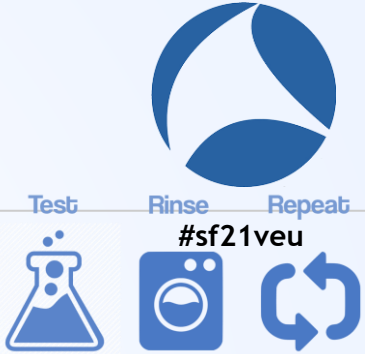
How did *we* automate



Steps:

1. Upload pcap (can be automated)
2. Sanity check
   - Reject: corrupted files, too many snapped packets, too low traffic volume, high dropped packet rate
3. Split in known protocols/applications – simplifies analysis
4. Process individual protocols/applications
   - Filter out incomplete streams, process stream by stream
5. Send results to report server (JSON format)

Split in known protocols



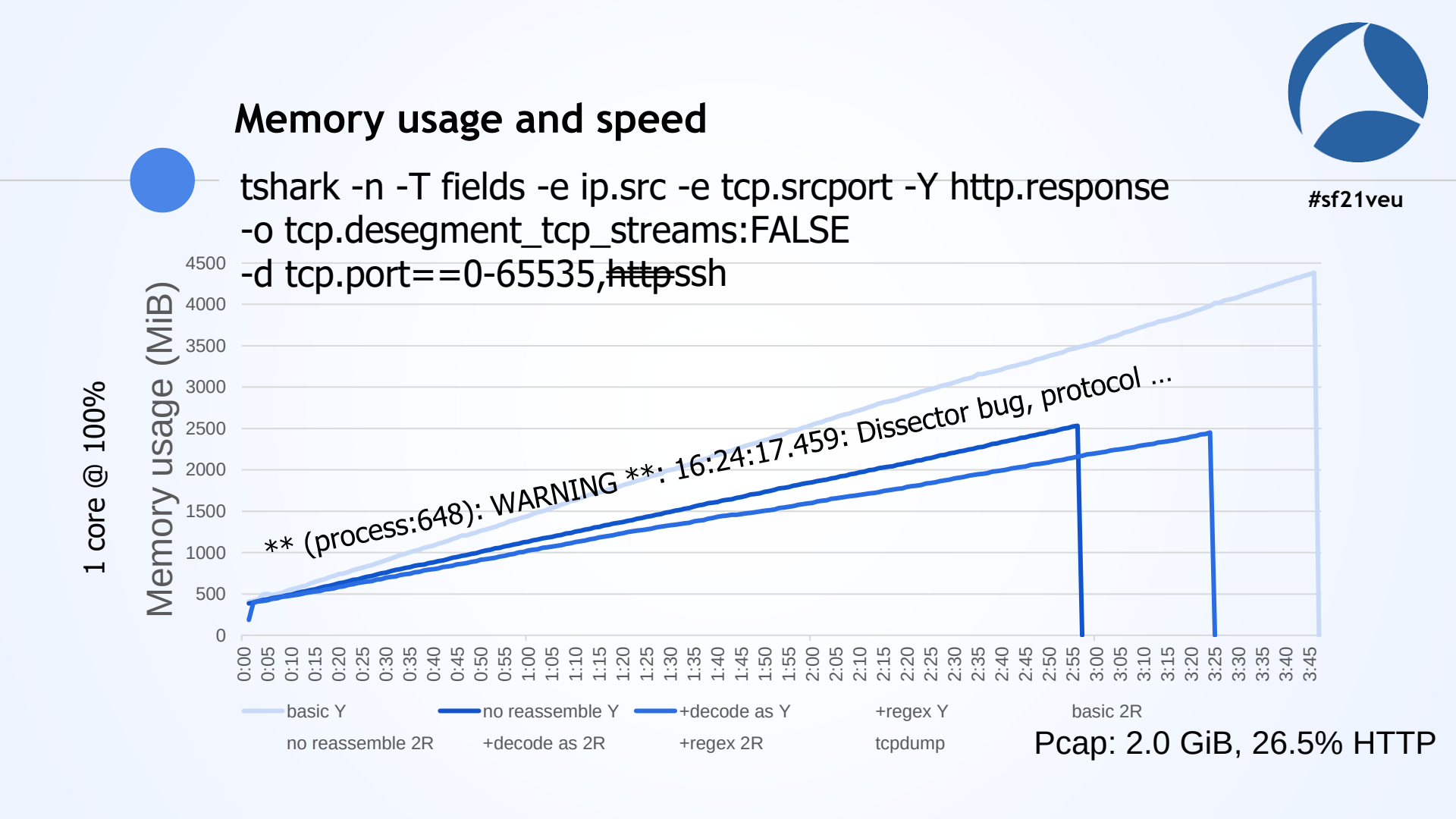
Steps:

1. Detect protocol/application by packet data
   - Avoid need for config files, more robust
2. Create pcap using filter
3. Create new pcap using ‘not’ filter
4. Repeat steps 1-3 for next protocol/application

Requirements for tool

- Network capture should contain enough data for proper statistical analysis, thus preferably:
  - Capture duration at least 5 minutes
  - Not ‘pre filtered’
- Process pcap with file size up to 5 Gbyte
  - What server configuration needed?

Memory usage and speed



#sf21veu

tshark -n -T fields -e ip.src -e tcp.srcport -Y http.response
-o tcp.desegment_tcp_streams:FALSE
-d tcp.port==0-65535,~~http~~ssh

1 core @ 100%

Memory usage (MiB)

4500
4000
3500
3000
2500
2000
1500
1000
500
0

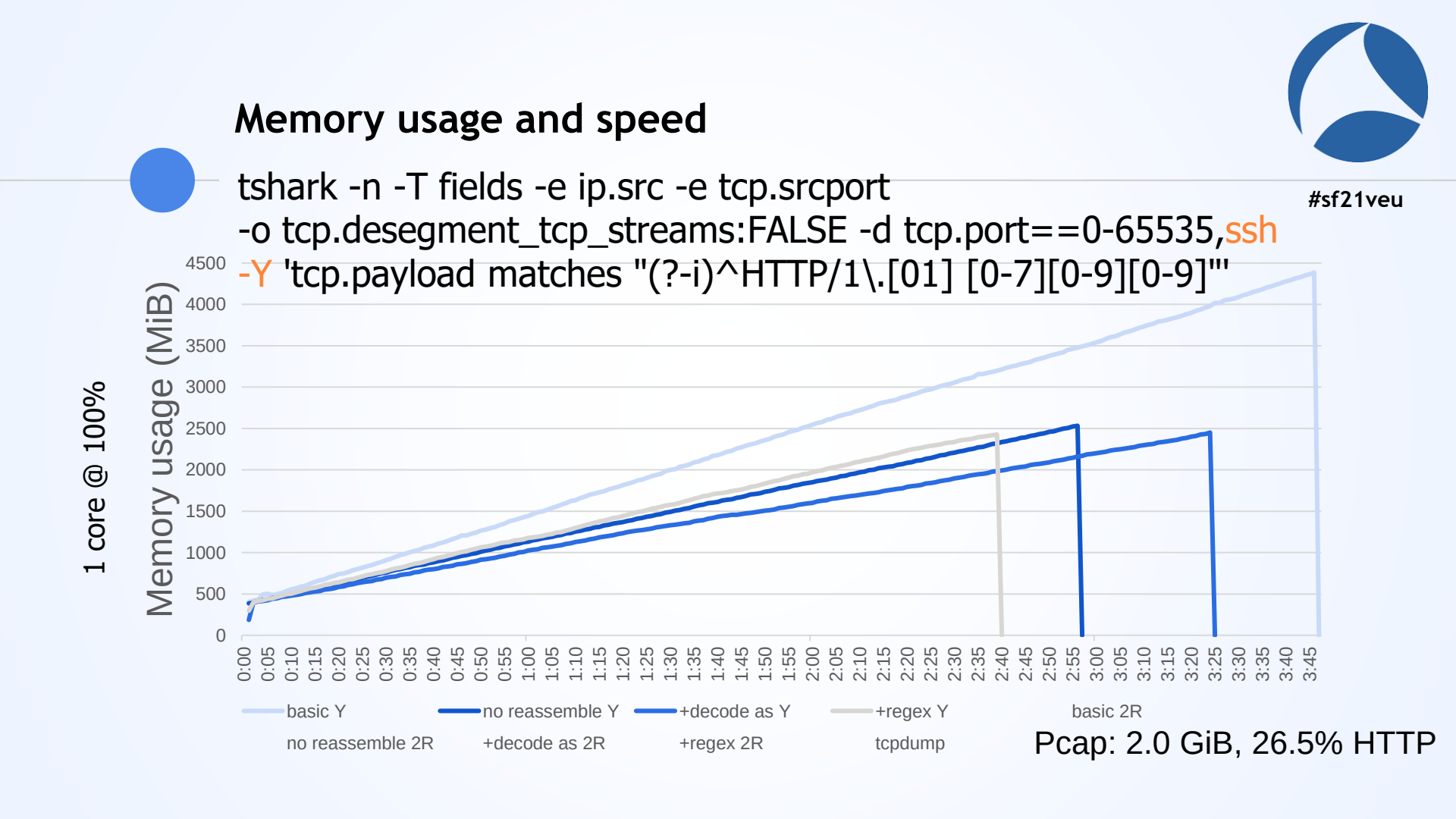
** (process:648): WARNING **: 16:24:17.459: Dissector bug, protocol ...

0:00 0:05 0:10 0:15 0:20 0:25 0:30 0:35 0:40 0:45 0:50 0:55 1:00 1:05 1:10 1:15 1:20 1:25 1:30 1:35 1:40 1:45 1:50 1:55 2:00 2:05 2:10 2:15 2:20 2:25 2:30 2:35 2:40 2:45 2:50 2:55 3:00 3:05 3:10 3:15 3:20 3:25 3:30 3:35 3:40 3:45

basic Y
no reassemble Y
+decode as Y
+regex Y
basic 2R
no reassemble 2R
+decode as 2R
+regex 2R
tcpdump

Pcap: 2.0 GiB, 26.5% HTTP

Memory usage and speed



#sf21veu

```
tshark -n -T fields -e ip.src -e tcp.srcport
-o tcp.desegment_tcp_streams:FALSE -d tcp.port==0-65535,ssh
-Y 'tcp.payload matches "(?-i)^HTTP/1\.[01] [0-7][0-9][0-9]"'
```

1 core @ 100%

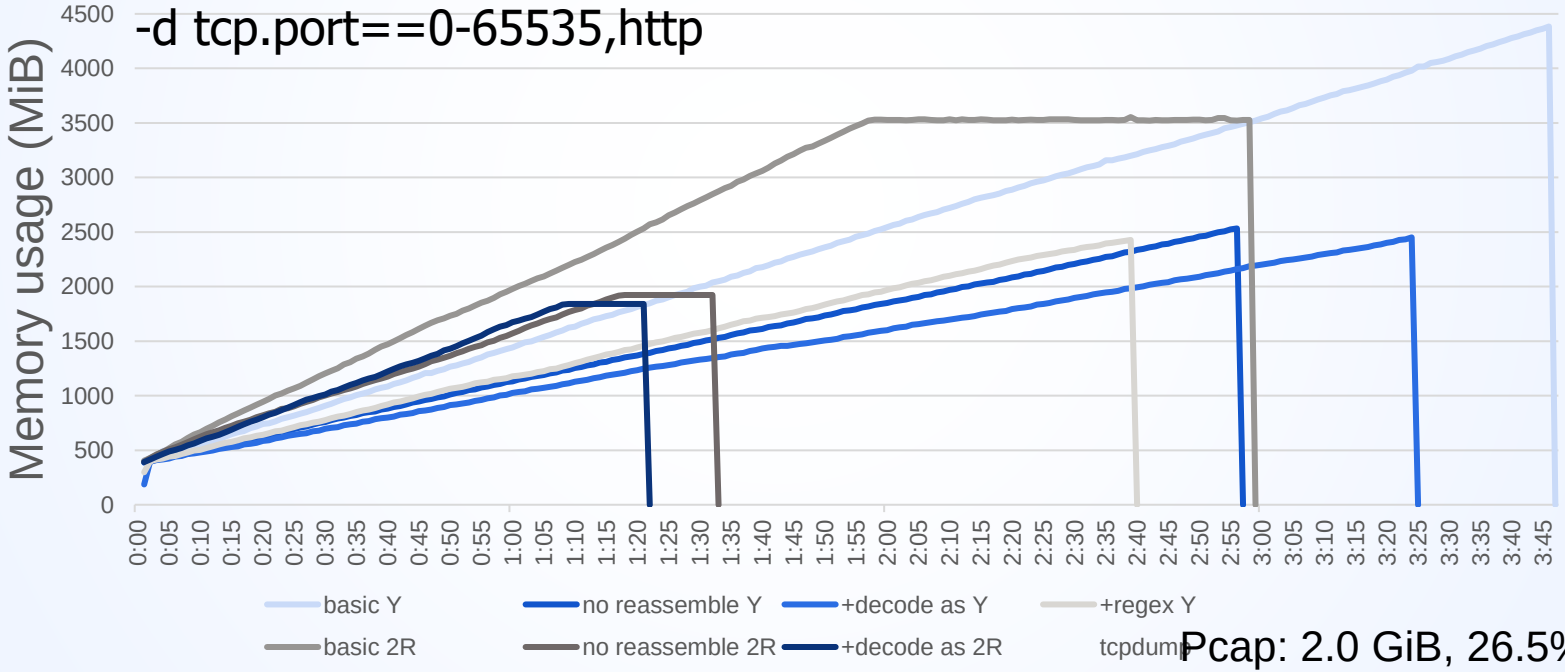
Pcap: 2.0 GiB, 26.5% HTTP

Memory usage and speed

#sf21veu

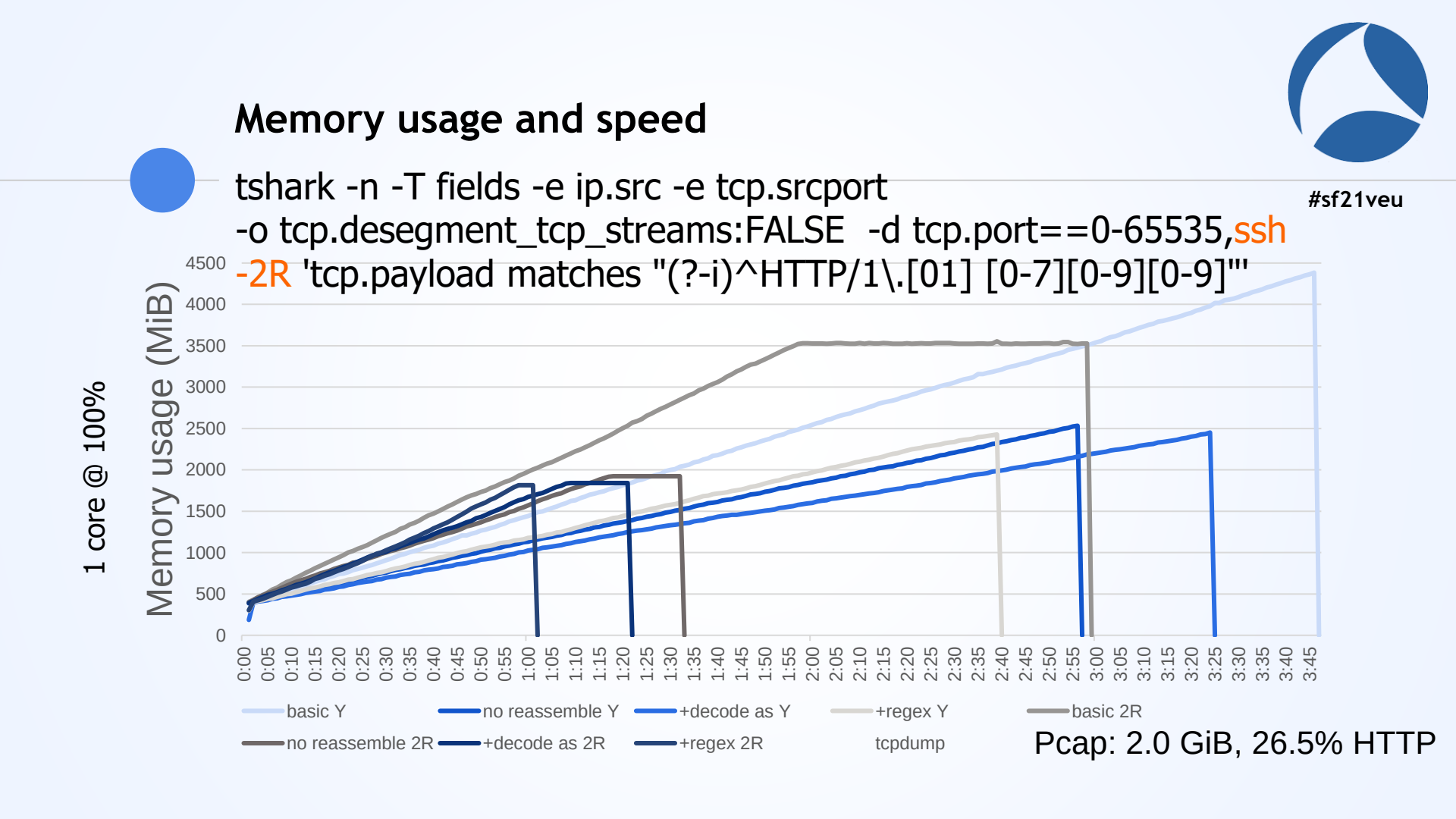
tshark -n -T fields -e ip.src -e tcp.srcport -2R http.response
-o tcp.desegment_tcp_streams:FALSE
-d tcp.port==0-65535,http

1 core @ 100%



Pcap: 2.0 GiB, 26.5% HTTP

Memory usage and speed



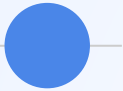
#sf21veu

tshark -n -T fields -e ip.src -e tcp.srcport
-o tcp.desegment_tcp_streams:FALSE -d tcp.port==0-65535,ssh
-2R 'tcp.payload matches "(?-i)^HTTP/1\.[01] [0-7][0-9][0-9]"'

1 core @ 100%

Pcap: 2.0 GiB, 26.5% HTTP

Memory usage and speed

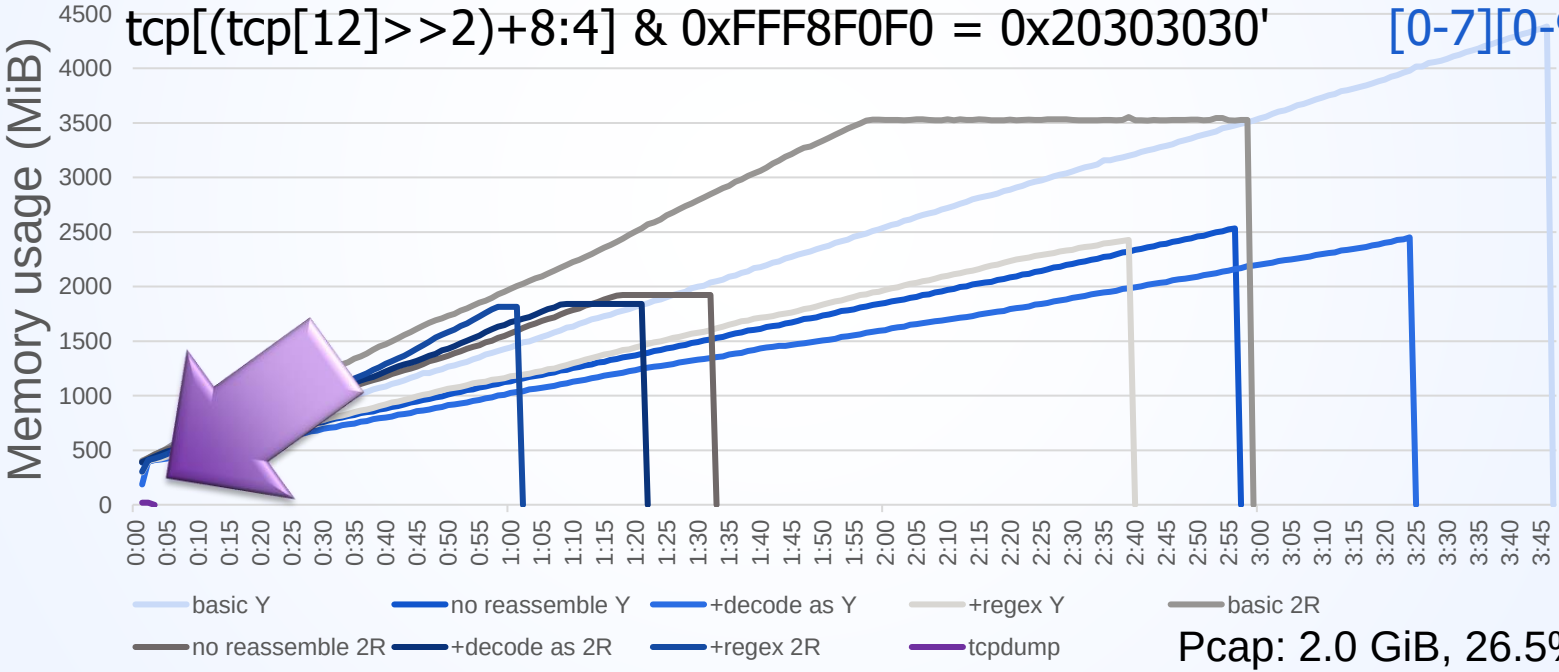


```
tcpdump -qnt -nn 'tcp[tcp[12]>>2:4] = 0x48545450 &&
tcp[(tcp[12]>>2)+4:4] & 0xFFFFFFFE = 0x2F312E30 &&
tcp[(tcp[12]>>2)+8:4] & 0xFFF8F0F0 = 0x20303030'
```

```
HTTP
/1\.[01]
[0-7][0-9][0-9]
```

#sf21veu

1 core @ 100% CPU



Pcap: 2.0 GiB, 26.5% HTTP

TShark versus tcpdump for pre-processing

TShark

+ Elaborate filtering options
+ Flexible output
+ Output statistics

– Slow
– Memory hog
– File size limited by available memory

Tcpdump

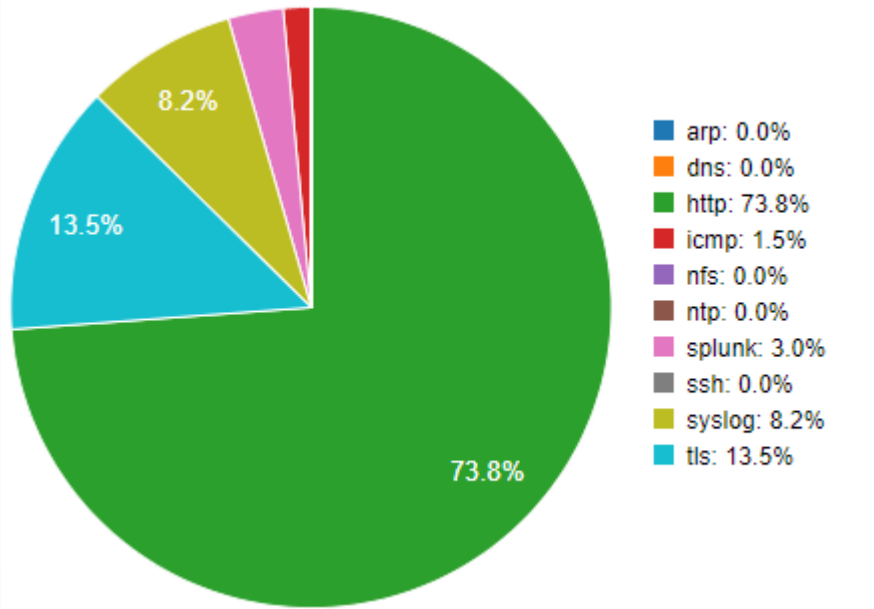
BPF: frame by frame filtering:

+ Fast
+ Unlimited file size
+ Small memory footprint

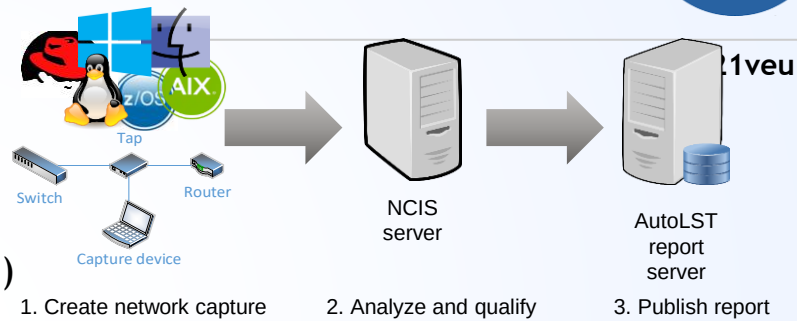
– Limited output options
– Pcap and pcapng only

Graph protocols

Example:



How did *we* automate

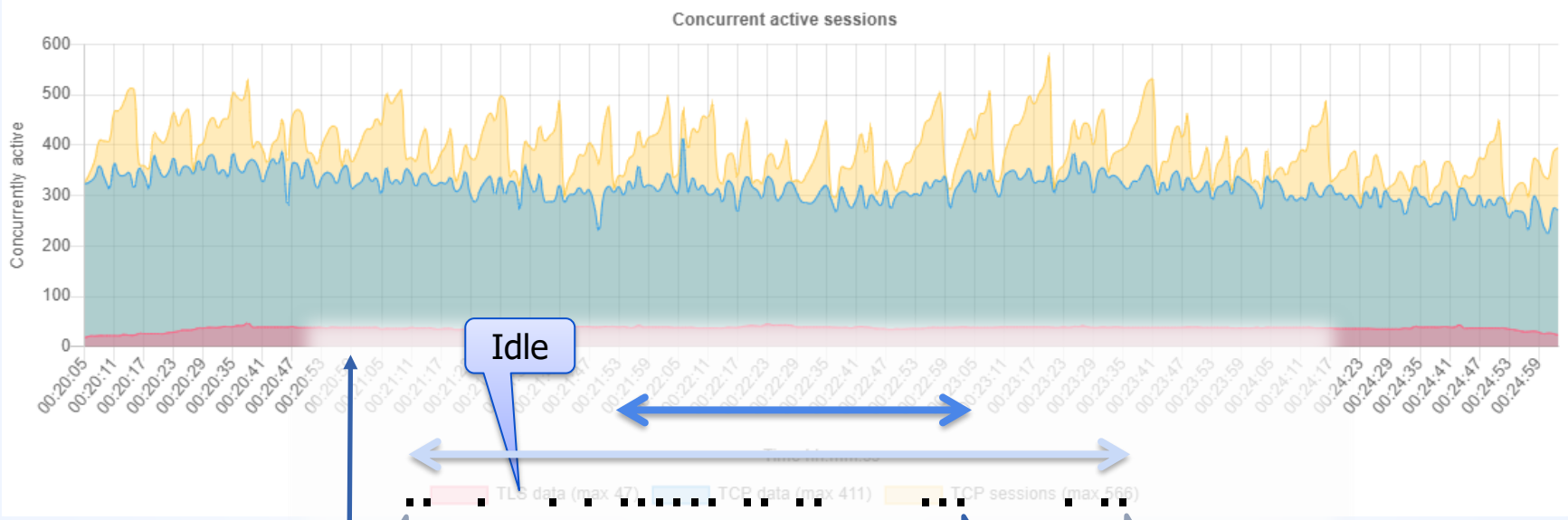


Steps:

1. Upload pcap (can be automated)
2. Sanity check
   - Reject: corrupted files, too many snapped packets, too low traffic volume, high dropped packet rate
3. Split in known protocols/applications – simplifies analysis
4. Process individual protocols/applications
   - Filter out incomplete streams, process stream by stream
5. Send results to report server (JSON format)

Plot concurrent sessions

#sf21veu



#sf21veu

Plot concurrent sessions (1)

```
#!/bin/bash
tshark -r "$1" -n2R tcp -d tcp.port==0-65535,ssh -T fields
   -e tcp.stream -e frame.time_epoch -e tcp.len |
sort -k 1,1n -k 2,2n |
awk 'BEGIN { tcp = -1; OFS=FS="\t"; RS="\r?\n";
     timemin = 1e11; timemax = 0 }
   $1 != tcp { new_tcp(); tcp = $1; start=$2; datastart=0 }
   $3 > 0 { # if tcp.len > 0
     if (datastart == 0) datastart = $2
     dataend = $2 }
   { end = $2 }
```

```
3   1547594405.013646000   0
3   1547594405.013658000   0
3   1547594405.014117000   0
3   1547594405.014124000   1460
3   1547594405.014134000   0
3   1547594405.014147000   2
3   1547594405.014150000   0
4   1547594405.014386000   1655
5   1547594405.015258000   0
5   1547594405.015266000   0
```

Plot concurrent sessions (2)


```
function new_tcp() { # note: all times in epoch format
  if (tcp < 0) return
  end = int(end); start = int(start) # round down to sec
  for(i = start; i <= end; i++) arr[i]++ # fill sec array
  if (timemin > start) timemin = start
  if (timemax < end  ) timemax = end
  if (datastart != 0) {
    dataend = int(dataend); datastart = int(datastart)
    for(i = datastart; i <= dataend; i++) darr[i]++ } }
END {
  new_tcp(); print "Time", "TCP sessions", "TCP data"
  for (idx = timemin; idx <= timemax; idx++)
    print strftime("%T", idx), arr[idx], darr[idx]}'
```

#sf21veu

Plot sessions summary

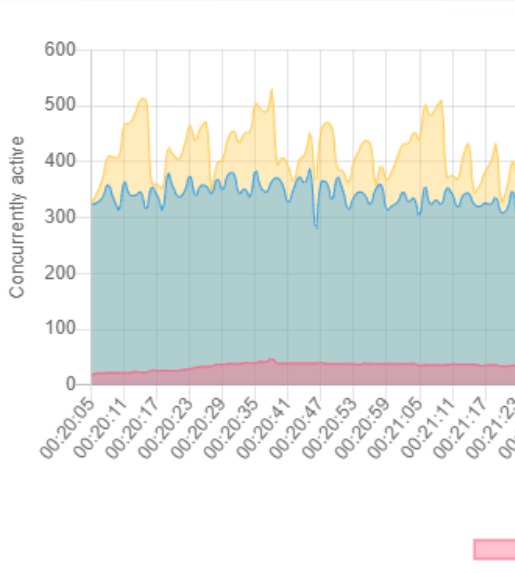
From sorted TShark output

```
3   1547594405.013646000    0
3   1547594405.013658000    0
3   1547594405.014117000    0
3   1547594405.014124000    1460
3   1547594405.014134000    0
3   1547594405.014147000    2
3   1547594405.014150000    0
4   1547594405.014386000    1655
5   1547594405.015258000    0
5   1547594405.015266000    0
5   1547594405.015478000    0
5   1547594405.015485000    2018
5   1547594405.015492000    0
6   1547594405.015593000    0
6   1547594405.015600000    0
7   1547594405.015696000    2211
6   1547594405.016513000    0
6   1547594405.016528000    1972
```

to data table

Time	TCP sessions	TCP data
00:20:05	304	302
00:20:06	318	301
00:20:07	342	309
00:20:08	381	330
00:20:09	379	307
00:20:10	380	288
00:20:11	438	334
00:20:12	436	308
00:20:13	452	306
00:20:14	480	314
00:20:15	477	285
00:20:16	329	319
00:20:17	315	296
00:20:18	309	282
00:20:19	379	341
00:20:20	368	319
00:20:21	362	303

and to the graph



TLS (encryption) statistics

Determining ratio resumed versus full handshakes

- For best performance use resumed handshakes
- Setting up TLS is difficult
  - A server falling back to full handshakes often goes unnoticed until performance related incidents occur
  - Typically such change is not visible in application log
  - Example: Java upgrade at client side only enabling TLS Extended Master Secret extension (RFC 7627)

Embedding decryption secrets in a pcapng file

- Embed the SSLKEYLOGFILE in a pcapng file
- Easier to distribute capture files
  - Or automate analysis!
- No need to update TLS protocol preference

For example:

```
editcap --inject-secrets tls,keys.log in.pcap out.pcapng
```

Ratio resumed / full handshakes

```
tshark -z 'io,stat,0,COUNT(tls.resumed)tls.resumed,
  COUNT(tls.handshake.type)tls.handshake.type==1'
  -nq -r tls.pcap
```

```
==========================================================
| IO Statistics                                          |
|                                                        |
| Duration: 504.5 secs                                   |
| Interval: 504.5 secs                                   |
|                                                        |
| Col 1: COUNT(tls.resumed)tls.resumed                   |
|     2: COUNT(tls.handshake.type)tls.handshake.type==1  |
|--------------------------------------------------------|
| Interval       | COUNT | COUNT |                       |
|--------------------------------|                       |
|   0.0 <> 504.5 |   770 |  7291 |                       |
```

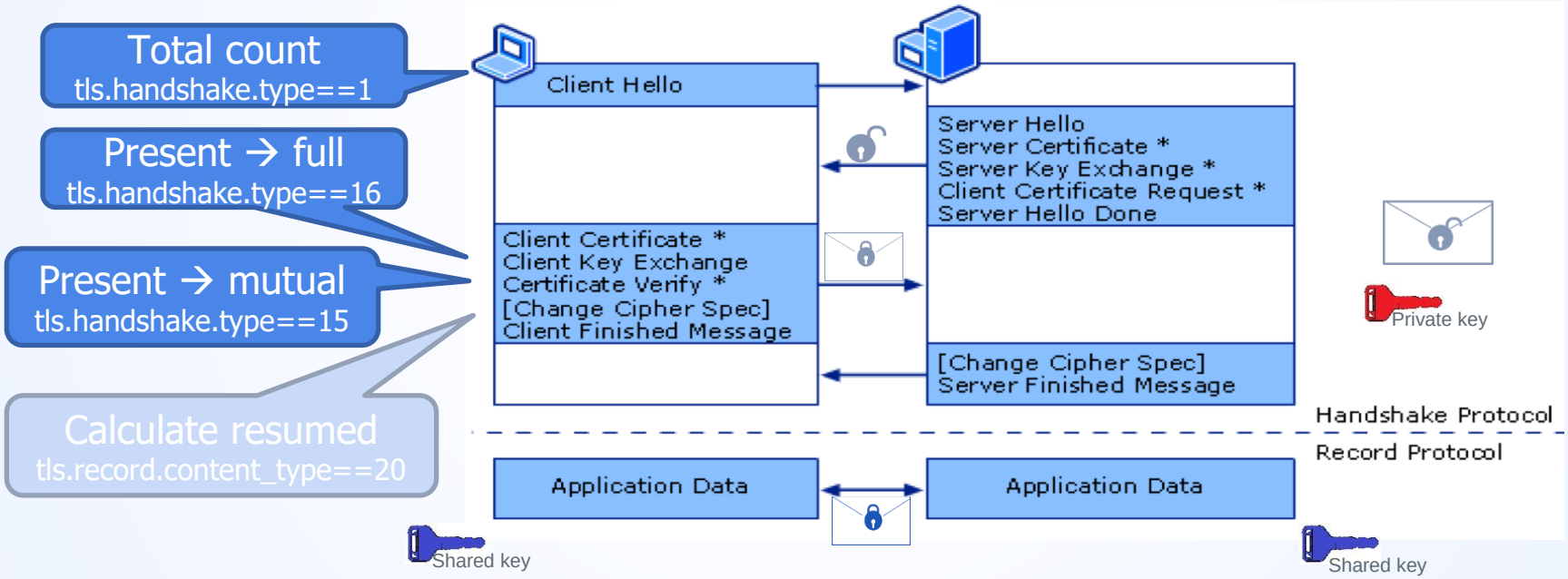
Resumed, since v3.0.0

ClientHello

```
awk -F '[| ]+' '/<>/ { printf "%.4g%%\n", $5/$6*100 }'
```

TLS statistics

Ratio resumed/full/mutual TLS handshakes (<v3.0)



TLS handshakes

Normally encrypted

tls

Src port	Dst port	Protocol	TCP stream	Bytes in Flight	Length	Info
42739	443	TLSv1	0	95	161	Client Hello
443	42739	TLSv1	0	1448	1514	Server Hello
443	42739	TLSv1	0	905	971	Certificate, Server Hello Done
42739	443	TLSv1	0	267	333	Client Key Exchange
42739	443	TLSv1	0	43	109	Change Cipher Spec, Finished
443	42739	TLSv1	0	43	109	Change Cipher Spec, Finished
42739	443	TLSv1	0	222	288	[TLS segment of a reassembled PDU]
42739	443	HTTP/XML	0	1521	1365	POST /ta/ HTTP/1.1
443	42739	TLSv1	0	25	91	Hello Request
42739	443	TLSv1	0	123	189	Client Hello
443	42739	TLSv1	0	1448	1514	Server Hello
443	42739	TLSv1	0	2648	1514	Certificate [TCP segment of a reassembled
443	42739	TLSv1	0	311	377	Certificate Request, Server Hello Done
42739	443	TLSv1	0	4027	1197	Certificate, Client Key Exchange
42739	443	TLSv1	0	283	349	Certificate Verify
42739	443	TLSv1	0	305	88	Change Cipher Spec
42739	443	TLSv1	0	342	103	Finished
443	42739	TLSv1	0	59	125	Change Cipher Spec, Finished
443	42739	TLSv1	0	1507	1514	[TLS segment of a reassembled PDU]
443	42739	HTTP/XML	0	384	450	HTTP/1.1 200 OK

Count handshake

Full handshake

Renegotiate request due to Step-Up Authentication

Count handshake

Mutual handshake

TLS ciphers used

- ClientHello contains ciphers supported
- ServerHello contains the selected cipher

```
-Y tls.handshake.type==2 -e tls.handshake.ciphersuite
```

Outputs decimal number

Create conversion table:

```
curl -k https://www.iana.org/assignments/tls-parameters/tls-parameters-4.csv | awk -F '[",]+'
'/^"0x..,0x..",[^RU]/ { print $2 substr($3, 3) " =>
\"" $4 "\","; c++ }
END { print "# Count:", c }'
```

```
0x0000 => "TLS_NULL_WITH_NULL_NULL",
0x0001 => "TLS_RSA_WITH_NULL_MD5",
0x0002 => "TLS_RSA_WITH_NULL_SHA",
0x0003 => "TLS_RSA_EXPORT_WITH_RC4_40_MD5",
```

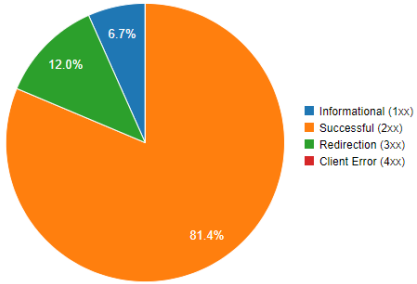
#sf21veu

HTTP statistics

- Take a look at `tshark -z http,stats`
- And other `-z http`…

HTTP responses

HTTP rsponses by type



HTTP version

HTTP versions used by category

Request	Response	Version	Count	Percent
DELETE		HTTP/1.1	1093	1.12%
GET		HTTP/1.1	76990	78.7%
HEAD		HTTP/1.1	1207	1.23%
OPTIONS		HTTP/1.1	1204	1.23%
POST		HTTP/1.1	16204	16.6%
PUT		HTTP/1.1	1081	1.11%
	100 Continue	HTTP/1.1	6963	6.65%
	200 OK	HTTP/1.1	85168	81.4%
	302 Found	HTTP/1.1	12555	12%
	405 Method Not Allowed	HTTP/1.1	4	0.00382%

HTTP stats in Perl (1)


```
my $command = "tshark -r '$file' -n -T Fields "
   . " -e http.request.version -e http.request.method"
   . "  -e http.response.version -e http.response.code"
   . "  -e http.response.phrase"
   . " -Y 'http.request||http.response'";
my %stats; # init associative array
open(my $fh, "-|", $command) or die "Command failed $!";
while (<$fh>) { chomp;
   my ($cver, $cmethod, $sver, $scode, $stext) = split "\t";
   $stats{requests}{"$cver $cmethod"}++ if $cver ne "";
   $stats{responses}{"$sver $scode $stext"}++ if $sver ne
   "";
} close($fh);
```

HTTP stats in Perl (2)

```
# print results
foreach my $id(sort keys %stats) {
   my $tot = 0;
   foreach (values %{$stats{$id}}) { $tot += $_ }
print "$id:\n";
foreach my $ind(sort keys %{$stats{$id}}) {
    printf "%#.3g%% %6d %s\n",
        stats{$id}{$ind}/$tot*100,
        $stats{$id}{$ind}, $ind
}
}
```

```
Requests:
0.00511%      5 HTTP/1.0 GET
0.00511%      5 HTTP/1.0 POST
1.12%    1093 HTTP/1.1 DELETE
78.7%   76985 HTTP/1.1 GET
1.23%    1207 HTTP/1.1 HEAD
1.23%    1204 HTTP/1.1 OPTIONS
16.6%   16199 HTTP/1.1 POST
1.11%    1081 HTTP/1.1 PUT
Responses:
0.00478%      5 HTTP/1.0 200 OK
6.65%    6963 HTTP/1.1 100 Continue
81.3%   85163 HTTP/1.1 200 OK
12.0%   12555 HTTP/1.1 302 Found
0.00382%    4 HTTP/1.1 405 Method Not Allowed
```

Lua script

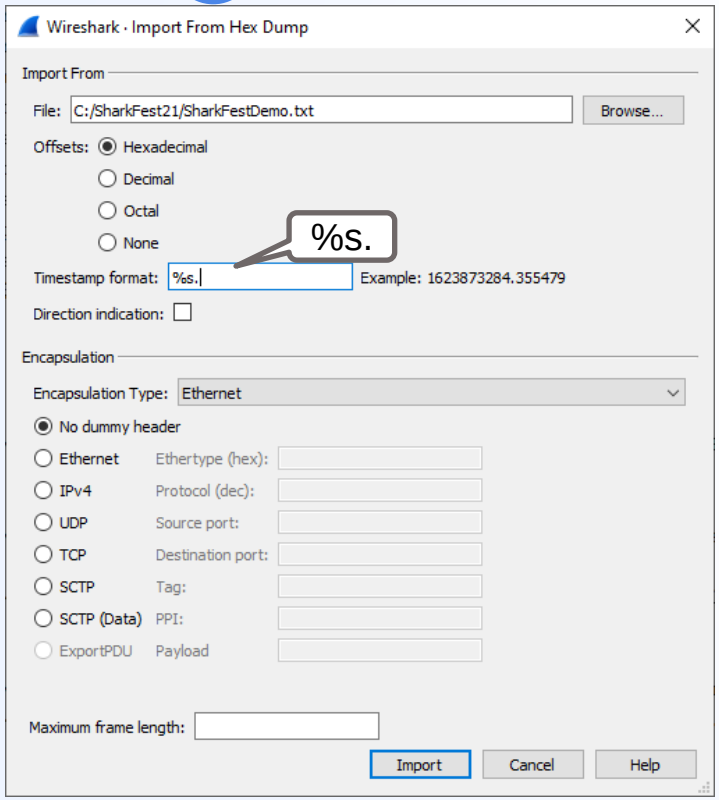
- Lua is a light-weight programming language designed for extending applications
- To analyse special / propriety cases

- Demo

```
tshark -X lua_script:http_len.lua
-o http.dechunk_body:FALSE -o http.decompress_body:FALSE
-o tcp.desegment_tcp_streams:TRUE
```

#sf21veu

Pcap2text using TShark



- Create a profile, name it ‘Disabled’
- Add file ‘disabled_protos’, contents “eth”
- `tshark -r sf21demo.pcapng -t e -o gui.column.format:'"Time","%t"' -P -x -C Disabled > SharkFestDemo.txt`

- Or use: `-d tcp.port==0-65535,ssh`

Thank you

#sf21veu

Thank you. Questions?

Cygwin on Windows

```
$ tshark -Tfields -e frame.time -c 3 -r tz.pcap
Jun 12, 2021 09:54:53.092532000 ope
Jun 12, 2021 09:54:53.105981000 ope
Jun 12, 2021 09:54:53.106076000 ope
$ TZ= tshark -Tfields -e frame.time -c 3 -r tz.pcap
Jun 12, 2021 10:54:53.092532000 W. Europe Summer Time
Jun 12, 2021 10:54:53.105981000 W. Europe Summer Time
Jun 12, 2021 10:54:53.106076000 W. Europe Summer Time
```

Cause: running Windows native executable using ‘unknown’ TZ value, is not understood by localtime function → falls back to UTC

Fix: unset TZ variable: `alias tshark="TZ= tshark"` or `TZ=UTC`