Automate your analysis

TShark, the Swiss army knife

André Luyer
Rabobank

#sf19eu • Palacio Estoril Hotel, Estoril, Portugal • Nov 4 - 8
About me?
Topics

• Why did we automate analysis
• How did we automate
  • Overview of the tool
  • Design concepts & lessons learned
• Available command line options
• Coding samples
  • How-to’s
Area of expertise

Network knowledge

Coding skills

Today’s topic
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
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 url

Global (all combined) summary
Traffic by type graph.

Result summary

<table>
<thead>
<tr>
<th></th>
<th>positive</th>
<th>minor</th>
<th>major</th>
<th>critical</th>
<th>undetermined</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occurrence</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
TShark output options

Most used text output options:

1. `-T fields -e field ...`
   Line based output, selected fields only, tab separated

2. `-O filter`
   Multi line tree view, context of fields

3. `-z io,stat,interval,filter,... -q`
   Statistics per interval, for selected filter(s)
Generate hosts file from pcap

Nowadays reverse DNS lookups does 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

Solution:
Get the hostnames actually used out of the pcap file itself and launch tshark with ‘-H <input hosts file>’ option or save as the personal hosts file for wireshark
Generate hosts file from pcap

```bash
tshark -r filename.pcap -H hosts.txt -w file-with-hosts.pcapng -P -F pcapng -W n
```
Most used text output options:

1. `-T fields -e field ...`
   Line based output, selected fields only, tab separated

2. `-O filter`
   Multi line tree view, context of fields

3. `-z io,stat,interval,filter,... -q`
   Statistics per interval, for selected filter(s)
TShark -O example

```
tshark -r tls.pcapng -O tls -2R tls.handshake.type==11 -c 1 |
grep -A3 notBefore
```

```
notBefore: utcTime (0)
  utcTime: 16-07-25 07:54:21 (UTC)

notAfter: generalizedTime (1)
  generalizedTime: 2050-06-02 07:54:21 (UTC)
```

```
x509af.utcTime
  utcTime: 17-12-09 13:54:53 (UTC)
```

```
x509af.generalizedTime
  generalizedTime: 2050-06-02 07:54:21 (UTC)
```

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

Which is before and after?
TShark output options

Most used text output options:

1. **-T fields -e field ...**
   - Line based output, selected fields only, tab separated

2. **-O filter**
   - Multi line tree view, context of fields

3. **-z io,stat,interval**
   - Statistics per interval
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


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

Pcap: 2.0 GiB, 26.5% HTTP
Memory usage and speed

```
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.[0-7][0-9][0-9]""
```

Pcap: 2.0 GiB, 26.5% HTTP
Memory usage and speed

```
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
```

Pcap: 2.0 GiB, 26.5% HTTP
Memory usage and speed

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]\]"'

Pcap: 2.0 GiB, 26.5% HTTP
Memory usage and speed

```
tcp[(tcp[12]>>2)+4:4] & 0xFFFFFFFF = 0x2F312E30 &&
tcp[(tcp[12]>>2)+8:4] & 0xFFF8F0F0 = 0x20303030'
```

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

Pcap: 2.0 GiB, 26.5% HTTP
<table>
<thead>
<tr>
<th><strong>TShark</strong></th>
<th><strong>Tcpdump</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Elaborate filtering options</td>
<td>BPF: frame by frame filtering:</td>
</tr>
<tr>
<td>+ Flexible output</td>
<td>+ Fast</td>
</tr>
<tr>
<td>+ Output statistics</td>
<td>+ Unlimited file size</td>
</tr>
<tr>
<td></td>
<td>+ Small memory footprint</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Slow</td>
<td></td>
</tr>
<tr>
<td>Memory hog</td>
<td></td>
</tr>
<tr>
<td>File size <strong>limited</strong> by available memory</td>
<td></td>
</tr>
</tbody>
</table>
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)
Process stream by stream

To process data stream by stream, sort by stream & time

```
tshark -n -r filename.pcap -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
```

```
0  1528205116.256929000  172.18.224.71  56293 → 80 [SYN, ECN, CWR] Seq
0  1528205116.258545000  172.17.184.32  80 → 56293 [SYN, ACK] Seq=0 Ack=1
0  1528205116.258588000  172.18.224.71  56293 → 80 [ACK] Seq=1 Ack=1
0  1528205116.258769000  172.18.224.71  POST /cgi-bin/Sharkfest2019al
0  1528205116.258825000  172.18.224.71  56293 → 80 [ACK] Seq=354 Ack=1
0  1528205116.258826000  172.18.224.71  56293 → 80 [ACK] Seq=1734 Ack=354
```
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 = $tcp; }
  # handle fields here...
}
close($fh);
new_tcp_stream() if $tcpnr >= 0;
Filter out incomplete streams

```bash
#!/bin/bash
options="-o tcp.desegment_tcp_streams:FALSE -d tcp.port==0-65535,ssh"
output=$(
tshark -r "$1" $options -Tfields -2R "tcp.flags & 7"
-e tcp.stream -e frame.number -e tcp.flags.syn |
sort -k 1,1n -k 2,2n |
awk -F "\t" 'BEGIN { tcps = -1; filter = "" }'
{ if ($1 != tcps) { tcps = $1; syn = 0 } # new stream
  if ($3) syn = 1; else
    { if (syn) filter = filter " " $1 # else FIN|RST
      syn = 0 } }
END {if (filter!="") print "tcp.stream in {" filter "}" }')
tshark -r "$1" -w "$2" $options -2R "$output"
```
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)
Filter out incomplete streams

Optimize by using range operator, e.g.: tcp.stream in { 3..99 123 }
else { if (syn) arr[$1] = 1; syn = 0 } } # store in array

END { filter = ""

for (i = 0; i <= tcps; i++)

    if (i in arr) { # if complete TCP stream found

        j = i + 1; while (j in arr) 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

}

if (filter != "") print "tcp.stream in {" filter "}" }')

tshark -r "$1" -w "$2" $options -2R "$output"
Plot concurrent sessions

Concurrent active sessions

0 100 200 300 400 500 600
Concurrently active

Start TCP e.g. SYN

Start data tcp.len > 0

End data

END TCP e.g. FIN/ACK
# Plot concurrent sessions (1)

```bash
#!/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
4 1547594405.014386000 1655
5 1547594405.015258000 0
5 1547594405.015266000 0
```
function new_tcp() { # note: all times in epoch format
    if (tcp < 0) return
    end = int(end); start = int(start) # round down to seconds
    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]++ }
}

new_tcp(); print "Time", "TCP sessions", "TCP data"
for (idx = timemin; idx <= timemax; idx++)
    print strftime("%T", idx), arr[idx], darr[idx]}'
### Plot sessions summary

**From sorted TShark output to data table and to the graph**

<table>
<thead>
<tr>
<th>Time</th>
<th>TCP sessions</th>
<th>TCP data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:20:05</td>
<td>304</td>
<td>302</td>
</tr>
<tr>
<td>00:20:06</td>
<td>318</td>
<td>301</td>
</tr>
<tr>
<td>00:20:07</td>
<td>342</td>
<td>309</td>
</tr>
<tr>
<td>00:20:08</td>
<td>381</td>
<td>330</td>
</tr>
<tr>
<td>00:20:09</td>
<td>379</td>
<td>307</td>
</tr>
<tr>
<td>00:20:10</td>
<td>380</td>
<td>288</td>
</tr>
<tr>
<td>00:20:11</td>
<td>438</td>
<td>334</td>
</tr>
<tr>
<td>00:20:12</td>
<td>436</td>
<td>308</td>
</tr>
<tr>
<td>00:20:13</td>
<td>452</td>
<td>306</td>
</tr>
<tr>
<td>00:20:14</td>
<td>480</td>
<td>314</td>
</tr>
<tr>
<td>00:20:15</td>
<td>477</td>
<td>285</td>
</tr>
<tr>
<td>00:20:16</td>
<td>329</td>
<td>319</td>
</tr>
<tr>
<td>00:20:17</td>
<td>315</td>
<td>296</td>
</tr>
<tr>
<td>00:20:18</td>
<td>309</td>
<td>282</td>
</tr>
<tr>
<td>00:20:19</td>
<td>379</td>
<td>341</td>
</tr>
<tr>
<td>00:20:20</td>
<td>368</td>
<td>319</td>
</tr>
<tr>
<td>00:20:21</td>
<td>362</td>
<td>303</td>
</tr>
</tbody>
</table>
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)
tshark -z 'io,stat,0,COUNT(tls.resumed)
tls.resumed,COUNT(tls.handshake.type)
tls.handshake.type==1' -nq -r tls.pcap

Resumed, since v3.0.0

ClientHello
TLS statistics

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

Total count
`tls.handshake.type==1`

Present → full
`tls.handshake.type==16`

Present → mutual
`tls.handshake.type==15`

Calculate resumed
`tls.record.content_type==20`
# TLS handshakes

<table>
<thead>
<tr>
<th>Src port</th>
<th>Dst port</th>
<th>Protocol</th>
<th>TCP stream</th>
<th>Bytes in Flight</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>42739</td>
<td>443</td>
<td>TLSv1</td>
<td>0</td>
<td>95</td>
<td>161</td>
<td>Client Hello</td>
</tr>
<tr>
<td>443</td>
<td>42739</td>
<td>TLSv1</td>
<td>0</td>
<td>1448</td>
<td>1514</td>
<td>Server Hello</td>
</tr>
<tr>
<td>443</td>
<td>42739</td>
<td>TLSv1</td>
<td>0</td>
<td>905</td>
<td>971</td>
<td>Certificate, Server Hello Done</td>
</tr>
<tr>
<td>42739</td>
<td>443</td>
<td>TLSv1</td>
<td>0</td>
<td>267</td>
<td>333</td>
<td>Client Key Exchange</td>
</tr>
<tr>
<td>42739</td>
<td>443</td>
<td>TLSv1</td>
<td>0</td>
<td>43</td>
<td>109</td>
<td>Change Cipher Spec, Finished</td>
</tr>
<tr>
<td>42739</td>
<td>443</td>
<td>TLSv1</td>
<td>0</td>
<td>43</td>
<td>109</td>
<td>Change Cipher Spec, Finished</td>
</tr>
<tr>
<td>42739</td>
<td>443</td>
<td>TLSv1</td>
<td>0</td>
<td>222</td>
<td>288</td>
<td>[TLS segment of a reassembled PDU]</td>
</tr>
<tr>
<td>42739</td>
<td>443</td>
<td>HTTP/XML</td>
<td>0</td>
<td>1521</td>
<td>1365</td>
<td>POST /ta/ HTTP/1.1</td>
</tr>
</tbody>
</table>

- **Count handshake**
- **Full handshake**
- **Renegotiate request due to Step Up Authentication**
- **Count handshake**
- **Mutual handshake**

Normally encrypted
TLS ciphers used

- ClientHello contains ciphers supported
- ServerHello contains the selected cipher

-y tls.handshake.type==2 -e tls.handshake.ciphersuites

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 }
```

Outputs decimal number

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"
HTTP statistics

• Take a look at `tshark -z http,stats`
• And other `-z http...`
• Sample code behind this report
HTTP stats in Perl (1)

```perl
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 "";
}
```

#sf19eu • Palacio Estoril Hotel, Estoril, Portugal • Nov 4 - 8
```
# 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:
```
Thank you. Questions?

https://www.linkedin.com/in/andreluyer
Extra slides...
Wireshark command line

Did you know Wireshark has useful command line options too?
Read the online help or .html file in the program directory.

For example:

```sh
wireshark -r "my example.pcapng"
  -o tls.keylog_file:"my example.key"
  -Y "tls && http"
```

- open this file
- override config with this setting
- start with this display filter
capinfos
  • Shows properties of a pcap
editcap
  • Filter time frame from large pcap
  • Snap or chop packets
  • Deduplicate
mergecap
  • Sew pcaps together
dumpcap
  • Create network captures
SSLKEY capture & analyse
(Windows)

rem Capture with SSLKEYLOGFILE - AU Luyer - 2018-09-10
set timestamp=%DATE:/=-% %TIME::=-%
start /realtime "Dumpcap - stop with Control-C" ^
   "%ProgramFiles%\Wireshark\dumpcap" -q -i1 -i2 -w "trace-%timestamp%.pcap"
rem make sure the browser is not already running (in the background)...
taskkill /f /im chrome.exe
timeout 3

rem Set logfile. Must be absolute path!
set SSLKEYLOGFILE=%CD%\key-%timestamp%.log
start "Chrome-tls" "%ProgramFiles%\Google\Chrome\Application\chrome.exe" ^
   --disable-http2 https://sharkfesteurope.wireshark.org/

rem Using option tls.keylog_file allows for temporary use without altering
the configuration.
echo start "Wireshark" "%ProgramFiles%\Wireshark\wireshark.exe" ^
   -r "trace-%timestamp%.pcap" -o tls.keylog_file:"key-%timestamp%.log" ^
   -Y "tls && http" > "start-wireshark-%timestamp%.cmd"
#!/bin/bash
# Capture with SSLKEYLOGFILE - AU Luyer - 2018-09-10
timestamp=$(date +%F_%H-%M-%S)
pcapfile=trace_${timestamp}.pcap
keylogfile=keys_${timestamp}.log

sudo nice -n -18 dumpcap -B 16 -q -i any -w - > $pcapfile &
# -w - > == workaround "Permission denied" bug.
echo $!
sleep 3

SSLKEYLOGFILE=$(realpath $keylogfile)
firefox
https://sharkfest.wireshark.org/ &
# Logfile must be absolute path!

script=start_wireshark_${timestamp}.sh
echo "wireshark -r $pcapfile -o tls.keylog_file:$keylogfile -Y 'tls &
(http||http2)' &" > $script && chmod +rx $script
echo "Stop capture with: sudo pkill dumpcap"
Since Wireshark 3.0 you can embed the TLS key log file in a pcapng file. This makes it much easier to distribute capture files with decryption secrets, and makes switching between capture files easier since the TLS protocol preference does not have to be updated.

For example:

```
editcap --inject-secrets tls,keys.txt in.pcap out.pcapng
```
Cygwin on Windows

$ tshark -Tfields -e frame.time -c 3 -r tz.pcap
Jan 30, 2019 10:30:30.000227000 Eur
Jan 30, 2019 10:30:30.012122000 Eur
Jan 30, 2019 10:30:30.014504000 Eur

$ TZ= tshark -Tfields -e frame.time -c 3 -r tz.pcap
Jan 30, 2019 11:30:30.000227000 W. Europe Standard Time
Jan 30, 2019 11:30:30.012122000 W. Europe Standard Time
Jan 30, 2019 11:30:30.014504000 W. Europe Standard Time

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

Fix: unset TZ variable: alias tshark="TZ= tshark"
    or TZ=UTC
RX capture mechanism (simplified)

Host → Wire → NIC → Virtual switch → Virt. NIC → Rx Ring buffer → SoftIRQ → Drivers Eth/IP/TCP → Application

Virt. NIC

HardIRQ

ethtool -g eth0

tcpdump dumpcap

PCAP Buffer → User space → Disk IO → IO