

Wireshark Developer and User Conference

Hands-On Lab: Using Wireshark CLI Tools & Scripting

June 14, 2011

Sake Blok

Application Delivery Troubleshooter | SYN-bit
sake.blok@SYN-bit.nl

SHARKFEST '11

Stanford University
June 13-16, 2011

Hands-On Exercises

Get the exercise files from:

<http://www.SYN-bit.nl/files/sharkfest-2011.zip>

Agenda

- Introductions
- Why use CLI tools?
... and how?
- Wireshark CLI tools
- Useful shell commands
- Some Scripting Examples
- Q&A



Introductions

- In Networking since 1995
- Jobs of influence:
 - EuroNet, one of the first ISP's in The Netherlands
 - ABN/Amro bank, Routing, Switching, Loadbalancing
 - ION-IP, reseller of Alteon, F5, Cisco ACE
 - SYN-bit, my own company, troubleshooting, training and ADC consultancy (F5 iRules)
- Have been using ethereal/wireshark since 1999
- Developing for wireshark since 2006 (GUI, IP/TCP/HTTP/SSL, bug fixes)

Why use the CLI tools?

- When GUI is not available (shell access)
- Quick and Easy Analysis
- Postprocessing results
 - GUI is powerful & interactive, but fixed functionality
 - CLI combined with other tooling is very flexible
- Automation

CLI not only when GUI is unavailable

How?

- What information do I need?
 - visualize your output
- What (raw) data sources do I have?
 - Know the output formats of your data sources
- What tools are available?
 - What can they do, browse through manpages for unknown options

Practice, Experiment & be Creative :-)

Wireshark CLI tools


- tshark
- dumpcap
- capinfos
- editcap
- mergecap
- rawshark
(not covered)



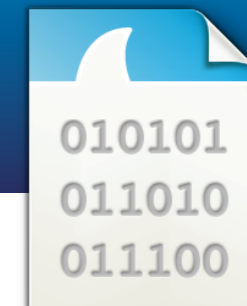
tshark (1)

- CLI version of wireshark
- Similar to tcpdump, but statefull / reassembly and MANY full protocol decodes
- uses dumpcap as capture engine
- standard options: -D, -i, -c, -n, -l, -f, -R, -s, -w, -r
- name resolving (-n)
- time stamps (-t<format>)
- decode as (-d tcp.port==8080,http)
- preferences (-o <pref>:<value>)

tshark (2)

- output formats (-V or -T <format>)
 - default: summary, uses column prefs
 - Verbose (-V), hex dump (-x), protocol selection (-O) 
 - PDML (-T pdml)
 - fields (-T fields -E <sep> -e <field1> -e <field2> ...)
- statistics (-z ...)
 - protocol hierarchy (-qz io,phs)
 - conversations (-qz conv,eth , -qz conv,tcp)
 - i/o statistics (-qz io,stat,10,ip,icmp,udp,tcp)

Exercise 1



http.cap

Using different output formats

- a) First use 'tshark -r http.cap' to show normal output
- b) Show full decodes (use 'tshark -r http.cap -V')
- c) Show PDML (XML) decodes (use '-T pdml')
- d) Do a, b and c again, but now pipe the output through the command wc (word count), like 'tshark -r http.cap | wc'. How much output is generated with each output format? How large was the file http.cap to begin with?

SHARKFEST

Exercise 2



port-1234.cap

Decoding protocols on non-standard ports with tshark ("Decode as...")

- a) Display the contents of the with tshark. What protocol is recognized for port 1234?
- b) Use the option '-x' to view hex/ascii output too. What protocol is transported over tcp port 1234?
- c) Now use 'tshark -r port-1234.cap -d tcp.port==1234,http' to decode tcp port 1234 as http. Is it possible to filter on http now?

SHARKFEST

Exercise 3



ssl.cap

Protocol preferences from the command line

- a) Display the contents of file ssl.cap with tshark, do you see http traffic?
- b) Use '-o ssl.keys_list:192.168.3.3,443,http,key.pem', do you see http traffic now?
- c) Which version of OpenSSL is used by the webserver (use '-V' and look at the "Server: <xxx>" http header)

SHARKFEST '11

Exercise 4



http.cap

Extracting interesting traffic to a new file

- a) Use tshark with option '-o tcp.desegment_tcp_streams:TRUE' and filter on http
- b) Now use tshark with option '-o tcp.desegment_tcp_streams:FALSE' and filter on http. How is this output different from the output in 4a?
- c) Do 4a and 4b again, but now use '-w' to write the output to 4a.cap and 4b.cap respectively. Read 4a.cap and 4b.cap with tshark, can you explain the difference?

SHARKFEST

Exercise 5



mail.cap

The tshark -z statistics

- a) Create a protocol hierarchy with '-qz io,phs', which protocols are present in the file?
- b) Create a ip conversation list with '-qz conv,ip'
- c) Create a tcp conversation list with '-qz conv,tcp'
- d) Create some io statistics with '-qz io,stat,60,ip,tcp,smtp,pop'
- e) Did the previous commands give you an overview of the contents of mail.cap?

SHARKFEST '11

dumpcap

- used by (wire|t)shark
... for privilege separation
- can be used separately
- options similar to tshark
- fast! only network->disk
- stateless! so traces can run forever
- ring buffer feature extremely useful:

```
dumpcap -i 5 -s0 -b filesize:16384 -files:1024 -w ring.cap
```

capinfos

- display summary of a tracefile
- all info vs specific info
- Or in table form with -T

```
$ capinfos example.cap
File name: example.cap
File type: Wireshark/tcpdump/... -
libpcap
File encapsulation: Ethernet
Number of packets: 3973
File size: 1431813 bytes
Data size: 1368221 bytes
Capture duration: 1299.436650 seconds
Start time: Thu Jan 17 11:37:16 2008
End time: Thu Jan 17 11:58:55 2008
Data rate: 1052.93 bytes/s
Data rate: 8423.47 bits/s
Average packet size: 344.38 bytes
```

```
$ capinfos -ae sharkfest-*.cap
File name: example.cap
Start time: Thu Jan 17 11:37:16 2008
End time: Thu Jan 17 11:58:55 2008

File name: sharkfest-2.cap
Start time: Thu Jan 17 11:39:27 2008
End time: Thu Jan 17 12:02:52 2008
```


editcap (1)

- used to **select** packets in a capture file

- select frame ranges or time ranges

```
editcap -r example.cap tmp.cap 1-1000 2001-3000
```

```
editcap -A "2008-01-17 11:40:00" \
```

```
    -B "2008-01-17 11:49:59" example.cap tmp.cap
```

- split file in chunks

```
editcap -c 1000 example.cap tmp.cap
```

```
editcap -i 60 example.cap tmp.cap
```

- remove duplicate packets

```
editcap -d example.cap tmp.cap
```

editcap (2)

- used to **change** (packets in) a capture file
 - change snaplen
`editcap -s 96 example.cap tmp.cap`
 - change timestamps
`editcap -t -3600 example.cap tmp.cap`
 - change link layer type
`editcap -T user0 example.cap tmp.cap`
 - change file type
`editcap -F ngsniffer example.cap tmp.cap`

SHARKFEST '11

mergcap

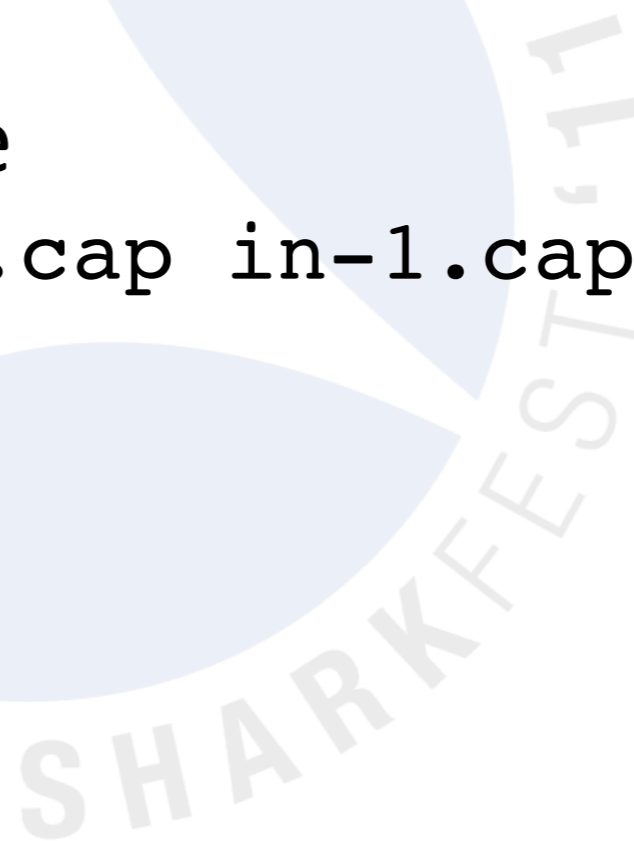
- used to merge capture files:

- based on timestamps

```
mergcap -w out.cap in-1.cap in-2.cap
```

- or just append each file

```
mergcap -a -w out.cap in-1.cap in-2.cap
```



Exercise 6



mail.cap

Splitting capture files with editcap

- a) Execute the command `'editcap -i 60 mail.cap tmp.cap'`. How many files are created?
- b) Use `'capinfos -Tcae tmp*'` to display a summary of these new files. Why are the timestamps not exactly 60 seconds apart?
- c) Remove the `'tmp*'` files
- d) Execute the command `'editcap -c 1000 mail.cap tmp.cap'`. How many files are created?
- e) Use `'capinfos -Tcae tmp*'` to display a summary of these new files.

Exercise 6 (continued)



tmp*.cap

Merging capture files with mergecap

- a) Use 'mergecap -w mail-new.cap tmp*'. Is the resulting file exactly the same as mail.cap?
(tip: use 'cmp <file1> <file2>')



Exercise 7



mail.cap

Adjusting timestamps with editcap

- a) Use 'editcap -t <delta>' to create a new tracefile (tmp.cap) where the first packet arrived exactly at 11:39:00 (tip: use '-V -c1' to see the exact timestamp of the first packet). What is your '<delta>'?
- b) What is the timestamp of the last packet in the new file? Are all packets adjusted with the same '<delta>'?

SHARKFEST

Getting Help

- Use “<command> -h” for options
... check once-in-a-while for new features
- Read the man-pages for in-depth guidance
(see: <http://www.wireshark.org/docs/man-pages/>)



Useful shell commands

- bash internals:
|, >, for ... do ... done, ``<command>``
- cut
- sort
- uniq
- tr
- sed
- awk
- scripting (sh/perl/python/...)

|, >, for ... do ... done

- Command piping with '|'

```
ls -lt | head
```

- Output redirection with '>'

```
ls -lt | head > 10-newest-files.txt
```

- Looping with for ... do ... done

```
for word in 'one' 'two' 'three'; do echo  
$word; done
```

SHARKFEST '11

`<command>`, variable assignments

- Command evaluation with backticks (` `)

```
for file in `ls -lt | head`  
do  
    echo $file  
    head -1 $file  
    echo ""  
done > firstlines.txt
```

- Variable assignments

```
backupfile=`echo ${file}.bak`
```

SHARKFEST '11

cut

- By character position (-c <range>)

```
cut -c1-10 /etc/passwd
```

- By field (-f<index> [-d '<delimiter>'])

```
cut -d ':' -f1 /etc/passwd
```



sort

- General alphabetic sort (no option)

```
sort names.txt
```

- Reverse sorting (-r)

```
sort -r names.txt
```

- Numerical (-n)

```
sort -n numbers.txt
```

- Or combined:

```
du -ks * | sort -rn | head
```

uniq

- De-duplication (no option)

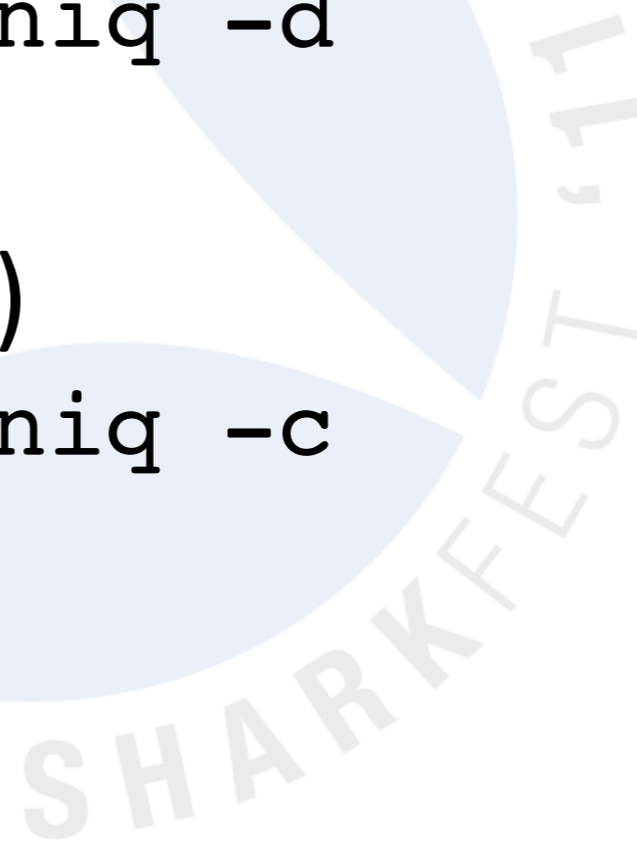
```
sort names.txt | uniq
```

- Show only 'doubles' (-d)

```
sort names.txt | uniq -d
```

- Count occurrences (-c)

```
sort names.txt | uniq -c
```



tr

- Translate a character(set)

```
echo "one two" | tr " " "_"
```

```
echo "code 217" | tr "[0-9]" "[A-J]"
```

```
echo "What is a house?" | tr "aeiou" "eioua"
```

- Delete a character(set)

```
echo "no more spaces" | tr -d " "
```

```
echo "no more vowels" | tr -d "aeiou"
```

```
cat dosfile.txt | tr -d "\015" > unixfile.txt
```

SHARKFEST '11

sed

- Stream editor
- Very powerful 'editing language'
- Some simple examples:
 - deleting text:
`sed -e 's/<deleteme>/'`
 - replacing text:
`sed -e 's/<replaceme>/<withthis>/'`
 - extracting text:
`sed -e 's/^\.*\(<keepme>\)\.*\(<andme>\)\.*$/\1 \2/'`

SHARKFEST '11

awk

- Pattern scanning and processing language
- Also a very powerful language
- Some simple examples:

```
netstat -an | \  
awk '$1~"tcp" {print $4}' | \  
sort | uniq -c
```

```
... | awk '{printf("%stcp.port==%s", sep, $1); sep=" || "}'
```

SHARKFEST '11

scripting

- parsing output when command piping is not enough
- automate execution of tshark/dumpcap/mergcap etc
- use your own favorite language (sh/perl/python/etc)

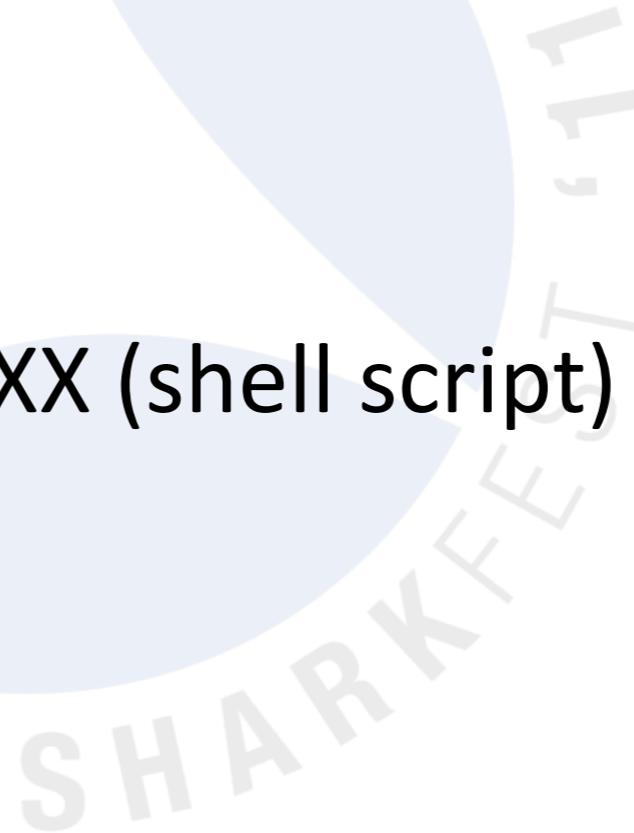
do anything you want :-)

Some Examples



example.cap

- Using command piping
 - Counting http response codes
 - Top 10 URL's
 - All TCP sessions which contain session-cookie XXXX
- Using scripting
 - All sessions for user XXXX (shell script)



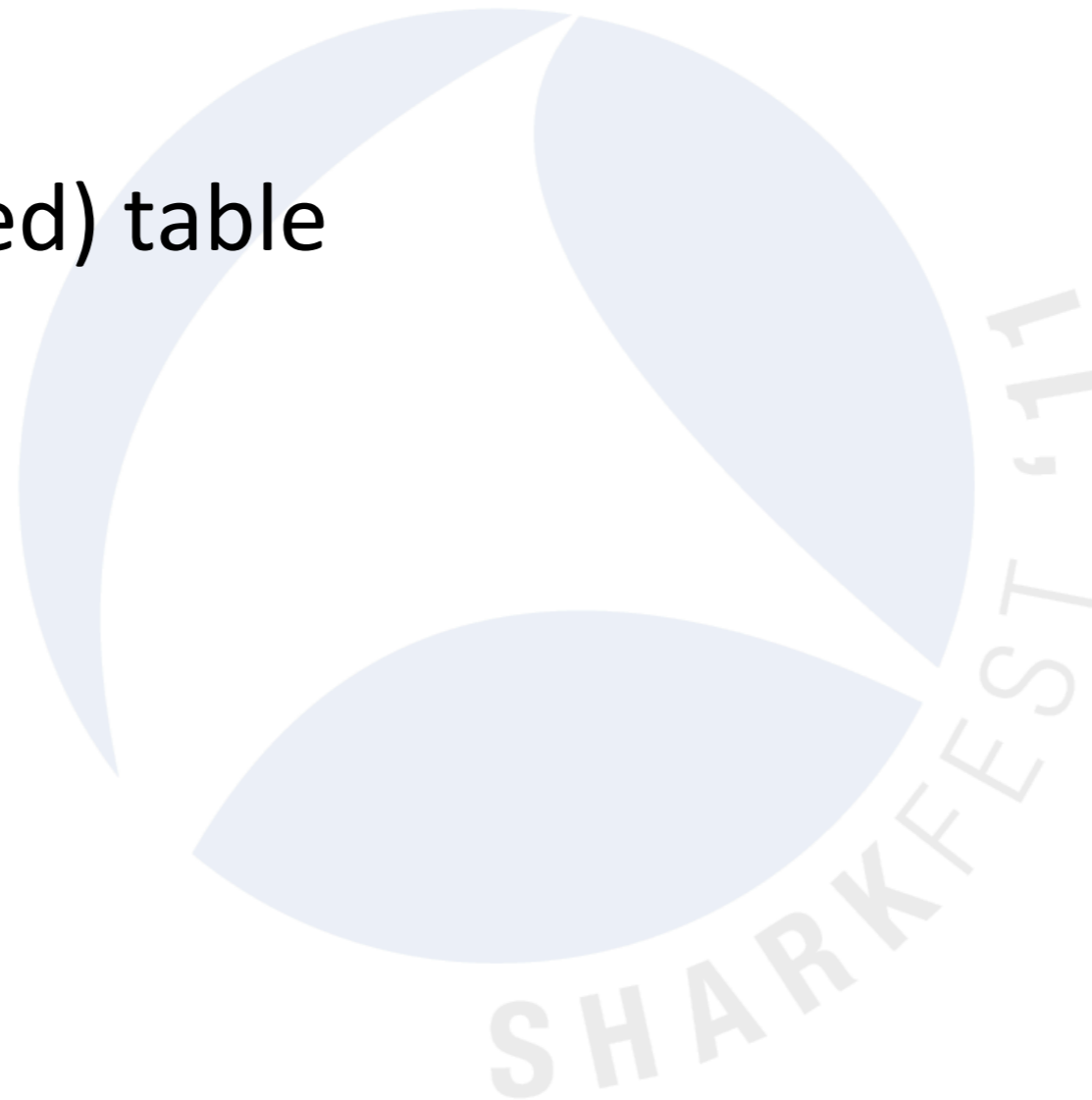
Example 1: Counting http response codes (1)

- Problem
 - I need an overview of http response codes
- Output
 - table with http response codes & counts
- Input
 - Capture file with http traffic



Example 1: Counting http response codes (2)

- Steps to take
 - print only http response code
 - count
 - make (sorted) table



Example 1: Counting http response codes (3)

- Command:

```
tshark -r example.cap -R http.response  
      -T fields -e http.response.code | \  
sort | uniq -c
```

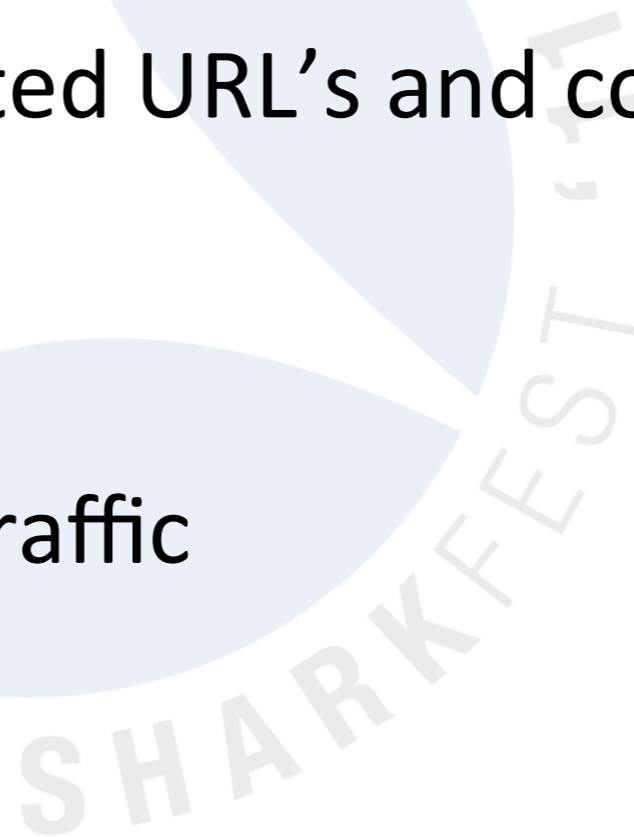
- New tricks learned:

```
-T fields -e <field>  
| sort | uniq -c
```



Example 2: Top 10 requested URL's (1)

- Problem
 - I need a list of all URL's that have been visited
- Output
 - Sorted list with requested URL's and count
- Input
 - Capture file with http traffic



Example 2: Top 10 requested URL's (2)

- Steps
 - Print `http.host` and `http.request.uri`
 - Strip everything after “?”
 - Combine host + uri and format into normal URL
 - count url's
 - make top 10



Example 2: Top 10 requested URL's (3)

- Command:

```
tshark -r example.cap -R http.request \  
  -T fields -e http.host -e http.request.uri | \  
sed -e 's/?.*$//' | \  
sed -e 's#^\(.*\) \|t\(.*\)$#http://\1\2#' | \  
sort | uniq -c | sort -rn | head
```

- New tricks learned:

remove unnecessary info : `sed -e 's/?.*$//'`

transform : `sed -e 's#^\(.*\) \|t\(.*\)$#http://\1\2#'`

top10 : `| sort | uniq -c | sort -rn | head`

Example 2: Top 10 requested URL's (3)

- Command:

```
tshark -r example.cap -R http.request
-T fields -e http.host -e http.uri
sed -e 's/?.*$//' | \
sed -e 's#^\(.*\) \|t\| \(.*\) $#http://\1\2#' | \
sort | uniq -c | sort -rn | head
```

- New Command:

```
unnecessary info : sed -e 's/?.*$//'
transform : sed -e 's#^\(.*\) \|t\| \(.*\) $#http://\1\2#'
top10 : | sort | uniq -c | sort -rn | head
```

http.request.full_uri

Example 3: All sessions with cookie XXXX (1)

- Problem

- I know in which “session” a problem exists, but I need all data from that session to work it out

- Output

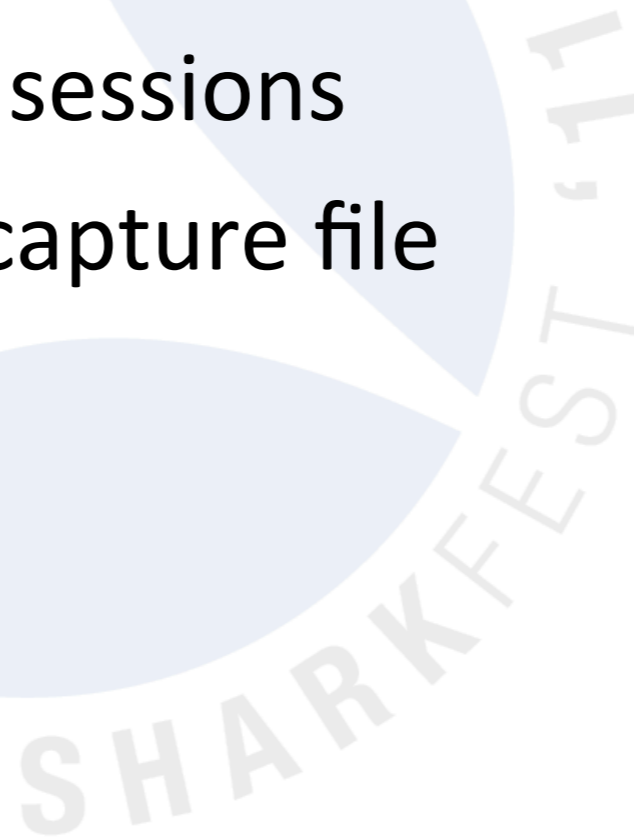
- New capture file with whole tcp sessions that contain cookie `PHPSESSID=c0bb9d04cebbc765bc9bc366f663fcf`

- Input

- Capture file with http traffic

Example 3: All sessions with cookie XXXX (2)

- Steps
 - select packets that contain the cookie
 - print the port numbers
 - create new filter based on port numbers
 - use filter to extract tcp sessions
 - save packets to a new capture file



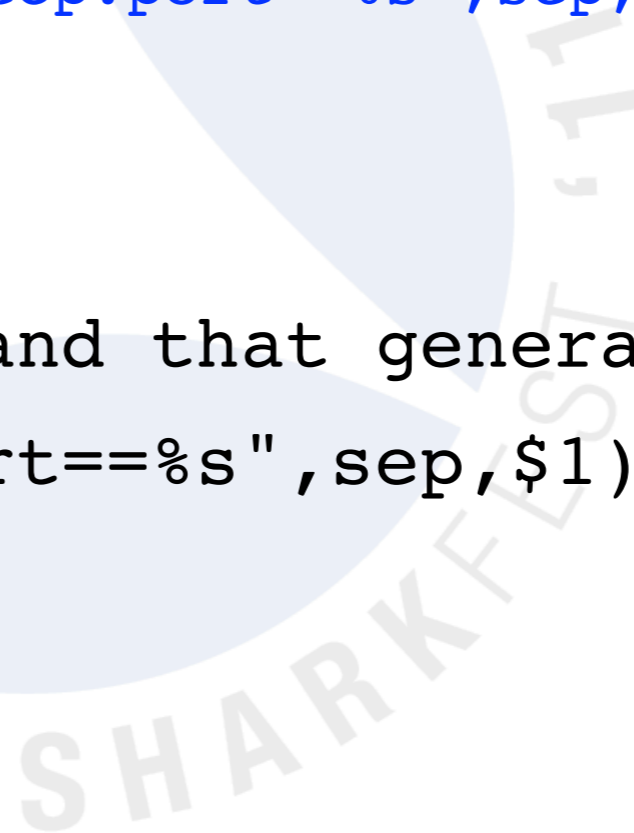
Example 3: All sessions with cookie XXXX (3)

- Command:

```
tshark -r example.cap -w cookie.cap \  
-R `tshark -r example.cap -T fields -e tcp.srcport  
-R "http.request and http.cookie contains \  
"PHPSESSID=c0bb9d04ceb765bc9bc366f663fcac\"" |\  
awk '{printf("%stcp.port==%s", sep, 1); sep="||"}'`
```

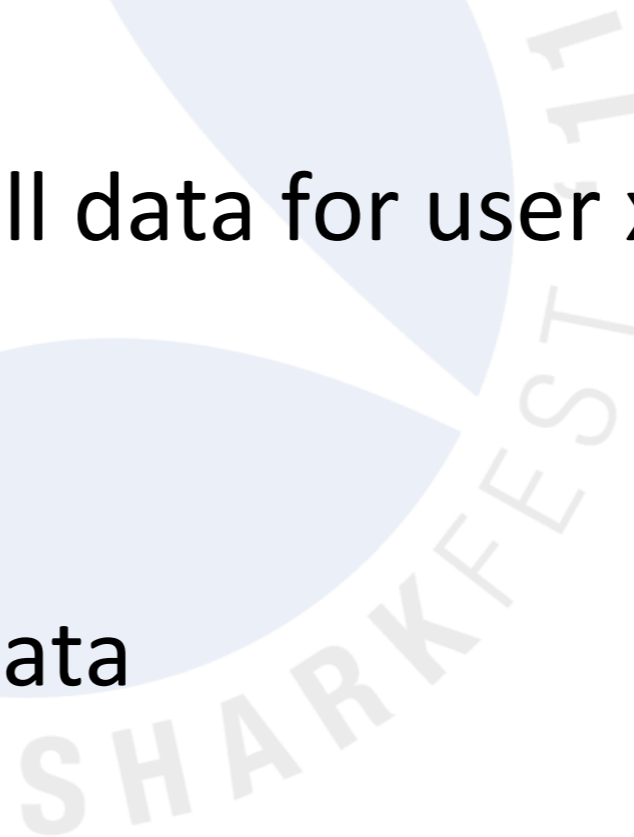
- New tricks learned:

```
tshark -R `  
awk '{printf("%stcp.port==%s", sep, $1); sep="||"}'`
```



Example 4: All sessions for user XXXX (1)

- Problem
 - A particular user has multiple sessions and I need to see all sessions from that user
- Output
 - New capture file with all data for user xxxx
- Input
 - Capture file with http data



Example 4: All sessions for user XXXX (2)

- Steps
 - print all session cookies for user XXXX
 - create new capture file per session cookie (see example 3)
 - merge files to new output file



Example 4: All sessions for user XXXX (3)

```
#!/bin/bash

file=$1
user=$2

for cookie in `tshark -r $file -R "http.request and http contains $user" -T
fields -e http.cookie | cut -d ' ' -f2`
do
    tmpfile="tmp_`echo $cookie | cut -d '=' -f 2`.cap"
    echo "Processing session cookie $cookie to $tmpfile"

    tshark -r $file -w $tmpfile -R `tshark -r $file -T fields -e tcp.srcport \
-R "http.request and http.cookie contains \"$cookie\" | \
awk '{printf("%stcp.port==%s",sep,$1);sep="||"}'`
done

mergcap -w $user.cap tmp_*.cap
rm tmp_*.cap
```

Example 4: All sessions for user XXXX (4)

- New tricks learned:

- `for ... do ... done`

- `<var>=`echo ... | ...``

- `cut -d <FS> -f <x>`

- `mergecap -w <outfile> <infile1> <infile2> ...`



Exercise 8



mail.cap

Create a new trace file for a specific pop user that contains only his pop sessions.

- a) First get an idea of a typical POP session, use :
`tshark -r mail.cap -R 'tcp.port==64315 and tcp.len>0'`
- b) Use the following steps to create a list of tcp ports used by user 'sake-test2':
 - Use the filter ' pop.request.parameter=="sake-test2" ' to only show sessions of user sake-test2
 - Add '-T fields -e tcp.srcport' to the command to just show the tcp ports.
 - Add | awk '{printf("%stcp.port==%s",sep,\$1);sep="||"}' to create a display filter that will only display packets belonging to the sessions for user sake-test2.

Exercise 8 (continued)

- c) Now use the output of the previous command between backticks to create the new file:

```
tshark -r mail.cap -w sake-test2.cap -R `  
previous  
command>`
```

- d) Use 'tshark -r sake-test2.cap -R pop.request.command==USER' to verify that the new file only contains sessions of user sake-test2. Did we succeed? What went wrong? How can we fix it?

Exercise 9



mail.cap

Creating a separate trace file for each pop user automatically.

- a) Delete the file sake-test2.cap
- b) Create a list of users with the following steps:
 - Use a filter to only select the packets where the pop command was “USER” and use '-T fields' to only print the username.
 - Use '| sort | uniq' to create a list of unique usernames

SHARKFEST

Exercise 9 (continued)

- c) Loop through the list of usernames and create the file per user with:

```
for user in `<command from 9b>`  
do  
    echo $user  
    <command from case 8c with $user as variable>  
done
```



Exercise 10 : Challenge!



mail.cap

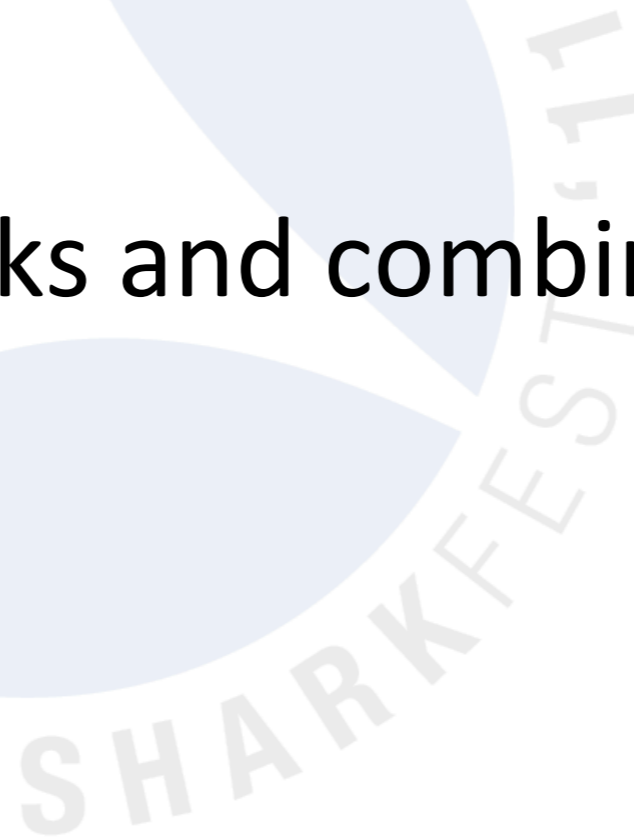
Create a shell script [or a one-liner ;-)] that produces the following output:

```
Mail check times for : sake-test1
11:39:43 : 1 message (2833 octets)
11:40:00 : 0 messages (0 octets)
11:42:33 : 7 messages (25958 octets)
11:45:04 : 6 messages (21538 octets)
11:47:37 : 5 messages (17480 octets)
11:50:09 : 8 messages (32297 octets)
11:52:40 : 5 messages (17017 octets)
11:55:13 : 6 messages (21075 octets)
11:57:46 : 6 messages (20859 octets)
12:00:28 : 7 messages (25416 octets)
12:02:49 : 1 message (3677 octets)

Mail check times for : sake-test2
11:39:44 : 5 messages (14512 octets)
11:40:01 : 6 messages (16811 octets)
11:42:34 : 5 messages (17568 octets)
11:45:05 : 4 messages (8551 octets)
11:47:38 : 6 messages (16337 octets)
11:50:10 : 2 messages (5396 octets)
11:52:42 : 7 messages (20601 octets)
11:55:14 : 5 messages (12089 octets)
11:57:46 : 4 messages (14463 octets)
12:00:22 : 5 messages (15016 octets)
12:02:50 : 4 messages (14805 octets)
```

Summary

- Wireshark comes with powerful CLI tools (tshark, dumpcap, capinfos, editcap, mergecap)
- tshark+scripting can complement GUI
- use little building blocks and combine them



Q&A



Thank you for listening!

e-mail:

sake.blok@SYN-bit.nl

