

SharkFest'17 US

the 10 Top Wireshark Tips & Tricks

Sample trace and supplemental files are located
<http://www.ikeriri.ne.jp/download/sharkfest>

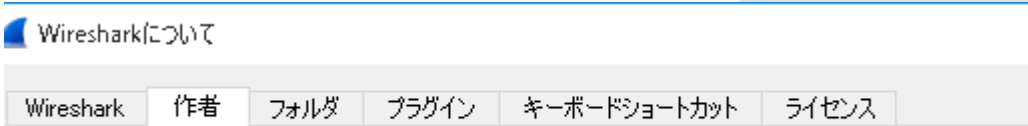
Megumi Takeshita

Packet Otaku | ikeriri network service co.,ltd

Megumi Takeshita, ikeriri network service a.k.a. packet otaku since first Sharkfest



- Founder, ikeriri network service co.,ltd
- Wrote 10+ books of Wireshark and capturing and network analysis.
- Reseller of Riverbed Technology (former CACE technologies) and Metageek, Dualcomm etc. in Japan
- Attending all Sharkfest 9 times
- Translator of QT Wireshark into Japanese



竹下 恵 (Megumi Takeshita)

<megumi@ikeriri.ne.jp>



23 Top 10 Wireshark TIPS & TRICKS

1. Collect only troubled IEEE802.11 packet
2. Use of Exclude Display filter
3. Stream ID and follow TCP in 2 way
4. Visualize using I/O Graph
5. Capture packet periodically and merge for stats
6. Sort and stack the pcap file automatically and divide by each filter value
7. Decrypting TLS/SSL data without key pair
8. HTTP2 dissection
9. QUIC dissection
10. Visualize using ErasticSearch + Kibana



#1. Collect only troubled IEEE802.11 packet

Wireless pcapng file contains **tons of other packets**

On capturing specified channel,

you can collect **only troubled IEEE802.11 packet**
such as Deauthentication and Disassociation.

The screenshot shows the Wireshark interface with a list of captured packets. The list includes several IEEE 802.11 Beacon frames from various sources. An orange callout box is overlaid on the right side of the interface, containing the text "5423 packet in just a minute".

No.	Time	Signal (dBm)	Source	Destination	Type/Subtype	Data rate	Protocol	Length
1	0.000000	-86	NecPlatf_80:97:af	Broadcast	Beacon frame	1	802.11	250
2	0.007007	-84	Logitec_7c:7b:7c	Broadcast	Beacon frame	1	802.11	297
3	0.009636	-84	Apple_ee:04:8e	Broadcast	Beacon frame	1	802.11	282
4	0.017504	-88	Buffalo_6d:68:50	Broadcast	Beacon frame	1	802.11	329
5	0.029577	-46	Modacom_a8:55:d9	Broadcast	Beacon frame	1	802.11	175
6	0.049608	-86	12:66:82:80:97:af	Broadcast	Beacon frame	1	802.11	57

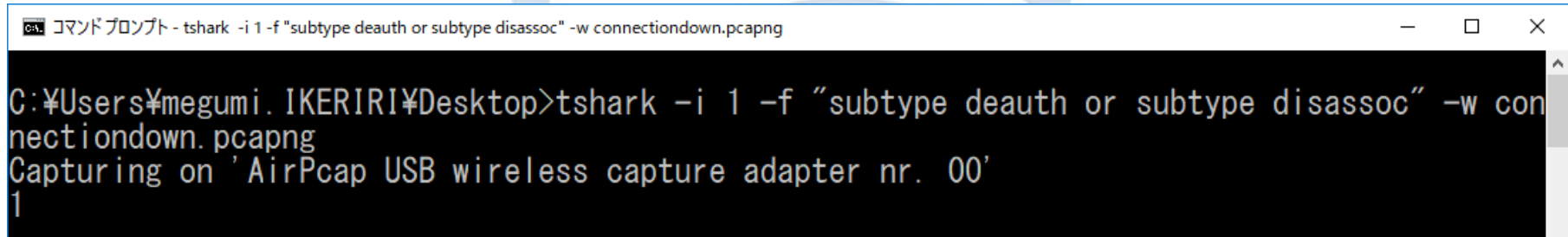
Retransmission flag (wlan.fc.retry), 1 バイト | パケット数: 5423 · 表示: 5423 (100.0%) · 読込時間: 0:0.161 | プロファイル無線LANセミナー準備用Wireshark設定

#1. Collect only troubled IEEE802.11 packet

When you collect Deauthentication and Disassociation,
wlan.fc.type_subtype == 12 or wlan.fc.type_subtype==14

No display filter but **capture filter** is good for wireless troubleshooting (especially in Tokyo 's crowded air)

tshark -i 1 -f "**subtype death or subtype disassoc**"
(<http://www.tcpdump.org/manpages/pcap-filter.7.txt>



```
コマンドプロンプト - tshark -i 1 -f "subtype death or subtype disassoc" -w connectiondown.pcapng
C:\Users\megumi.IKERIRI\Desktop>tshark -i 1 -f "subtype death or subtype disassoc" -w con
nectiondown.pcapng
Capturing on 'AirPcap USB wireless capture adapter nr. 00'
1
```

#1. Collect only troubled IEEE802.11 packet

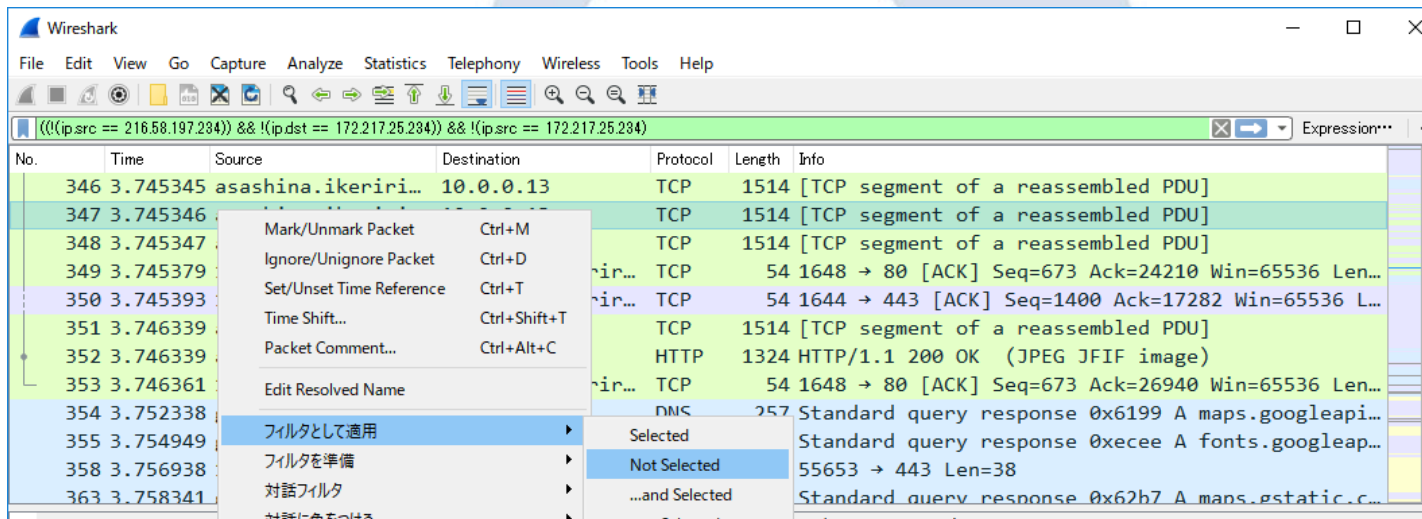
You can collect Deauthentication/Disassociation, regardless of knowing the WPA2 key (management frames are sent in clear text)
Using capture filter is also good for reducing pcapng size.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	37:bf:39:15:f0:aa	a9:65:28:a2:d9:36	802...	1283	Disassociate, SN=3943, FN=14, Flags=o..PR.FTC...
2	69.246...	6b:e2:d4:4f:64:47	18:42:20:c1:1b:e0	802...	1620	Fragmented IEEE 802.11 frame
3	71.955...	ea:fc:8f:5b:5a:8c	c4:5c:90:da:93:54	802...	1620	Deauthentication, SN=873, FN=1, Flags=op.PR....
4	84.342...	2b:b6:79:d8:fc:bd	13:cf:f5:b0:c0:92	802...	213	Fragmented IEEE 802.11 frame

> Frame 1: 1283 bytes on wire (10264 bits), 1283 bytes captured (10264 bits) on interface 0
> Radiotap Header v0, Length 20
> 802.11 radio information
> IEEE 802.11 Disassociate, Flags: o..PR.FTC
> IEEE 802.11 wireless LAN management frame
> [Malformed Packet: IEEE 802.11]

#2. Use the exclude Display filter

If you have no idea to find packet you needed,
Use the exclude Display filter is the easiest and
best way, and do not forget right click and choose
Apply filter > „... and not selected „

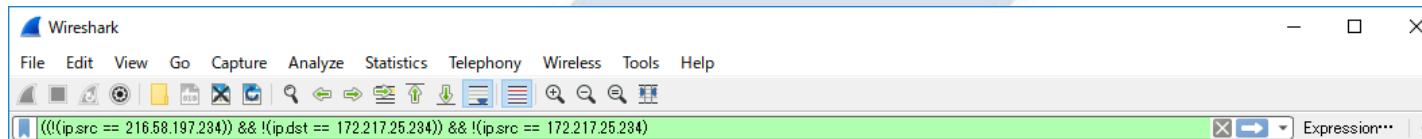


The screenshot shows the Wireshark interface with a display filter applied: `((!(ip.src == 216.58.197.234)) && !(ip.dst == 172.217.25.234)) && !(ip.src == 172.217.25.234)`. The packet list pane shows several packets, with packet 354 selected. A context menu is open over packet 354, showing options in Japanese and English. The 'Not Selected' option is highlighted.

No.	Time	Source	Destination	Protocol	Length	Info
346	3.745345	asashina.ikeriri...	10.0.0.13	TCP	1514	[TCP segment of a reassembled PDU]
347	3.745346			TCP	1514	[TCP segment of a reassembled PDU]
348	3.745347			TCP	1514	[TCP segment of a reassembled PDU]
349	3.745379		ir...	TCP	54	1648 → 80 [ACK] Seq=673 Ack=24210 Win=65536 Len...
350	3.745393		ir...	TCP	54	1644 → 443 [ACK] Seq=1400 Ack=17282 Win=65536 L...
351	3.746339			TCP	1514	[TCP segment of a reassembled PDU]
352	3.746339			HTTP	1324	HTTP/1.1 200 OK (JPEG JFIF image)
353	3.746361		ir...	TCP	54	1648 → 80 [ACK] Seq=673 Ack=26940 Win=65536 Len...
354	3.752338			DNS	257	Standard query response 0x6199 A maps.googleapi...
355	3.754949					Standard query response 0xecee A fonts.googleleap...
358	3.756938					55653 → 443 Len=38
363	3.758341					Standard query response 0x62h7 A mans.gstatic.c...

#2. Use the exclude Display filter

- The direction („ip.src“ or „ip.dst“) is determined by **the position of right click** in packet list pane.
- Applying exclude display filter many times...
(!(ip.src == 216.58.197.234)) && !(ip.dst == 172.217.25.234)) && !(ip.src == 172.217.25.234)...
- You may find the packets you have not recognized.
- Excluding MAC Address is also useable

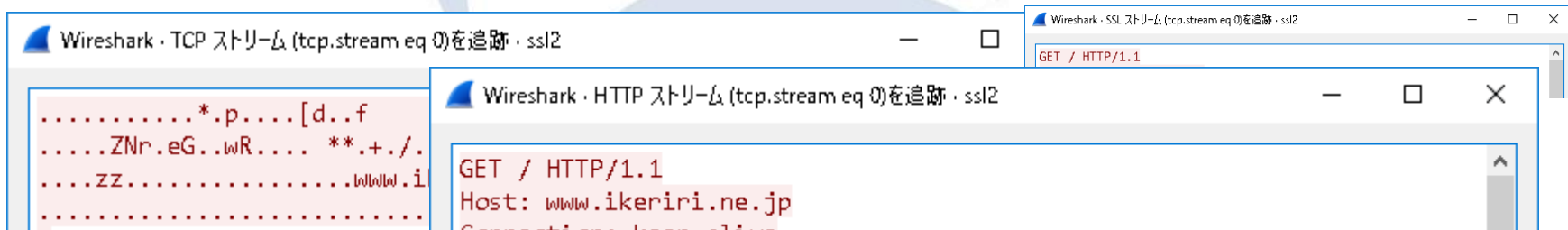


#2. Use the exclude Display filter Practice

1. Open wireless.pcapng file
2. Choose „Broadcast“ in Destination row in packet list pane, right click, **apply filter, „not selected“**
3. Choose „02:cc:37:11:8a:04“ in Destination row, right click, **apply filter, ... „and not selected“**
4. set WPA-PWD (pass phrase : SSID)
sharkfest:sharkfest-ikeriri in IEEE802.11 wireless LAN preferences, Decryption Keys...
5. Then we got the packets we wanted.

#3 Stream ID and follow TCP in 2 way

- Wireshark set sequence number of transaction of TCP, UDP, SSL and HTTP. For example Wireshark find first SYN packet, and set stream number.
- We can check flow using stream id using display filter.
tcp.stream udp.stream ssl.stream http.stream
- It is very useful to understand follow TCP stream



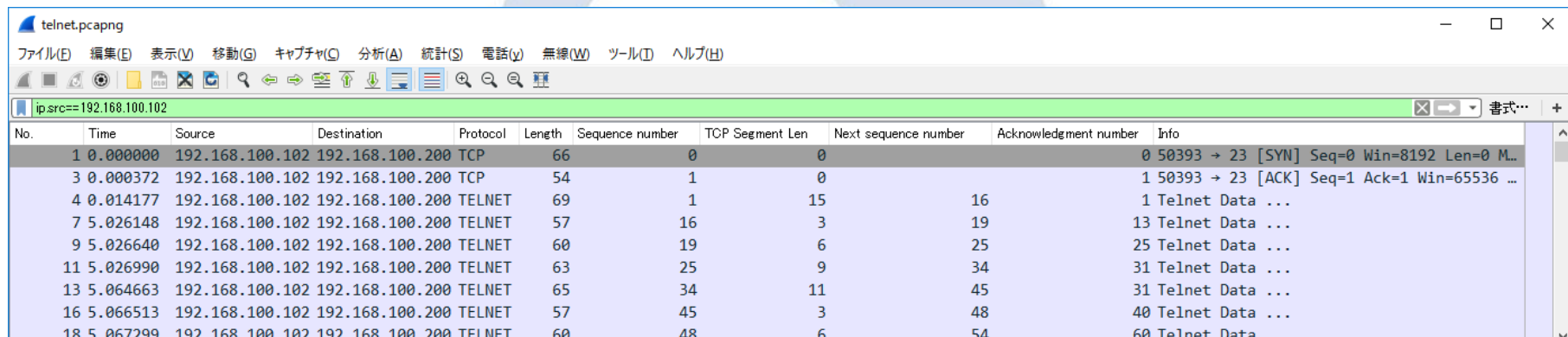
#3 Stream ID and follow TCP in 2 way

- It is also useful to add columns of TCP
Sequence no (tcp.seq)
TCP Segment Len (tcp.len)
Next sequence no (tcp.nxtseq)
Acknowledgement no (tcp.ack)

No.	Time	Source	Destination	Protocol	Length	Sequence number	TCP Segment Len	Next sequence number	Acknowledgment number	Info
1	0.000000	192.168.100.102	192.168.100.200	TCP	66	0	0			0 50393 →
2	0.000242	192.168.100.200	192.168.100.102	TCP	66	0	0			1 23 → 50393
3	0.000372	192.168.100.102	192.168.100.200	TCP	54	1	0			1 50393 →
4	0.014177	192.168.100.102	192.168.100.200	TELNET	69	1	15		16	1 Telnet [

#3 Stream ID and follow TCP in 2 way

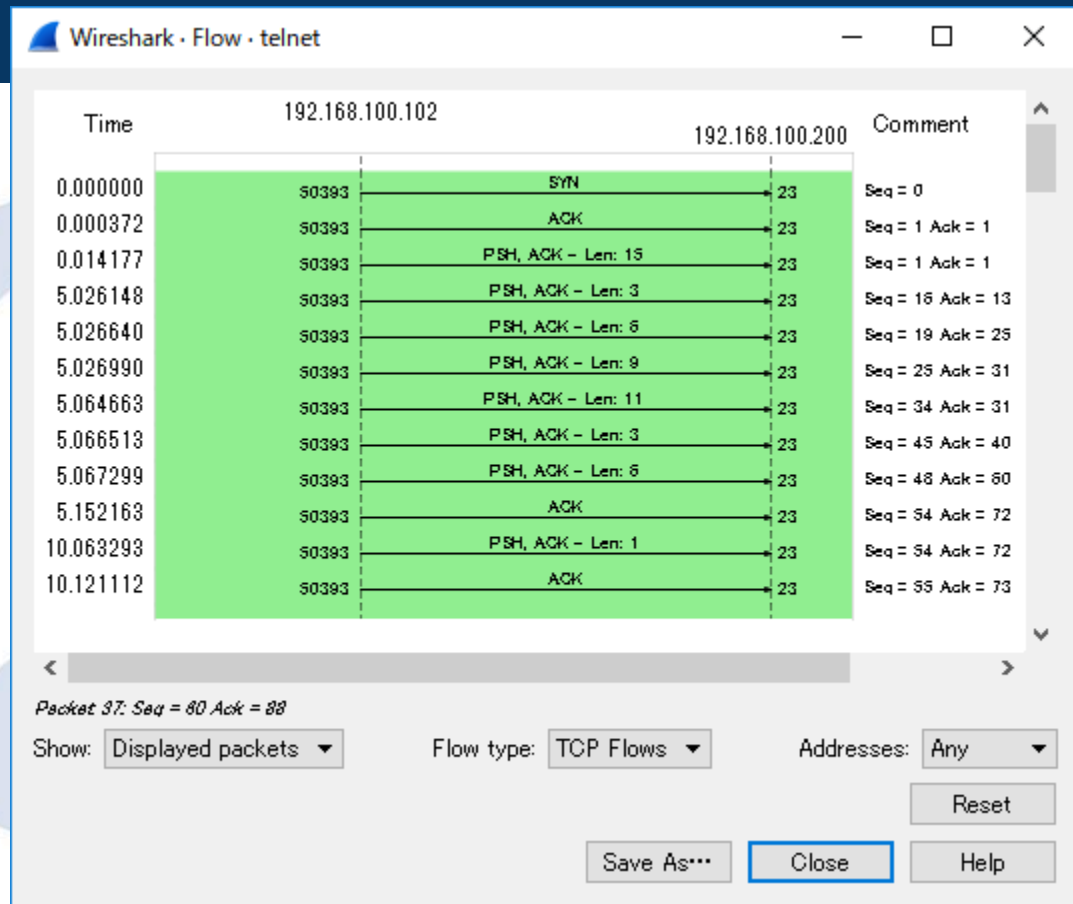
- Enter ip.src== client/server IP address in display filter.
- You can understand the mechanism of TCP
(seq no + segment length = next sequence number)



No.	Time	Source	Destination	Protocol	Length	Sequence number	TCP Segment Len	Next sequence number	Acknowledgment number	Info
1	0.000000	192.168.100.102	192.168.100.200	TCP	66	0	0			0 50393 → 23 [SYN] Seq=0 Win=8192 Len=0 M...
3	0.000372	192.168.100.102	192.168.100.200	TCP	54	1	0			1 50393 → 23 [ACK] Seq=1 Ack=1 Win=65536 ...
4	0.014177	192.168.100.102	192.168.100.200	TELNET	69	1	15	16		1 Telnet Data ...
7	5.026148	192.168.100.102	192.168.100.200	TELNET	57	16	3	19		13 Telnet Data ...
9	5.026640	192.168.100.102	192.168.100.200	TELNET	60	19	6	25		25 Telnet Data ...
11	5.026990	192.168.100.102	192.168.100.200	TELNET	63	25	9	34		31 Telnet Data ...
13	5.064663	192.168.100.102	192.168.100.200	TELNET	65	34	11	45		31 Telnet Data ...
16	5.066513	192.168.100.102	192.168.100.200	TELNET	57	45	3	48		40 Telnet Data ...
18	5.067299	192.168.100.102	192.168.100.200	TELNET	60	48	6	54		60 Telnet Data ...

#3 Stream ID and follow TCP in 2 way

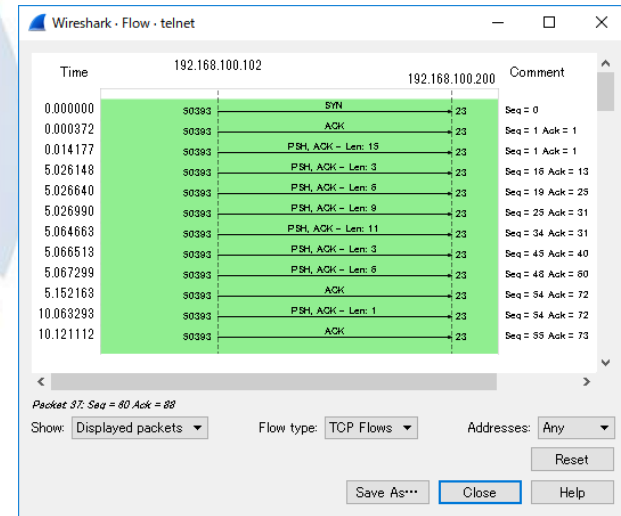
- Statistics > Flow Graph, choose Displayed packets in Show, choose TCP flow in flow type
- It is another way to understand the flow of TCP from a peer.



#3 Stream ID and follow TCP in 2 way Practice

- Open telnet.pcapng
- Set as columns of Sequence no (tcp.seq) TCP Segment Len (tcp.len) Next sequence no (tcp.nxtseq) Acknowledgement no (tcp.ack)
- Set display filter as ip.src from Client and Server
- Choose Statistics > Flow Graph, choose Displayed packets in Show, choose TCP flow in flow type

No.	Time	Source	Destination	Protocol	Length	Sequence number	TCP Segment Len	Next sequence number	Acknowledgment number	Info
1	0.000000	192.168.100.102	192.168.100.200	TCP	66	0	0	0	0	50393 → 23 [SYN] Seq=0 Win=8192 Len=0 M...
3	0.000372	192.168.100.102	192.168.100.200	TCP	54	1	1	15	16	1 50393 → 23 [ACK] Seq=1 Ack=1 Win=65536 ...
4	0.014177	192.168.100.102	192.168.100.200	TELNET	69	1	15	3	19	1 Telnet Data ...
7	5.026148	192.168.100.102	192.168.100.200	TELNET	57	16	3	6	25	13 Telnet Data ...
9	5.026640	192.168.100.102	192.168.100.200	TELNET	60	19	6	9	25	25 Telnet Data ...
11	5.026990	192.168.100.102	192.168.100.200	TELNET	63	25	3	11	45	31 Telnet Data ...
13	5.064663	192.168.100.102	192.168.100.200	TELNET	65	34	3	45	48	31 Telnet Data ...
16	5.066513	192.168.100.102	192.168.100.200	TELNET	57	45	3	54	60	40 Telnet Data ...
18	5.067299	192.168.100.102	192.168.100.200	TELNET	60	48	6	54	60	60 Telnet Data ...



#4 Visualize using I/O Graph

- Wireshark GUI has 3 types of Graph feature.

Rich interface and customizable interface

I/O Graph (set Y axis as bits, packets, and Y fields)

TCP specified analysis (OS level)

TCP Graph (time/seq, RTT, throughput, window scaling)

Simple arrow diagram

Flow Graph (TCP flow, general flow)

I/O Graph			TCP Graph				Flow Graph	
Rich and customizable interface			TCP specified analysis (OS level)				Simple arrow diagram	
bits	packets	Y fields	Time/seq	RTT	Throughput	Window scaling	TCP flow	General flow

#4 Visualize using I/O Graph

I/O Graph (set Y axis as bits, packets, and Y fields)

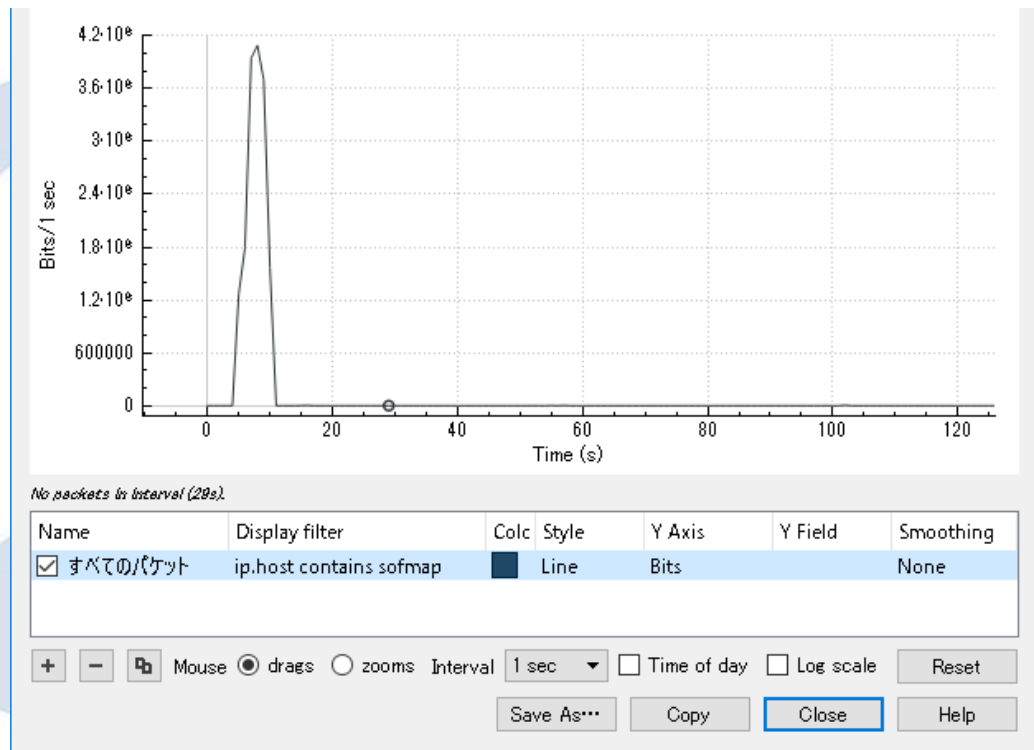
Bits (line)	packets (histogram)	Y fields
bits/s and bytes/s	Frequency	Time, Size, ...

- To see performance and throughput, set Y axis to bit/sec (line)
- To see errors and counting the number of packet, set Y axis to packet/sec (histogram)
- To see time or size or other value, set Y axis to Y fields

#4 Visualize using I/O Graph

Practice4-1 To see performance and throughput

1. Add Item
2. Set Display filter as ip.host contains FQDN (sofmap.com)
3. Set Style as Line
4. set Y axis to bit/sec
5. Check Interval 1 sec

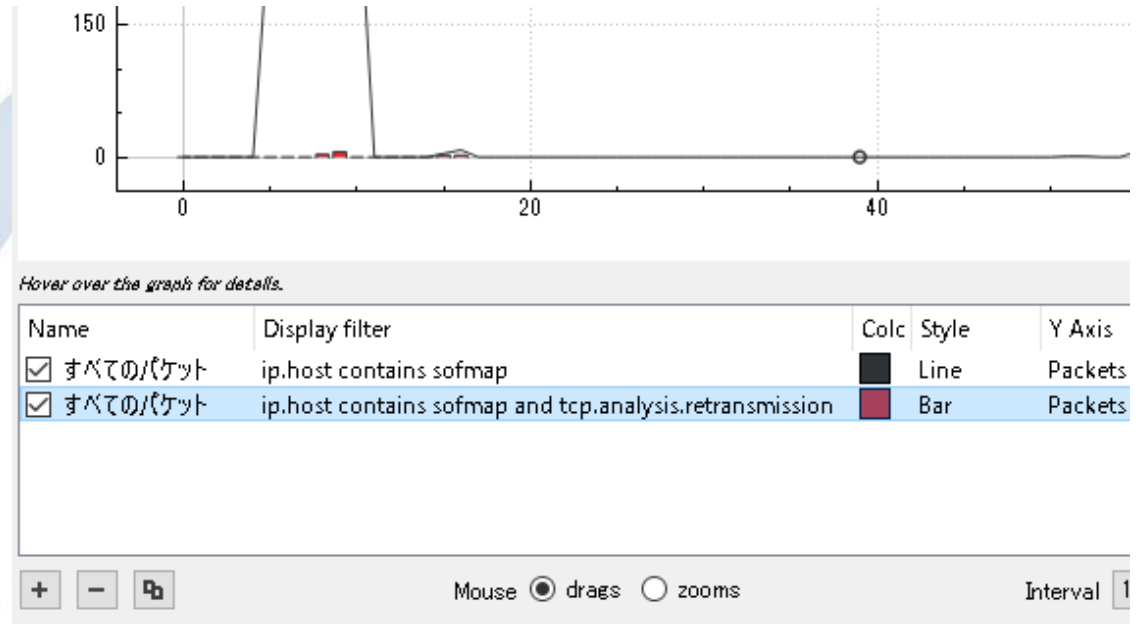


#4 Visualize using I/O Graph

Practice4-2 To see errors and counting

webbrowse.pcapng

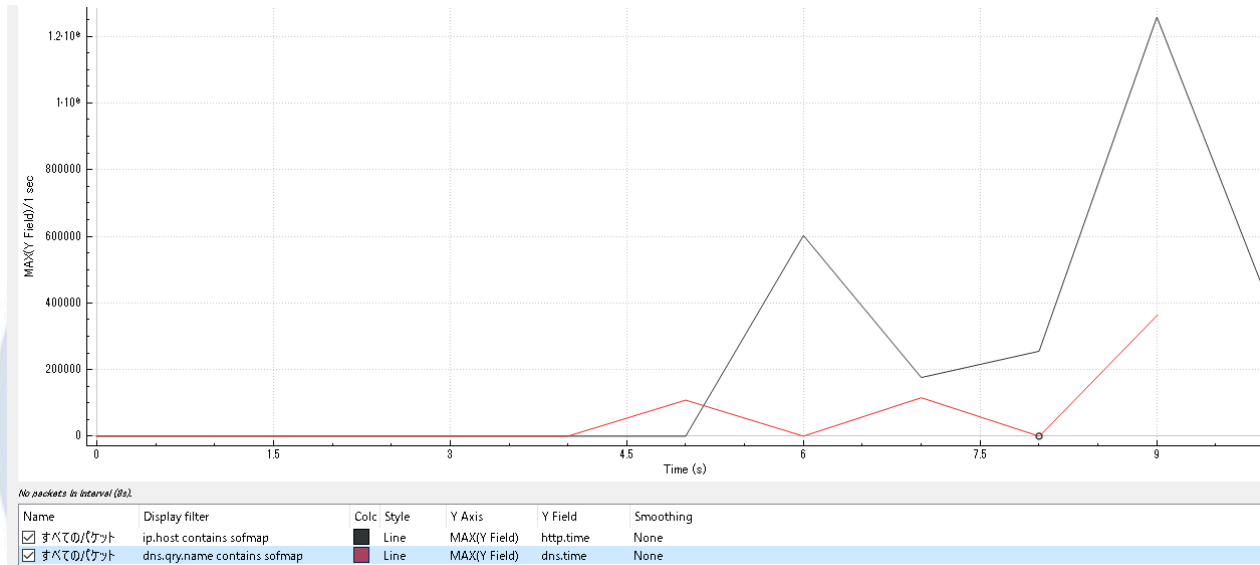
1. Copy current item
2. Add the item's Display filter as "and tcp.analysis.retransmission"
3. Set Style as Bar
4. set Y axis to Packets/sec



#4 Visualize using I/O Graph

Practice4-3 To see time or size or other value,
webbrowse.pcapng

1. Delete 2nd item
2. Set Y Axis as “MAX(Y Field)”
3. Set Y field as “http.time”
4. Copy item
5. Set Display filter as “dns.qry.name contains FQDN”
6. Set Y field as “dns.time”



compare

#5 Capture packet periodically and merge for stats.

- dumpcap/tshark/wireshark has the same option to capture packets for long term,
- At first check the interface index with `-D` option

`dumpcap / tshark / wireshark -b (capture multiple files)`

`duration:NUM`

`filesize:NUM`

`files:NUM`

switch to next file after
NUM secs

switch to next file
after NUM KB

ringbuffer: replace
after NUM files

- The pcap files are created as

`name_seq(5digit)_YYYYMMDDHHMMSS.pcapng`

#5 Capture packet periodically and merge for stats.

- Using duration option is good for statistics.
- You can set duration as you want to set Units for stats.
60(1 minutes) / 3600(1 hour) / 86400(1 day)
- `-a (autostop)` option is also available for statistics

`dumpcap / tshark / wireshark -a (autostop option)`

`duration:NUM`

`filesize:NUM`

`files:NUM`

Stop after NUM secs

Stop after NUM KB

Stop after NUM files

- After capturing, you just check the number of files, then merge using wildcard option of `mergcap`.

#5 Capture packet periodically and merge for stats.

```
C:\Users\megumi\Desktop>dumpcap -D
1. ¥Device¥NPF_{2D82BEBC-1E21-4E44-AF3E-3F9C4EA6748C} (網ノ纒+纒+ 纒イ纒工纒「謗・甘
2. ¥Device¥NPF_{4F0D27B0-E075-4FF7-9D28-BA58ABD38307} (網ノ纒+纒+ 纒イ纒工纒「謗・甘
3. ¥Device¥NPF_{F4FFD736-AD7B-455A-9DD7-EAF47FB51F31} (ethernet)
4. ¥Device¥NPF_{B98C1496-712F-4755-AEB9-87981430B800} (local area connection)
5. ¥Device¥NPF_{411DFF04-625C-4BA3-A3DB-25EFF691C708} (Wi-Fi)

C:\Users\megumi\Desktop>dumpcap -i 3 -w stat.pcapng -b duration:60 -a files:60
Capturing on 'ethernet'
File: stat_00001_20170530143410.pcapng
```

- dumpcap -b duration:60 and -a files:60 option means we capture 60 files (1 hour) that each duration is 60 seconds (1 minute)
- You can merge packets for statistics, just counting files.
- For example, you can merge files just using wildcard

```
C:\Users\megumi\Desktop>mergcap * -w 1hourstat.pcapng
```

Includes Stat_00001_... ~
Stat_00061_...(60 files)

#5 Capture packet periodically and merge for stats.

Practice

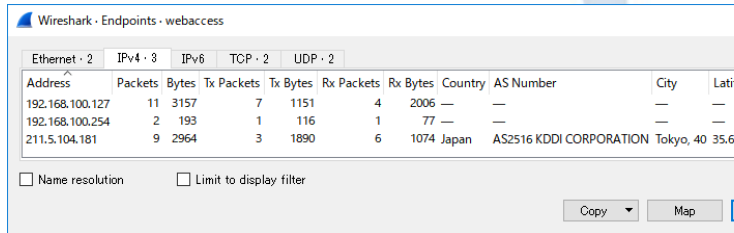
- Check interface index using `-D` option
- Set 6 as duration with `-b`, set 10 as files with `-a` option
- Capture packet to 10 test.pcapng files each duration is 6 second

```
C:\Users\megumi\Desktop>dumpcap -i 1 -b duration:6 -a files:10 -w test.pcapng
Capturing on '網1網2網3網4網5網6網7網8網9網10「誇・邯・11」'
File: test_00001_20170531165817.pcapng
```

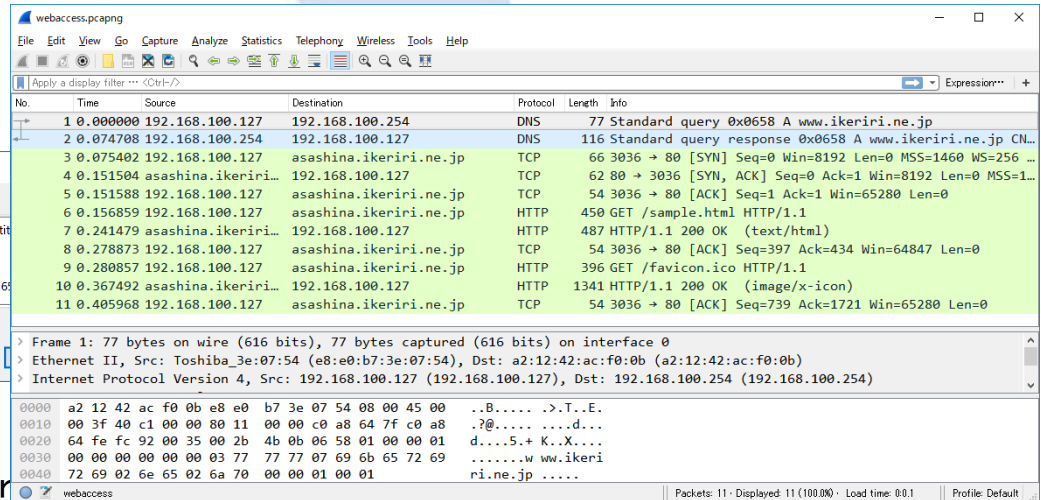
- Merge 10 files (60 seconds) using wildcard
- Create stats using tshark `-qz` option.
conv,ip conv,wlan dns,tree endpoints,ip endpoints,tcp
endpoints,wlan expert follow,http hosts http2,tree http_req,tree
http_srv,tree io,phs io,stat

#6 Sort and stack the pcap file automatically and divide by each filter value

- There are tons of bored and time-wasting pcapng task when we inspect and dissect packets using Wireshark
- Here is a very simple web access pcap file that contains 3 hosts.
- You need to create pcap files filtered with each host.
- How can you do this ?



Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	AS Number	City	Latitude
192.168.100.127	11	3157	7	1151	4	2006	—	—	—	—
192.168.100.254	2	193	1	116	1	77	—	—	—	—
211.5.104.181	9	2964	3	1890	6	1074	Japan	AS2516 KDDI CORPORATION	Tokyo, 40	35.68



webaccess.pcapng

Apply a display filter ... (Ctrl-/)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.127	192.168.100.254	DNS	77	Standard query 0x0658 A www.ikeriri.ne.jp
2	0.074788	192.168.100.254	192.168.100.127	DNS	116	Standard query response 0x0658 A www.ikeriri.ne.jp CN...
3	0.075402	192.168.100.127	asashina.ikeriri.ne.jp	TCP	66	3036 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 ...
4	0.151504	asashina.ikeriri...	192.168.100.127	TCP	62	80 → 3036 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1...
5	0.151588	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=1 Ack=1 Win=65280 Len=0
6	0.156859	192.168.100.127	asashina.ikeriri.ne.jp	HTTP	450	GET /sample.html HTTP/1.1
7	0.241479	asashina.ikeriri...	192.168.100.127	HTTP	487	HTTP/1.1 200 OK (text/html)
8	0.278873	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=397 Ack=434 Win=64847 Len=0
9	0.280857	192.168.100.127	asashina.ikeriri.ne.jp	HTTP	396	GET /favicon.ico HTTP/1.1
10	0.367492	asashina.ikeriri...	192.168.100.127	HTTP	1341	HTTP/1.1 200 OK (image/x-icon)
11	0.405968	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=739 Ack=1721 Win=65280 Len=0

> Frame 1: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0

> Ethernet II, Src: Toshiba_3e:07:54 (e8:e0:b7:3e:07:54), Dst: a2:12:42:ac:f0:0b (a2:12:42:ac:f0:0b)

> Internet Protocol Version 4, Src: 192.168.100.127 (192.168.100.127), Dst: 192.168.100.254 (192.168.100.254)

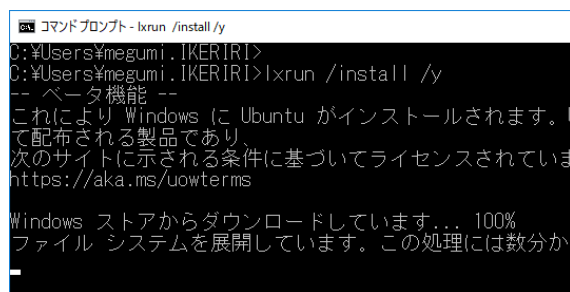
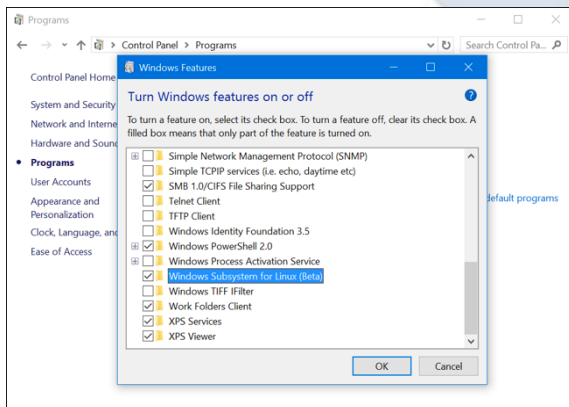
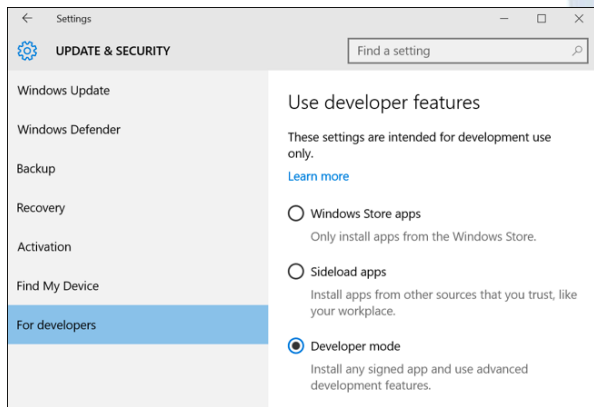
```
0000 a2 12 42 ac f0 0b e8 e0 b7 3e 07 54 08 00 45 00 ..B.....>.T..E.
0010 00 3f 40 c1 00 00 80 11 00 00 c0 a8 64 7f c0 a8 .?@.....d...
0020 64 fe fc 92 00 35 00 2b 4b 0b 06 58 01 00 00 01 d.....S.+K.X...
0030 00 00 00 00 00 03 77 77 77 07 69 6b 65 72 69 .....w ww.ikeri
0040 72 69 02 6e 65 02 6a 70 00 00 01 00 01 ri.ne.jp .....
```


#6 Sort and stack the pcap file automatically

- Windows10 64bit version supports bash shell
 - You can add bash and use another convenient shell command e.x. LOW (Linux on windows) or CYGWIN support too.
 - You can extend wireshark CUI command with these shell script and process packet task automatically.
- (1) Collect source host list from the pcapng files
Hint: tshark -T fileds and -e option,
 - (2) Filter packets by source host
and create each filtered pcapng file.
Hint: tshark -Y ip.src and shell script (for)

#6 Sort and stack the pcap file automatically

- Open Settings > Update & Security > For Developers and activate the Developer Mode
- Open Control Panel > Adding and Deleting Programs > Turn Windows Features On or Off and check Windows Subsystem for Linux (Beta)
- Open cmd.exe and type `bash`, or `lxrunc /install /y`



#6 Sort and stack the pcap file automatically

(1) Collect source host list from the pcapng files

- Wireshark display filter string `ip.src_host` means source IP host
- You can pick up source host list using
“Tshark `-T fields -e Display Filter String`”

```
C:\Users\megumi\Desktop>tshark -r web  
access.pcapng T fields -e ip.src_host
```

```
192.168.100.127  
192.168.100.127 GW  
asashina.ikeriri.ne.jp  
192.168.100.127  
192.168.100.127  
asashina.ikeriri.ne.jp  
192.168.100.127  
192.168.100.127  
asashina.ikeriri.ne.jp  
192.168.100.127
```

tshark returns output
stream with duplication

#6 Sort and stack the pcap file automatically

(1) Collect source host list from the pcapng files

- You can sort output stream using sort command at first, then delete duplication using uniq command

```
C:\Users\megumi\Desktop>bash
user@LetsNoteJ10:/mnt/c/Users/megumi/Desktop$ /mnt/c/"Program Files"/Wires
hark/tshark.exe -r webaccess.pcapng -
T fields -e ip.src_host | sort
192.168.100.127
192.168.100.127
192.168.100.127
192.168.100.127
192.168.100.127
192.168.100.127
192.168.100.127
192.168.100.127
GW
asashina.ikeriri.ne.jp
asashina.ikeriri.ne.jp
asashina.ikeriri.ne.jp
user@LetsNoteJ10:/mnt/c/Users/megumi/
Desktop$
```

Sort with
duplication
at first

```
user@LetsNoteJ10:/mnt/c/Users/megumi/
Desktop$ /mnt/c/"Program Files"/Wires
hark/tshark.exe -r webaccess.pcapng -
T fields -e ip.src_host | sort | uniq
192.168.100.127
GW
asashina.ikeriri.ne.jp
```

Delete duplication with uniq command

#6 Sort and stack the pcap file automatically

(1) Collect source host list from the pcapng files

- Piping “| sort | uniq” give a list without duplication
- You can create temporal working file (hostlist.txt) using redirect at first, then delete duplication using uniq command

```
root@CHEBURASHKA: /mnt/c/Users/megumi.IKERIRI/Desktop
root@CHEBURASHKA:/mnt/c/Users/megumi.IKERIRI/Desktop# /mnt/c/Program Files/Wire
shark/tshark.exe -r webaccess.pcapng -T fields -e ip.src_host | sort | uniq > ho
stlist.txt
root@CHEBURASHKA:/mnt/c/Users/megumi.IKERIRI/Desktop# cat hostlist.txt
192.168.100.127
192.168.100.254
asashina.ikeriri.ne.jp
```

Working file that contains ip.src host list

#6 Sort and stack the pcap file automatically

(2) Filter packets by source host and create file

- Filter packets by source host and create each filtered pcapng file.

```
C:\Users\megumi\Desktop>tshark -r webaccess.pcapng -Y ip.src_host==192.168.100.127  
-w filtered_192.168.100.127.pcapng
```

- The command means open webaccess.pcapng and apply display filter “ip.src_host==192.168.100.127” then create a file (192.168.100.127.pcapng)the display filter was applied.
- Let’s create batch jobs for this task
- We can use “for” and use hostlist.txt
for /f “delims=” %a in (hostlist.txt) do (TASK)

delimiter (¥n return code)

Parameter that contains a line
that was read from hostlist.txt

#6 Sort and stack the pcap file automatically

(2) Filter packets by source host and create file

- Exit bash shell and change directory to Desktop
test “for /f “delims=” %a in (hostlist.txt) do (TASK)”

```
C:\Users\megumi.IKERIRI>cd Desktop
```

Delimiter (¥n return code)

```
C:\Users\megumi.IKERIRI\Desktop>for /f "delims=" %a in (hostlist.txt) do (echo %a)
```

```
C:\Users\megumi.IKERIRI\Desktop>(echo 192.168.100.127 )  
192.168.100.127
```

Display a line that was read from hostlist.txt

```
C:\Users\megumi.IKERIRI\Desktop>(echo 192.168.100.254 )  
192.168.100.254
```

```
C:\Users\megumi.IKERIRI\Desktop>(echo asashina.ikeriri.ne.jp )  
asashina.ikeriri.ne.jp
```

#6 Sort and stack the pcap file automatically

Practice

- Collect source ip host list from webaccess.pcapng
`tshark.exe -r webaccess.pcapng -T fields -e ip.src_host | sort |
uniq > hostlist.txt`
- Filter packets by source host and create pcap file that contains
the each host
for /f "delims=" %a in (hostlist.txt) do (tshark -r
webaccess.pcapng -Y ip.src_host==%a -w filtered_%a.pcapng)

#6 Sort and stack the pcap file automatically

Practice

```
C:\Users\megumi.IKERIRI\Desktop>bash
^[[Aroot@CHEBURASHKA:/mnt/c/Users/megumi.IKERIRI/Desktop# /mnt/c/Program Files/Wireshark
ark.exe -r webaccess.pcapng -T fields -e ip.src_host | sort | uniq > hostlist.txt
root@CHEBURASHKA:/mnt/c/Users/megumi.IKERIRI/Desktop# exit
exit

C:\Users\megumi.IKERIRI\Desktop>for /f "delims=" %a in ( hostlist.txt) do ( tshark -r web
access.pcapng -Y ip.src_host==%a -w filtered_%a.pcapng )

C:\Users\megumi.IKERIRI\Desktop>(tshark -r webaccess.pcapng -Y ip.src_host==192.168.100.1
27 -w filtered_192.168.100.127.pcapng )

C:\Users\megumi.IKERIRI\Desktop>(tshark -r webaccess.pcapng -Y ip.src_host==192.168.100.2
54 -w filtered_192.168.100.254.pcapng )

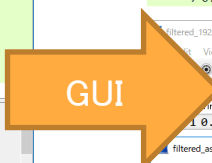
C:\Users\megumi.IKERIRI\Desktop>(tshark -r webaccess.pcapng -Y ip.src_host==asashina.iker
iri.ne.jp -w filtered_asashina.ikeriri.ne.jp.pcapng )

C:\Users\megumi.IKERIRI\Desktop>
```

#6 Sort and stack the pcap file automatically

Compare CLI with GUI

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.127	192.168.100.254	DNS	77	Standard query 0x0658 A www.ikeriri.ne.jp
2	0.074708	192.168.100.254	192.168.100.127	DNS	116	Standard query response 0x0658 A www.ikeriri.ne.jp CN...
3	0.075402	192.168.100.127	asashina.ikeriri.ne.jp	TCP	66	3036 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 ...
4	0.151504	asashina.ikeriri.ne.jp	192.168.100.127	TCP	62	80 → 3036 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1...
5	0.151588	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=1 Ack=1 Win=65280 Len=0
6	0.156859	192.168.100.127	asashina.ikeriri.ne.jp	HTTP	450	GET /sample.html HTTP/1.1
7	0.241479	asashina.ikeriri.ne.jp	192.168.100.127	HTTP	487	HTTP/1.1 200 OK (text/html)
8	0.278873	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=397 Ack=434 Win=64847 Len=0
9	0.280857	192.168.100.127	asashina.ikeriri.ne.jp	HTTP	396	GET /favicon.ico HTTP/1.1
10	0.367492	asashina.ikeriri.ne.jp	192.168.100.127	HTTP	1341	HTTP/1.1 200 OK (image/x-icon)
11	0.405968	192.168.100.127	asashina.ikeriri.ne.jp	TCP	54	3036 → 80 [ACK] Seq=739 Ack=1721 Win=65280 Len=0



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.127	192.168.100.254	DNS	77	Standard query 0x0658 A www.ikeriri.ne.jp
2	0.075402	192.168.100.127	211.5.104.181	TCP	66	3036 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 ...
3	0.151588	192.168.100.127	211.5.104.181	TCP	54	3036 → 80 [ACK] Seq=1 Ack=1 Win=16711680 Len=0
4	0.156859	192.168.100.127	211.5.104.181	HTTP	450	GET /sample.html HTTP/1.1
5	0.278873	192.168.100.127	211.5.104.181	TCP	54	3036 → 80 [ACK] Seq=397 Ack=434 Win=16600832 Len=0
6	0.280857	192.168.100.127	211.5.104.181	HTTP	396	GET /favicon.ico HTTP/1.1
7	0.405968	192.168.100.127	211.5.104.181	TCP	54	3036 → 80 [ACK] Seq=739 Ack=1721 Win=16711680 Len=0



webaccess.pcapng



filtered_192.168.100.127.pcapng



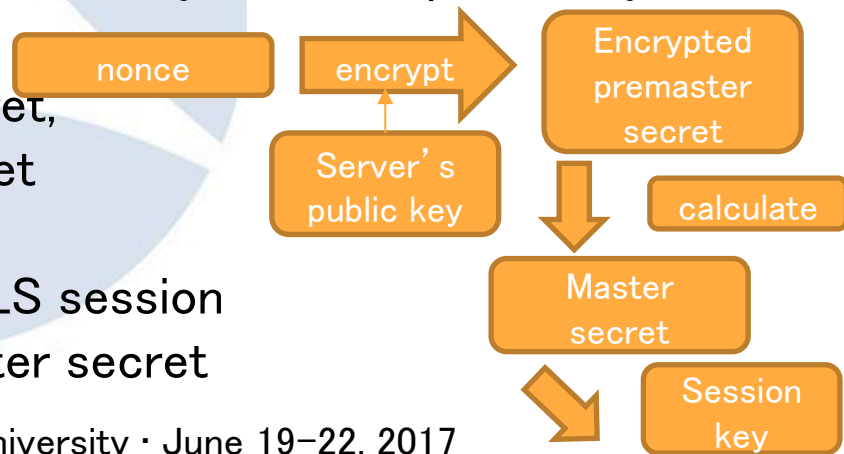
filtered_192.168.100.254.pcapng



filtered_asashina.ikeriri.ne.jp.pcapng

#7 Decrypting TLS/SSL data without key pair

- There are many encrypted communication using TLS/SSL in today's Internet.
- For example, HTTP2 is based on TLS/SSL
- If you have a server's private key with certification (PEM format), you can decrypt TLS/SSL trace file, but it is mere.
- So it is essential to decrypt TLS/SSL trace by session
- Client creates nonce (random) and encrypt nonce by server's public key, it is encrypted premaster secret
- Master secret is created by premaster secret, then session key is created by master secret
- If you have unencrypted premaster secret, you can retrieve session key of the SSL/TLS session
- You need to leak your unencrypted premaster secret from Chrome.



#7 Decrypting TLS/SSL data without key pair

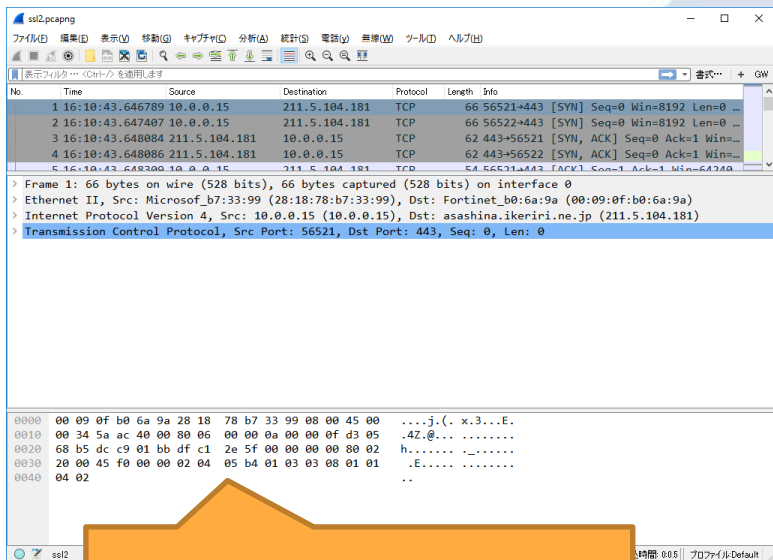
1. Set system environment variable SSLKEYLOGFILE=Path of the premaster secret

The image shows a Windows System Properties dialog box with the 'Environment' tab selected. The 'System variables' section is visible, and the 'Environment variables...' button is highlighted with an orange arrow. An orange callout box above the dialog contains the text: `SSLKEYLOGFILE = C:¥Users¥megumi¥Desktop¥ssl.key`. Below the dialog, a table lists system environment variables, with `SSLKEYLOGFILE` highlighted in orange. The table is as follows:

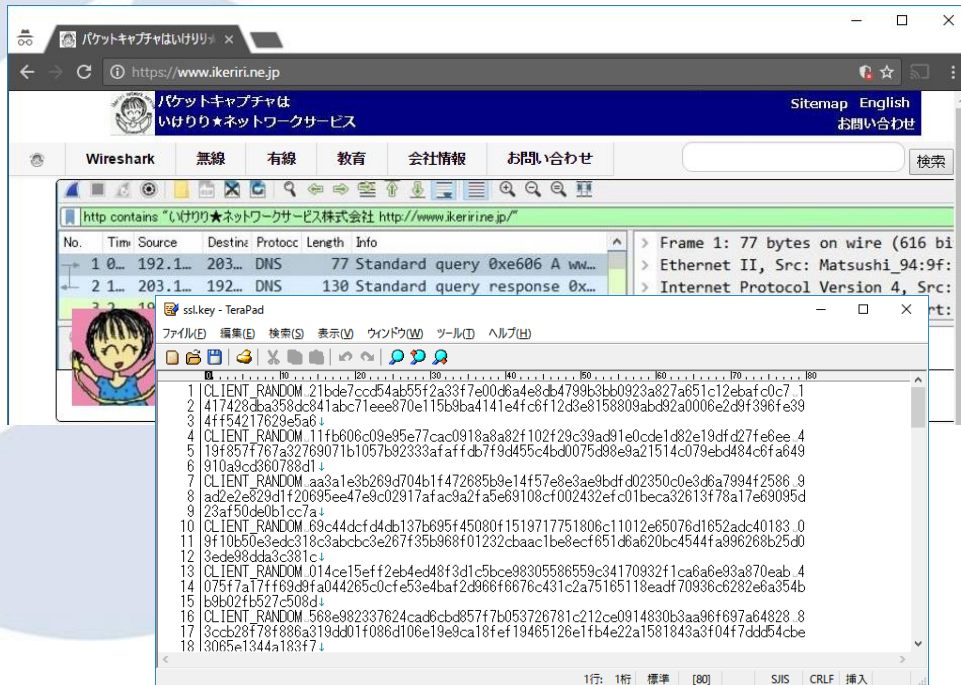
変数	値
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 44 Stepping 2, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	2c02
PSModulePath	C:¥WINDOWS¥system32¥WindowsPowerShell¥v1.0¥Modules¥C:¥Pr...
SSLKEYLOGFILE	C:¥Users¥megumi.IKERIRI¥Desktop¥ssl.key
TEMP	C:¥WINDOWS¥TEMP

#7 Decrypting TLS/SSL data without key pair

2. Capture packets using Chrome (<https://www.ikeriri.ne.jp/>) ssl2.pcapng
1. Check SSLKEYLOGFILE was generated



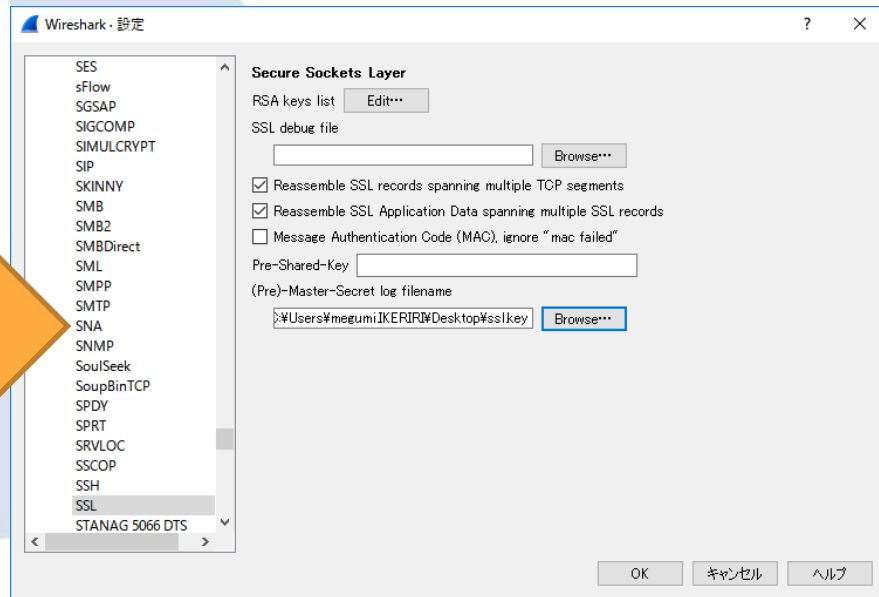
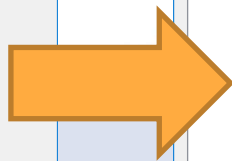
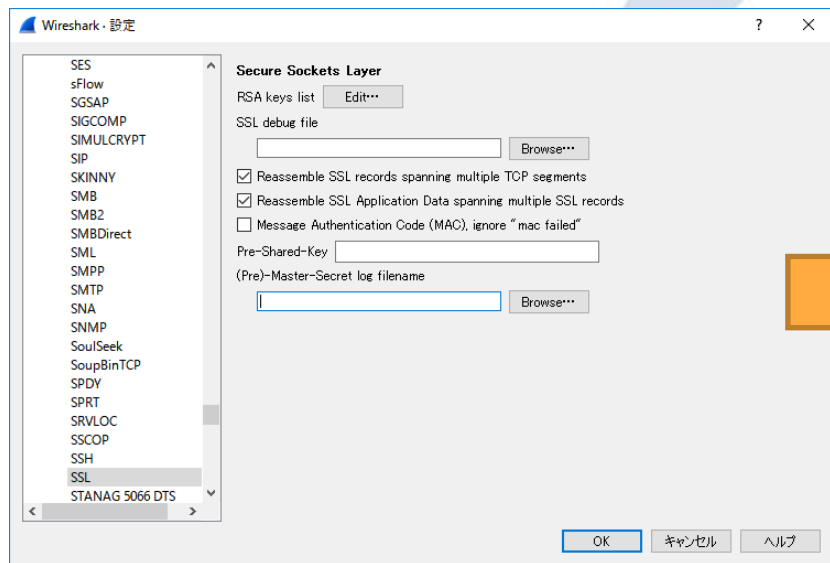
ssl2.pcapng



#7 Decrypting TLS/SSL data without key pair

3. Set (Pre)-Master-Secret log filename in SSL preference

Add path of ssl.key in (Pre)-Master-Secret log filename



#7 Decrypting TLS/SSL data without key pair

Check #23,28 in trace file ssl2.pcapng

ssl2.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
22	16:10:43.719725	10.0.0.15	211.5.104.181	TCP	54	56522->443 [ACK] Seq=525 Ack=3851 Win=...
23	16:10:44.095939	10.0.0.15	211.5.104.181	TLSv1	512	Application Data, Application Data
24	16:10:44.097781	211.5.104.181	10.0.0.15	TCP	1514	[TCP segment of a reassembled PDU]
25	16:10:44.097784	211.5.104.181	10.0.0.15	TCP	1514	[TCP segment of a reassembled PDU]

> Frame 23: 512 bytes on wire (4096 bits), 512 bytes captured (4096 bits) on interface 0
> Ethernet II, Src: Microsoft_b7:33:99 (28:18:78:b7:33:99), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a)
> Internet Protocol Version 4, Src: 10.0.0.15 (10.0.0.15), Dst: asashina.ikeriri.ne.jp (211.5.104.181)
> Transmission Control Protocol, Src Port: 56521, Dst Port: 443, Seq: 525, Ack: 3851, Len: 458
> Secure Sockets Layer

```
0030 f7 4e 47 ae 00 00 17 03 01 00 20 bb 34 c5 12 c8 .NG... ..4...
0040 da 2c dc a2 fc 66 8e f9 9f a4 45 75 5c 93 d3 36 ..F..Eu..6
0050 5e e9 05 51 ee d4 72 7e 87 3b 93 17 03 01 a0 V..Q... ..
0060 83 b0 a5 db f5 c7 07 3a 46 0f de 30 71 60 ee fe ..... F..0a"
0070 d8 97 c5 c5 34 23 0b c5 75 5d df 40 a6 78 ab 4 ..4#.. u]@.x.M
0080 f7 d9 61 97 6d fe 58 5f 6e 77 b6 0f ee 56 3c ac ..a..x..nw...V..
0090 2b da c1 59 aa 59 78 fd 7a 4e fd 22 25 68 0d 2a +.Y.Yx.zN."%h.#
00a0 46 60 bd 49 94 09 85 51 97 d6 51 8e 32 04 c6 b6 F..l...Q..Q...
00b0 4f a3 71 1c 50 e6 22 be 3b 17 80 b9 74 d8 49 0d 0.q.P." ..:..t..
```



ssl2.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
22	16:10:43.719725	10.0.0.15	211.5.104.181	TCP	54	56522->443 [ACK] Seq=525 Ack=3851 Win=...
23	16:10:44.095939	10.0.0.15	211.5.104.181	HTTP	512	GET / HTTP/1.1
24	16:10:44.097781	211.5.104.181	10.0.0.15	TCP	1514	[TCP segment of a reassembled PDU]
25	16:10:44.097784	211.5.104.181	10.0.0.15	TCP	1514	[TCP segment of a reassembled PDU]

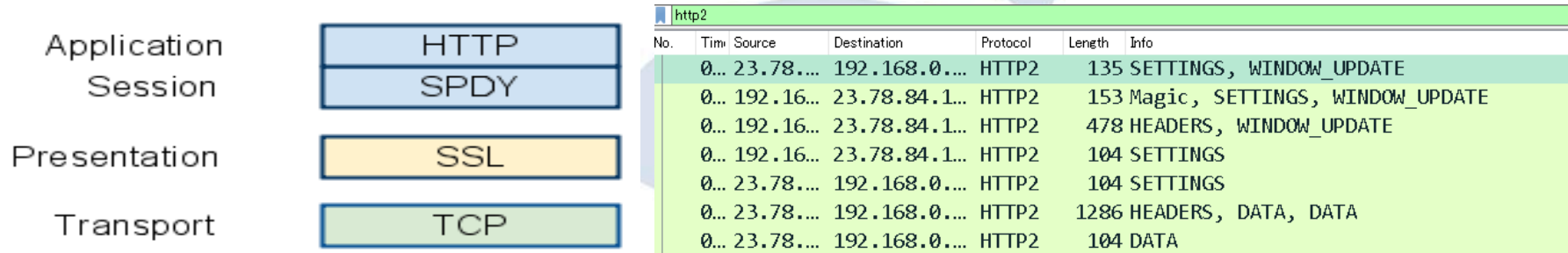
> Frame 23: 512 bytes on wire (4096 bits), 512 bytes captured (4096 bits) on interface 0
> Ethernet II, Src: Microsoft_b7:33:99 (28:18:78:b7:33:99), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a)
> Internet Protocol Version 4, Src: 10.0.0.15 (10.0.0.15), Dst: asashina.ikeriri.ne.jp (211.5.104.181)
> Transmission Control Protocol, Src Port: 56521, Dst Port: 443, Seq: 525, Ack: 3851, Len: 458
> Secure Sockets Layer
> [2 Reassembled SSL segments (389 bytes): #23(1), #23(388)]
▼ Hypertext Transfer Protocol
> GET / HTTP/1.1\r\nHost: www.ikeriri.ne.jp\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.14
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\nAccept-Encoding: gzip, deflate, sdch, br\r\n

```
0000 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a GET / HT TP/1.1.
0010 48 6f 73 74 3a 20 77 77 77 2e 69 6b 65 72 69 72 Host: ww w.ikeriri
0020 69 2e 6e 65 2e 6a 70 0d 0a 43 6f 6e 6e 65 63 74 i.ne.jp. .Connect
0030 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d ion: kee p-alive.
0040 0a 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75 72 _Upgrade-Insecur
0050 65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a 55 e-Request s: 1..L
0060 73 65 72 6d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 2a r-Agent t: Mozil
0070 6c 61 2f 35 2e 30 28 28 57 69 6e 64 6f 77 73 20 la/5.0 ( Windows
```

#8 HTTP2 dissection

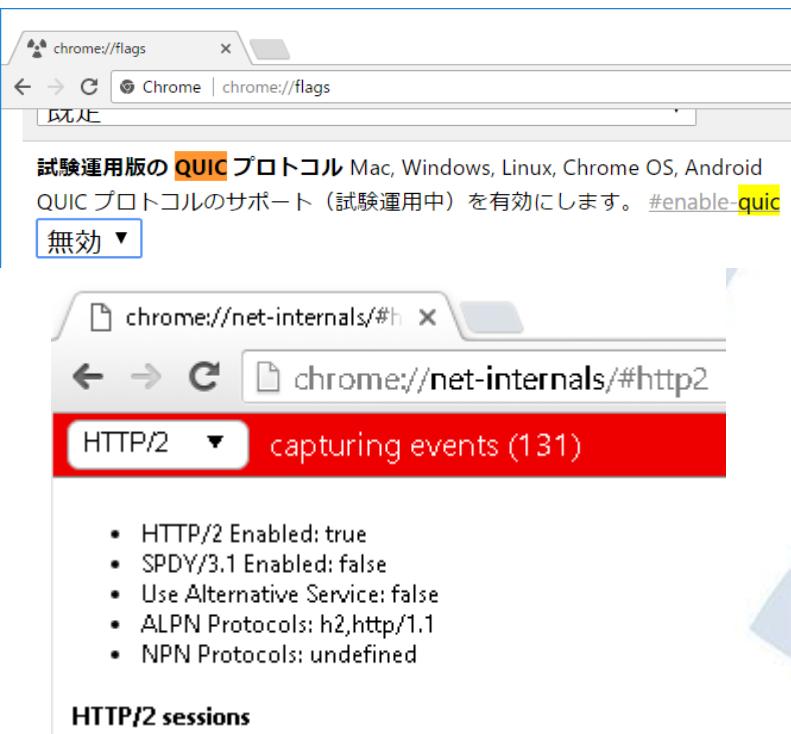
HTTP/2.0 has already here

- In Windows 10 age, major Web browser and websites such as Google services including Google Map and Gmail, Facebook, Twitter, Yahoo and web services are ready.
- HTTP/2.0 uses with TLS and all traffic is encrypted.
- Please run the Chrome, Microsoft Edge, Safari and capture major webpage, HTTP2 has already been here.



#8 HTTP2 dissection

HTTP/2.0 uses SSL/TLS



- HTTP/2.0 works on SSL/TLS connection
- Open Chrome URL “chrome://flags/”
- Set Disable to QUIC protocol in list box
- Now Chrome prefer to use HTTP2
- Start capture and open www.twitter.com in Chrome and save twitter.pcapng
- Open the Chrome and type chrome://net-internals/#http2 you can see the HTTP/2 sessions
- Wireshark has the dissector of HTTP2.0 <https://www.wireshark.org/docs/dfref/h/http2>

#8 HTTP2 dissection

Decryption setting of SSL/TLS

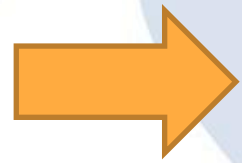
Open twitter.pcapng and set (Pre)-Master-Secret log filename
Twitter_unencrypted_premaster_secret in SSL preference

twitter.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.13	s.twitter.com	TCP	66	27964 → 443 [SYN] Seq=0 Win=8192
2	0.013941	s.twitter.com	10.0.0.13	TCP	66	443 → 27964 [SYN, ACK] Seq=0 Ack=...
3	0.014027	10.0.0.13	s.twitter.com	TCP	54	27964 → 443 [ACK] Seq=1 Ack=1 Win=...
4	0.014176	10.0.0.13	s.twitter.com	TLSv1.2	267	Client Hello
5	0.025522	s.twitter.com	10.0.0.13	TCP	60	443 → 27964 [ACK] Seq=1 Ack=214 Wi...
6	0.028770	s.twitter.com	10.0.0.13	TLSv1.2	1468	Server Hello
7	0.033601	s.twitter.com	10.0.0.13	TLSv1.2	1468	Certificate
8	0.033603	s.twitter.com	10.0.0.13	TLSv1.2	260	Server Key
9	0.033673	10.0.0.13	s.twitter.com	TCP	54	27964 → 443
10	0.035667	10.0.0.13	s.twitter.com	TLSv1.2	180	Client Key
11	0.039356	10.0.0.13	s.twitter.com	TLSv1.2	107	Application
12	0.039396	10.0.0.13	s.twitter.com	TLSv1.2	110	Application
13	0.039425	10.0.0.13	s.twitter.com	TLSv1.2	96	Application
14	0.039605	10.0.0.13	s.twitter.com	TLSv1.2	793	Application
15	0.042755	s.twitter.com	10.0.0.13	TLSv1.2	296	New Session
16	0.044729	s.twitter.com	10.0.0.13	TLSv1.2	98	Application
17	0.044758	10.0.0.13	s.twitter.com	TCP	54	27964 → 443 [ACK] Seq=1230 Ack=3...

Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_08:00:00:00:00:00 (08:00:00:00:00:00), Protocol: Internet Protocol Version 4, Src: 10.0.0.13 (10.0.0.13), Dst: s.twitter.com (104.244.42.67)

```
0000  00 09 0f b0 6a 9a 20 cf 30 55 f4 56 08 00 45 00  ....j...
0010  00 34 38 dc 40 00 80 06 00 0a 00 00 0d 68 f4    .48@...
0020  2a 43 6d 3c 01 bb bd fe 55 aa 00 00 00 00 02   *Cm....
0030  20 00 9d 6a 00 00 02 04 05 b4 01 03 03 08 01 01  ..j....
0040  04 02
```



Wireshark - Preferences

Secure Sockets Layer

RSA keys list: Edit

SSL debug file: Browse

Reassemble SSL records spanning multiple TCP segments

Reassemble SSL Application Data spanning multiple SSL records

Message Authentication Code (MAC) ignore "mac failed"

Pre-Shared-Key:

(Pre)-Master-Secret log filename: Twitter_unencrypted_premaster_secret.txt Browse

OK Cancel Help

twitter.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.13	104.244.42.67	TCP	66	27964 → 443 [SYN] Seq=0 Win=8192...
2	0.013941	104.244.42.67	10.0.0.13	TCP	66	443 → 27964 [SYN, ACK] Seq=0 Ack=...
3	0.014027	10.0.0.13	104.244.42.67	TCP	54	27964 → 443 [ACK] Seq=1 Ack=1 Win=...
4	0.014176	10.0.0.13	104.244.42.67	TLSv1.2	267	Client Hello
5	0.025522	104.244.42.67	10.0.0.13	TCP	60	443 → 27964 [ACK] Seq=1 Ack=214 Wi...
6	0.028770	104.244.42.67	10.0.0.13	TLSv1.2	1468	Server Hello
7	0.033601	104.244.42.67	10.0.0.13	TLSv1.2	1468	Certificate[TCP segment of a rea...
8	0.033603	104.244.42.67	10.0.0.13	TLSv1.2	260	Server Key Exchange, Server Hell...
9	0.033673	10.0.0.13	104.244.42.67	TCP	54	27964 → 443 [ACK] Seq=1230 Ack=30...
10	0.035667	10.0.0.13	104.244.42.67	TLSv1.2	180	Client Key Exchange, Change Ciph...
11	0.039356	10.0.0.13	104.244.42.67	HTTP2	107	Magic
12	0.039396	10.0.0.13	104.244.42.67	HTTP2	110	SETTINGS
13	0.039425	10.0.0.13	104.244.42.67	HTTP2	96	WINDOW_UPDATE
14	0.039605	10.0.0.13	104.244.42.67	HTTP2	793	HEADERS
15	0.042755	104.244.42.67	10.0.0.13	TLSv1.2	296	New Session Ticket, Change Ciph...
16	0.044729	104.244.42.67	10.0.0.13	HTTP2	98	SETTINGS
17	0.044758	10.0.0.13	104.244.42.67	TCP	54	27964 → 443 [ACK] Seq=1230 Ack=3...

Frame 10: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits) on interface 0
> Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_08:00:00:00:00:00 (08:00:00:00:00:00), Protocol: Internet Protocol Version 4, Src: 10.0.0.13 (10.0.0.13), Dst: 104.244.42.67 (104.244.42.67)

```
0000  00 09 0f b0 6a 9a 20 cf 30 55 f4 56 08 00 45 00  ....j...
0010  00 a6 38 e0 00 00 80 06 00 0a 00 00 0d 68 f4    .8@...
0020  2a 43 6d 3c 01 bb bd fe 56 80 f1 44 c9 ea 50 18  *Cm....
0030  01 03 9d dc 00 00 16 03 03 00 46 10 00 00 42 f1  .....F...
0040  04 7d 19 81 7d 7b df bd 0a 34 33 3e 55 3d ef 8a  ..}}{...43U=...
```

#8 HTTP2 dissection

Stream ID is important to dissect HTTP2

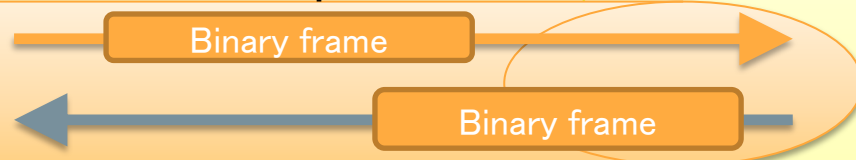
HTTP/2.0 manages communications using Stream mechanism

HTTP/2.0 uses 1 tcp connection and many Stream (virtual connection channel) that has id and priority

1 tcp connection used by HTTP/2.0

HTTP/2.0 request

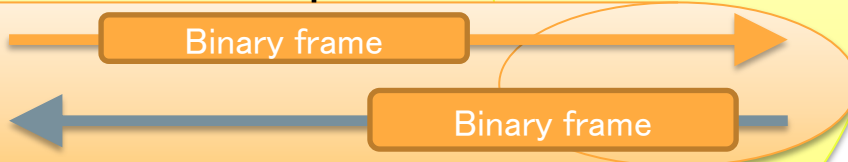
Stream
(id 1)



HTTP/2.0 response

HTTP/2.0 request

Stream
(id 2)



HTTP/2.0 response

Web
browser

server

#8 HTTP2 dissection

Add stream ID row and filter http2

Select packet 1 and open HTTP2 header, and select Stream Identifier (http2.streamid) and right click and select “apply as column”
Set http2 in Display Filter

The screenshot shows the Wireshark interface with packet 1 selected. The display filter is set to 'http2'. The packet list pane shows the following data:

No.	Time	Source	Destination	Protocol	Length	Stream Identifier	Info
10	0.035667	10.0.0.13	104.244.42.67	TLSv1.2	180		Client Key Exchange...
11	0.039356	10.0.0.13	104.244.42.67	HTTP2	107		Magic
12	0.039396	10.0.0.13	104.244.42.67	HTTP2	110		0 SETTINGS
13	0.039425	10.0.0.13	104.244.42.67	HTTP2	96		0 WINDOW_UPDATE

The packet details pane shows the expanded HTTP2 header for packet 11:

- Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a)
- Internet Protocol Version 4, Src: 10.0.0.13 (10.0.0.13), Dst: 104.244.42.67 (104.244.42.67)
- Transmission Control Protocol, Src Port: 27964, Dst Port: 443, Seq: 393, Ack: 3035, Len: 56
- Secure Sockets Layer
- HyperText Transfer Protocol 2
 - Stream: SETTINGS, Stream ID: 0, Length 18
 - Length: 18
 - Type: SETTINGS (4)
 - Flags: 0x00
 - 0... = Reserved: 0x0
 - .000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
 - Settings - Max concurrent streams: 1000

The screenshot shows the Wireshark interface with the display filter set to 'http2'. The packet list pane shows the following data:

No.	Time	Source	Destination	Protocol	Length	Stream Identifier	Info
11	0.039356	10.0.0.13	104.244.42.67	HTTP2	107		Magic
12	0.039396	10.0.0.13	104.244.42.67	HTTP2	110		0 SETTINGS
13	0.039425	10.0.0.13	104.244.42.67	HTTP2	96		0 WINDOW_UPDATE
14	0.039605	10.0.0.13	104.244.42.67	HTTP2	793		1 HEADERS
16	0.044729	104.244.42.67	10.0.0.13	HTTP2	98		0 SETTINGS
18	0.044797	10.0.0.13	104.244.42.67	HTTP2	92		0 SETTINGS
20	0.047417	104.244.42.67	10.0.0.13	HTTP2	92		0 SETTINGS
23	0.154656	104.244.42.67	10.0.0.13	HTTP2	421		1 HEADERS
24	0.154743	104.244.42.67	10.0.0.13	HTTP2	120		1 DATA
36	21.998814	10.0.0.13	104.244.42.67	HTTP2	107		Magic
37	21.998847	10.0.0.13	104.244.42.67	HTTP2	110		0 SETTINGS
38	21.998876	10.0.0.13	104.244.42.67	HTTP2	96		0 WINDOW_UPDATE
39	21.999033	10.0.0.13	104.244.42.67	HTTP2	632		1 HEADERS
43	22.003207	104.244.42.67	10.0.0.13	HTTP2	98		0 SETTINGS
45	22.003281	10.0.0.13	104.244.42.67	HTTP2	92		0 SETTINGS
46	22.004314	104.244.42.67	10.0.0.13	HTTP2	92		0 SETTINGS
49	22.110581	104.244.42.67	10.0.0.13	HTTP2	422		1 HEADERS
50	22.111030	104.244.42.67	10.0.0.13	HTTP2	264		1 DATA
52	22.316347	10.0.0.13	104.244.42.67	HTTP2	226		3 HEADERS
54	22.429617	104.244.42.67	10.0.0.13	HTTP2	181		3 HEADERS
55	22.430004	104.244.42.67	10.0.0.13	HTTP2	143		3 DATA
57	22.665791	10.0.0.13	104.244.42.67	HTTP2	155		5 HEADERS
59	22.776187	104.244.42.67	10.0.0.13	HTTP2	127		5 HEADERS
60	22.776273	104.244.42.67	10.0.0.13	HTTP2	143		5 DATA

#8 HTTP2 dissection

check HTTP2 request header

- Set `http2.header.value == "GET"` (it means `HEADER(http2.type == 1)`)
- Check 39 packet and open HTTP2 header

- HTTP2 are binary frame
- Check major header field
- Method HTTP method
- Authority Host of the server
- Scheme HTTP / HTTPS
- Path Path of the object
- Accept accept document type
- User-agent browser information
- Referrer last visited uri
- Cookie

```
HyperText Transfer Protocol 2
  Stream: HEADERS, Stream ID: 1, Length 540
    Length: 540
    Type: HEADERS (1)
  > Flags: 0x25
    0... .. = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
    1... .. = Exclusive: True
    .000 0000 0000 0000 0000 0000 0000 0000 = Stream Dependency: 0
    Weight: 109
    [Weight real: 110]
    Header Block Fragment: 82418f1d43a3d24c442e9f064a4b62e43d3f870084b958d3...
    [Header Length: 868]
    [Header Count: 13]
  > Header: :method: GET
  > Header: :authority: analytics.twitter.com
  > Header: :scheme: https
  > Header: :path: /tpm/p?_=1490016087878
  > Header: accept: application/json, text/javascript, */*; q=0.01
  > Header: origin: https://twitter.com
  > Header: user-agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KH
  > Header: referer: https://twitter.com/
  > Header: accept-encoding: gzip, deflate, sdch, br
  > Header: accept-language: ja,en-US;q=0.8,en;q=0.6
  > Header: cookie: guest_id=v1%3A149001608692454384
  > Header: cookie: _twitter_sess=BAh7CSIKZmxhc2hJQzonzONQW00aw9uQ29udHJvbjGx1cjo6Rmxhc:
  > Header: cookie: ct0=c629b54e39ddb3c1adce0286bce69b0a
    Padding: <MISSING>
```

#8 HTTP2 dissection

check HTTP2 response header

- `http2.header.value == "200"` (it means `HEADER(http2.type == 1)`)
- Check 49 packet and open HTTP2 header

- HTTP2 are binary frame
- Check major header field
- `:status` Status code
- Content-encoding compression
- Server Web server
- `content-type` document type

```
HyperText Transfer Protocol 2
  Stream: HEADERS, Stream ID: 1, Length 330
    Length: 330
    Type: HEADERS (1)
  > Flags: 0x24
    0... .. = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
    0... .. = Exclusive: False
    .000 0000 0000 0000 0000 0000 0000 0000 = Stream Dependency: 0
    Weight: 95
    [Weight real: 96]
    Header Block Fragment: 88409619085421621ea4d87a161d141fc2c4b0b216e
    [Header Length: 668]
    [Header Count: 18]
  > Header: :status: 200
  > Header: access-control-allow-credentials: true
  > Header: access-control-allow-origin: https://twitter.com
  > Header: access-control-expose-headers:
  > Header: content-disposition: attachment; filename=json.json
  > Header: content-encoding: gzip
  > Header: content-length: 172
  > Header: content-type: application/json;charset=utf-8
  > Header: date: Mon, 20 Mar 2017 13:21:28 GMT
  > Header: server: tsa_m
  > Header: strict-transport-security: max-age=631138519
  > Header: vary: Origin
  > Header: x-connection-hash: da31c3174c19882f2fe09406968e8320
  > Header: x-content-type-options: nosniff
  > Header: x-frame-options: SAMEORIGIN
  > Header: x-response-time: 104
  > Header: x-transaction: 0017a73c002a5c68
  > Header: x-xss-protection: 1; mode=block
  _ _Padding: <MISSING> _ _ _ _
```

#8 HTTP2 dissection more information

For more detail mechanism and information about HTTP2
Please refer my last year's presentation at Sharkfest 2016
“Wireshark 2.0 Tips for HTTP and HTTP2 Analysis”



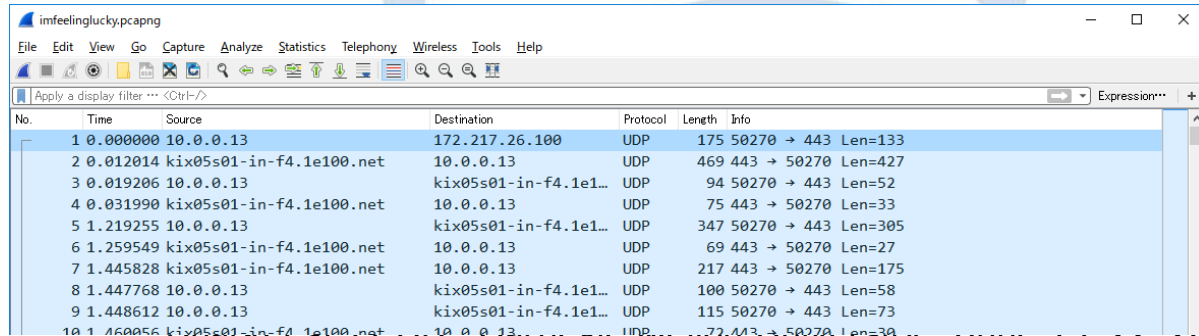
#9 QUIC dissection

What is QUIC

- TCP 3 way handshake needs more connection time.
(off course there is a TCP Fast Open)
- TCP have to send new segment after old segment is confirmed as received successfully (Head of Line Blocking)
- TCP resend timer calculation and receive window mechanism are not adequate in today's internet
- Anyway, TCP is slow protocol, so SSL/TLS is slow, and HTTP is also slow, so Google create new protocol instead of TCP/(SSL/TLS)/(HTTP/HTTP2) model
- QUIC (Quick UDP Internet Connection) comes.

#9 QUIC dissection capturing QUIC

- QUIC (Quick UDP Internet Connection) is Internet-Drafts <https://datatracker.ietf.org/wg/quic/documents/>
- open Wireshark and start capturing
open www.google.com and I'm feeling lucky, stop capture.
Then save filtered pcapng as imfeelinglucky.pcapng
- At this time we just see some UDP stream

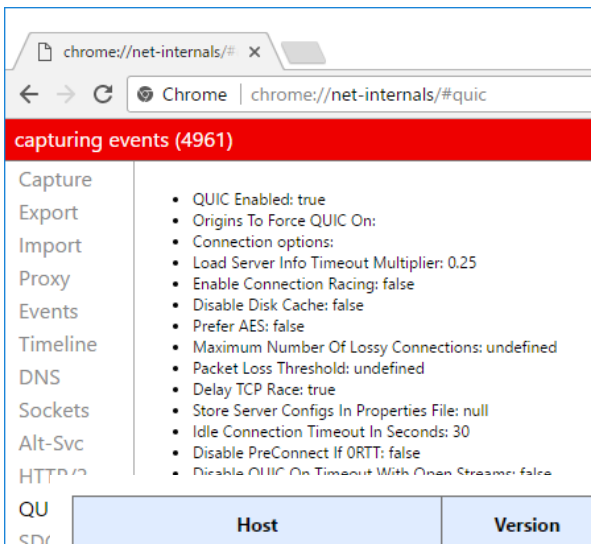


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.13	172.217.26.100	UDP	175	50270 → 443 Len=133
2	0.012014	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	469	443 → 50270 Len=427
3	0.019206	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	94	50270 → 443 Len=52
4	0.031990	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	75	443 → 50270 Len=33
5	1.219255	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	347	50270 → 443 Len=305
6	1.259549	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	69	443 → 50270 Len=27
7	1.445828	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	217	443 → 50270 Len=175
8	1.447768	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	100	50270 → 443 Len=58
9	1.448612	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	115	50270 → 443 Len=73
10	1.460056	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	113	50270 → 443 Len=71

#9 QUIC dissection

check QUIC connection

- Open the Chrome and type `chrome://net-internals/#quic` you can see the QUIC sessions
- Please check Connection UID



The screenshot shows the Chrome DevTools interface with the address bar set to `chrome://net-internals/#quic`. The page title is "capturing events (4961)". On the left, there is a sidebar with navigation options: Capture, Export, Import, Proxy, Events, Timeline, DNS, Sockets, Alt-Svc, HTTP/2, and QUIC. The main content area displays a list of QUIC connections with columns for Host, Version, Peer address, Connection UID, Active stream count, Active streams, Total stream count, Packets Sent, Packets Lost, Packets Received, and Connected.

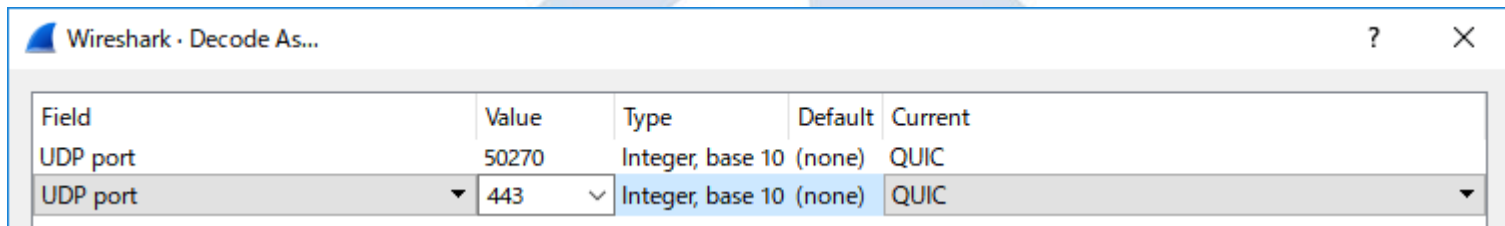
Host	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
apis.google.com:443	QUIC_VERSION_35	ogs.google.com:443	8126651583460386461	0	None	1	7	0	7	true
clients4.google.com:443	QUIC_VERSION_35	172.217.26.46:443	7635153523791641412	0	None	1	8	0	7	true
clients5.google.com:443	QUIC_VERSION_35	172.217.26.46:443	3964807834305814218	0	None	0	3	0	3	true
fonts.gstatic.com:443	QUIC_VERSION_35	ssl.gstatic.com:443	3002020916089671232	0	None	0	3	0	3	true
lh3.googleusercontent.com:443	QUIC_VERSION_35	216.58.197.129:443	1788128156380661810	0	None	0	4	0	4	true
notifications.google.com:443	QUIC_VERSION_35	216.58.197.238:443	5902717492422574525	0	None	0	3	0	3	true
play.google.com:443	QUIC_VERSION_35	216.58.197.238:443	10382952661970550630	0	None	1	7	0	5	true
www.google.co.jp:443	QUIC_VERSION_35	216.58.197.227:443	1168159024473252919	0	None	20	70	0	97	true
www.google.com:443	QUIC_VERSION_35	172.217.26.100:443	6629889265344322331	0	None	1	5	0	4	true
www.gstatic.com:443	QUIC_VERSION_35	216.58.197.131:443	8479664014582986847	0	None	0	4	0	4	true

Host	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
apis.google.com:443	QUIC_VERSION_35	ogs.google.com:443	8126651583460386461	0	None	1	7	0	7	true
clients4.google.com:443	QUIC_VERSION_35	172.217.26.46:443	7635153523791641412	0	None	1	8	0	7	true
clients5.google.com:443	QUIC_VERSION_35	172.217.26.46:443	3964807834305814218	0	None	0	3	0	3	true
fonts.gstatic.com:443	QUIC_VERSION_35	ssl.gstatic.com:443	3002020916089671232	0	None	0	3	0	3	true
lh3.googleusercontent.com:443	QUIC_VERSION_35	216.58.197.129:443	1788128156380661810	0	None	0	4	0	4	true
notifications.google.com:443	QUIC_VERSION_35	216.58.197.238:443	5902717492422574525	0	None	0	3	0	3	true
play.google.com:443	QUIC_VERSION_35	216.58.197.238:443	10382952661970550630	0	None	1	7	0	5	true
www.google.co.jp:443	QUIC_VERSION_35	216.58.197.227:443	1168159024473252919	0	None	20	70	0	97	true
www.google.com:443	QUIC_VERSION_35	172.217.26.100:443	6629889265344322331	0	None	1	5	0	4	true
www.gstatic.com:443	QUIC_VERSION_35	216.58.197.131:443	8479664014582986847	0	None	0	4	0	4	true

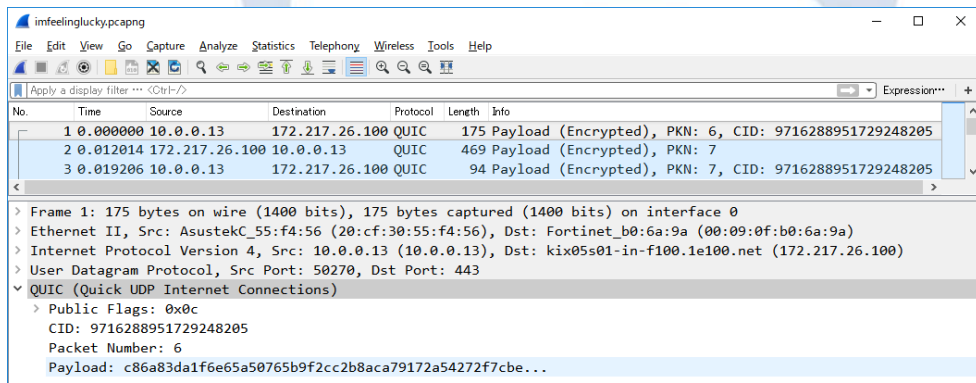
#9 QUIC dissection

Decode as QUIC

- Click udp packet and right click “Decode As...” and choose non-well-known port as QUIC (this time port 50270)



- Wireshark recognize the port is used for QUIC



#9 QUIC dissection

QUIC architecture

- HTTP2 (eth:ip:tcp:ssl:http2)

```
> Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a)
> Internet Protocol Version 4, Src: 10.0.0.13 (10.0.0.13), Dst: 104.244.42.67 (104.244.42.67)
> Transmission Control Protocol, Src Port: 27964, Dst Port: 443, Seq: 393, Ack: 3035, Len: 56
> Secure Sockets Layer
v HyperText Transfer Protocol 2
```

- QUIC(eth:ip:udp:quic:http2(Encrypted))

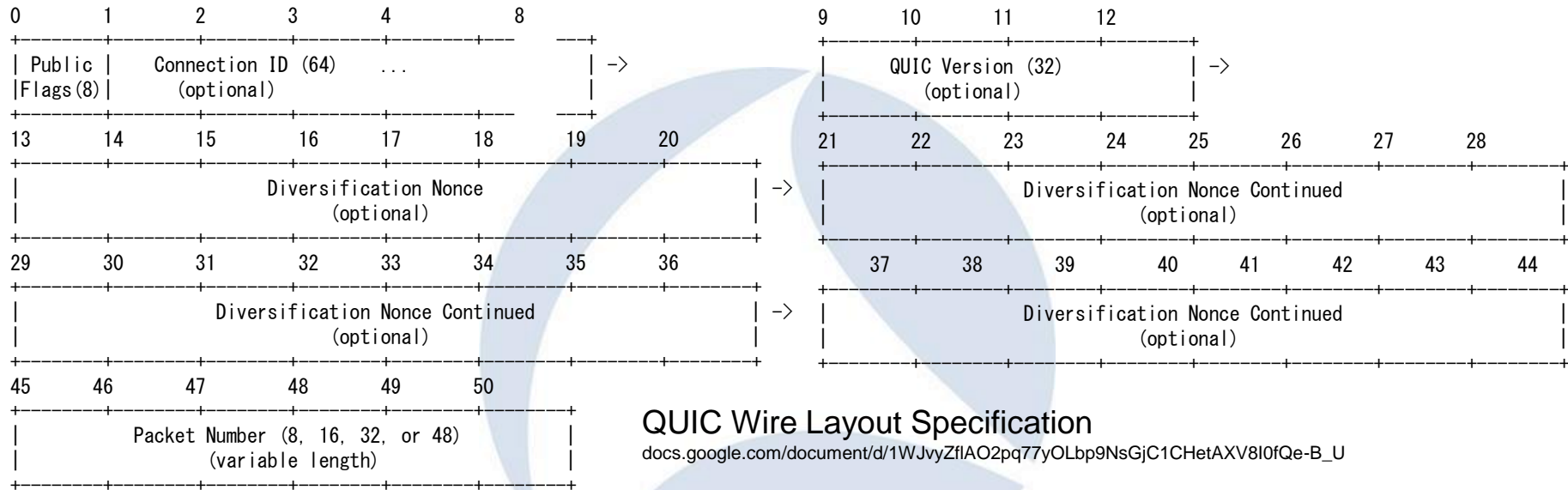
QUIC overlaps SSL/TLS+TCP on UDP

with Encryption, Authentication, Session establishment, Flow control, Error correction and Congestion Control

```
> Frame 1: 175 bytes on wire (1400 bits), 175 bytes captured (1400 bits) on interface 0
> Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a)
> Internet Protocol Version 4, Src: 10.0.0.13 (10.0.0.13), Dst: kix05s01-in-f100.1e100.net (172.217.26.100)
> User Datagram Protocol, Src Port: 50270, Dst Port: 443
v QUIC (Quick UDP Internet Connections)
  > Public Flags: 0x0c
  CID: 9716288951729248205
  Packet Number: 6
  Payload: c86a83da1f6e65a50765b9f2cc2b8aca79172a54272f7cbe...
```

#9 QUIC dissection

QUIC Public Packet Header



Connection ID: This is an unsigned 64 bit statistically random number selected by the client that is the identifier of the connection. Because QUIC connections are designed to remain established even if the client roams, the IP 4-tuple (source IP, source port, destination IP, destination port) may be insufficient to identify the connection. For each transmission direction, when the 4-tuple is sufficient to identify the connection, the connection ID may be omitted.

#9 QUIC dissection

Add Connection ID row and filter quic

Select packet 1 and open QUIC header, and select CID (quic.cid) and right click and select “apply as column” Set quic in Display Filter

The screenshot shows the Wireshark interface for a capture file named 'imfeelinglucky.pcapng'. The display filter is set to 'quic'. A custom column named 'CID' has been added to the packet list, showing the value '9716288951729248205' for all packets. Packet 2 is selected, and its details pane shows the QUIC header structure. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	CID	Info
1	0.000000	10.0.0.13	kix05s01-in...	QUIC	175	9716288951729248205	Payload (Encrypted), PKN: 6, CID: 9716
2	0.012014	kix05s01-i...	10.0.0.13	QUIC	469		Payload (Encrypted), PKN: 7
3	0.019206	10.0.0.13	kix05s01-in...	QUIC	94	9716288951729248205	Payload (Encrypted), PKN: 7, CID: 9716
4	0.031990	kix05s01-i...	10.0.0.13	QUIC	75		Payload (Encrypted), PKN: 8
5	1.219255	10.0.0.13	kix05s01-in...	QUIC	347	9716288951729248205	Payload (Encrypted), PKN: 8, CID: 9716
6	1.259549	kix05s01-i...	10.0.0.13	QUIC	69		Payload (Encrypted), PKN: 9
7	1.445828	kix05s01-i...	10.0.0.13	QUIC	217		Payload (Encrypted), PKN: 10
8	1.447768	10.0.0.13	kix05s01-in...	QUIC	100	9716288951729248205	Payload (Encrypted), PKN: 9, CID: 9716
9	1.448612	10.0.0.13	kix05s01-in...	QUIC	115	9716288951729248205	Payload (Encrypted), PKN: 10, CID: 9716

> Frame 2: 469 bytes on wire (3752 bits), 469 bytes captured (3752 bits) on interface 0
> Ethernet II, Src: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9a), Dst: AsustekC_55:f4:56 (20:cf:30:55:f4:56)
> Internet Protocol Version 4, Src: kix05s01-in-f100.1e100.net (172.217.26.100), Dst: 10.0.0.13 (10.0.0.13)
> User Datagram Protocol, Src Port: 443, Dst Port: 50270
> QUIC (Quick UDP Internet Connections)

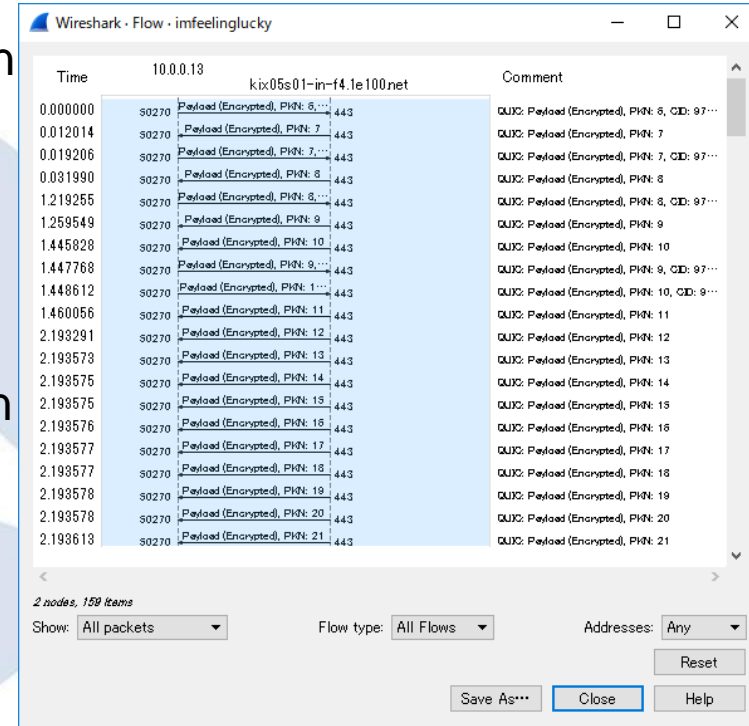
```
0000 20 cf 30 55 f4 56 00 09 0f b0 6a 9a 08 00 45 00  .0U.V.. .j...E.
0010 01 c7 00 00 40 00 34 11 73 dc ac d9 1a 64 0a 00  ....@.4. s....d..
0020 00 0d 01 bb c4 5e 01 b3 10 93 00 07 64 b0 86 a2  ....^.....d...
0030 85 68 80 8c c6 1a 53 1e 4c 7e 49 91 6a eb 29 28  .h....S. L~I.j.)(
0040 dd e0 10 98 2a fc ff 44 27 e1 e5 7c 68 fd c7 40  ...*..D '|h..@
```

imfeelinglucky | Packets: 159 · Displayed: 159 (100.0%) · Load time: 0:0.3 | Profile: Default

#9 QUIC dissection

Add Connection ID row and filter quic

- Now Wireshark can not decrypt QUIC negotiation but we can check and find traffic of QUIC
- Create flow graph with displayed packet, There are few RTT in this connection.
- You can find there are no 3 way handshake, There are few RTT in this connection.
- QUIC use sometimes no RTT for resumption and sometimes a few RTT for starting connection
- QUIC works if client IP and port changed, because QUIC has NAT-Traversal mechanism



The screenshot shows a Wireshark window titled "Wireshark · Flow · imfeelinglucky". The main display area shows a flow graph for the connection 10.0.0.13 to kix05s01-in-f4.1e100.net. The flow graph consists of 21 packets, all of which are QUIC payloads that have been encrypted. The packets are listed in a table with columns for Time, IP addresses, and Comment. The time values range from 0.000000 to 2.193613 seconds. The IP addresses are 10.0.0.13 and kix05s01-in-f4.1e100.net. The Comment column contains the text "QUIC: Payload (Encrypted), PI#N: N" for each packet, where N is the packet number (6 through 21). The flow graph is displayed as a vertical line of 21 nodes, each representing a packet. The nodes are connected by a single vertical line, indicating a continuous flow of data. The flow graph is titled "2 nodes, 159 items". At the bottom of the window, there are several controls: "Show: All packets", "Flow type: All Flows", "Addresses: Any", and buttons for "Reset", "Save As...", "Close", and "Help".

Time	10.0.0.13	kix05s01-in-f4.1e100.net	Comment	
0.000000	30270	Payload (Encrypted), PI#N: 6...	443	QUIC: Payload (Encrypted), PI#N: 6, CID: 97...
0.012014	30270	Payload (Encrypted), PI#N: 7...	443	QUIC: Payload (Encrypted), PI#N: 7
0.019206	30270	Payload (Encrypted), PI#N: 7...	443	QUIC: Payload (Encrypted), PI#N: 7, CID: 97...
0.031990	30270	Payload (Encrypted), PI#N: 8...	443	QUIC: Payload (Encrypted), PI#N: 8
1.219255	30270	Payload (Encrypted), PI#N: 8...	443	QUIC: Payload (Encrypted), PI#N: 8, CID: 97...
1.259549	30270	Payload (Encrypted), PI#N: 9...	443	QUIC: Payload (Encrypted), PI#N: 9
1.445028	30270	Payload (Encrypted), PI#N: 10...	443	QUIC: Payload (Encrypted), PI#N: 10
1.447768	30270	Payload (Encrypted), PI#N: 9...	443	QUIC: Payload (Encrypted), PI#N: 9, CID: 97...
1.448612	30270	Payload (Encrypted), PI#N: 11...	443	QUIC: Payload (Encrypted), PI#N: 10, CID: 9...
1.460056	30270	Payload (Encrypted), PI#N: 11...	443	QUIC: Payload (Encrypted), PI#N: 11
2.193291	30270	Payload (Encrypted), PI#N: 12...	443	QUIC: Payload (Encrypted), PI#N: 12
2.193573	30270	Payload (Encrypted), PI#N: 13...	443	QUIC: Payload (Encrypted), PI#N: 13
2.193575	30270	Payload (Encrypted), PI#N: 14...	443	QUIC: Payload (Encrypted), PI#N: 14
2.193575	30270	Payload (Encrypted), PI#N: 15...	443	QUIC: Payload (Encrypted), PI#N: 15
2.193576	30270	Payload (Encrypted), PI#N: 16...	443	QUIC: Payload (Encrypted), PI#N: 16
2.193577	30270	Payload (Encrypted), PI#N: 17...	443	QUIC: Payload (Encrypted), PI#N: 17
2.193577	30270	Payload (Encrypted), PI#N: 18...	443	QUIC: Payload (Encrypted), PI#N: 18
2.193578	30270	Payload (Encrypted), PI#N: 19...	443	QUIC: Payload (Encrypted), PI#N: 19
2.193578	30270	Payload (Encrypted), PI#N: 20...	443	QUIC: Payload (Encrypted), PI#N: 20
2.193613	30270	Payload (Encrypted), PI#N: 21...	443	QUIC: Payload (Encrypted), PI#N: 21

#10 Visualize using ErasticSearch + Kibana

Wireshark dissection is the source of big pcapng analysis

- I showed this demo at last Sharkfest Europe 2016
- Wireshark is almighty decoder, packet dissection is not only for trace file analysis within Wireshark itself, but also for the source of big data analysis
- We can recode everything in network using Wireshark, and export dissection result as JSON, JSON connect Wireshark with big data analysis tools such as Elastic search and Kibana.



- JSON (JavaScript Object Notation) is one of open source data format like XML, JSON is commonly used by many web application, data exchange. Tshark can also output packets as JSON
- Tshark JSON (EK) output can be used for open source data analysis tool, Elastic Search and Kibana.

#10 Visualize using ErasticSearch + Kibana

Set up Elasticsearch with Kibana environment

1. Download JDK, Curl and install, and set system environment variable
2. Download Elasticsearch and start server, check <http://localhost:9200>
3. Convert packet dissection data into Elasticsearch friendly JSON file
4. Entry packet dissection data (JSON) in Elasticsearch and check data
5. Modify the mapping, re-entry packet dissection data (JSON)
6. Download Kibana and start server, check <http://localhost:5601>
7. Access Kibana and set index
8. Search packet in full-text, visualize the packet and enjoy big data !

This time we are skipping STEP1-2, 4-7
and just enjoy Erastic Search and Kibana

#10 Visualize using ErasticSearch + Kibana

1. Download JDK, Curl and install, and set system environment variable

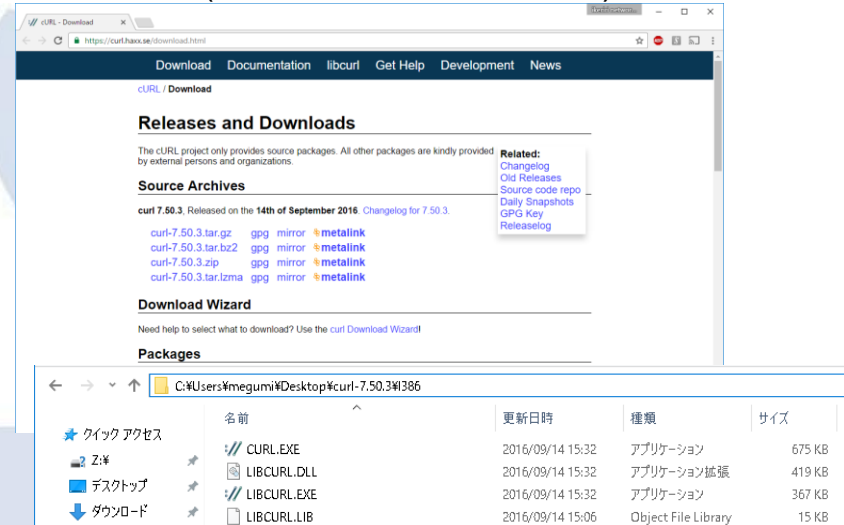
JDK(Java Developer Kit) 8 u101



<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Download `jdk-8u101-windows-x64.exe`
Execute and start setup program

Curl (web access command)

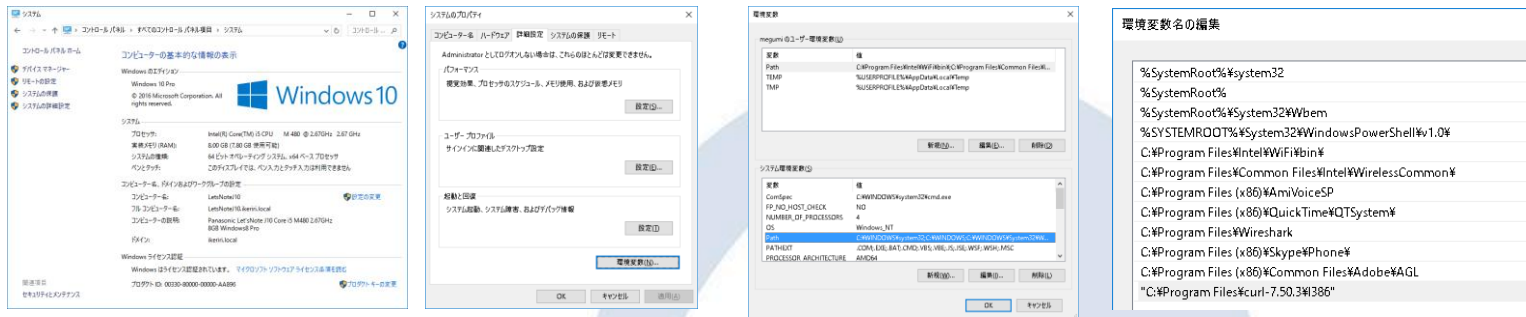


<https://curl.haxx.se/download.html>

Download `curl-7.50.3.cab`
Extract cab file and copy into "Program Files"
`C:\Program Files\curl-7.50.3\1386\curl.exe`

#10 Visualize using ErasticSearch + Kibana

1. Download JDK, Curl and install, and set system environment variable



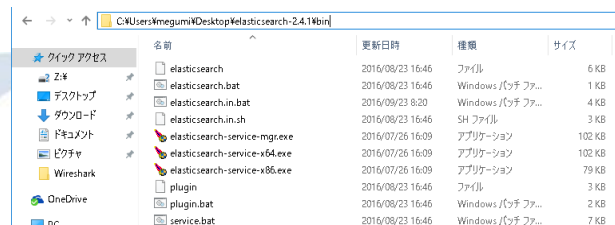
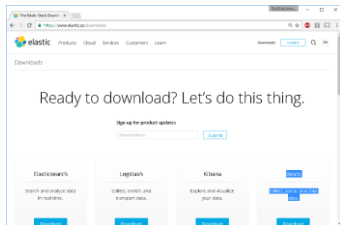
Control Panel > System > System setting > detail settings > environmental variable
set JAVA_HOME=C:¥Program Files¥Java¥jdk1.8.0_101
set Path=(current path);C:¥Program Files¥Java¥jdk1.8.0_101; C:¥Program Files¥curl-7.50.3¥I386

```
C:¥Users¥megumi>java -version
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
```

```
C:¥Users¥megumi>curl --version
curl 7.50.3 (i386-pc-win32) libcurl/7.50.3 WinSSL zlib/1.2.8
Protocols: dict file ftp ftps gopher http https imap imaps ldap pop3 pop3s rts
Features: AsynchDNS IPv6 Largefile SSI Kerberos SPNEGO NTLM SSL libz
```

#10 Visualize using ErasticSearch + Kibana

Download Elasticsearch and start server, check <http://localhost:9200>

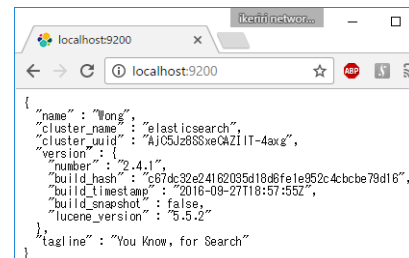


<https://www.elastic.co/downloads>

C:\Users\megumi\Desktop\elasticsearch-2.4.1\bin\elasticsearch.bat

1. Access <https://www.elastic.co/downloads>
2. Download elasticsearch-2.4.1.zip
3. Extract zip and open bin folder
4. Execute elasticsearch.bat
5. Check "started" in command prompt
6. Open <http://localhost:9200>

```
Elasticsearch 2.4.1
[2016-10-09 12:33:55.893][INFO ][node ] [Spirit of '76] version[2.4.1],
pid[8592], build[c67dc32/2016-09-27T18:57:55Z]
[2016-10-09 12:33:55.893][INFO ][node ] [Spirit of '76] initializing ..
[2016-10-09 12:33:56.904][INFO ][plugins ] [Spirit of '76] modules [reinde
x, lang-expression, lang-groovy], plugins [], sites []
[2016-10-09 12:33:56.941][INFO ][env ] [Spirit of '76] using [1] data
paths, mounts [{"O:"}], net usable_space [119.3gb], net total_space [211.5gb],
spins? [unknown], types [NFS]
[2016-10-09 12:33:56.941][INFO ][env ] [Spirit of '76] heap size [990.
7mb], compressed_ordinary_object_pointers [true]
[2016-10-09 12:33:59.877][INFO ][node ] [Spirit of '76] initialized
[2016-10-09 12:33:59.877][INFO ][node ] [Spirit of '76] starting ..
[2016-10-09 12:34:00.616][INFO ][transport ] [Spirit of '76] publish_address
[127.0.0.1:9200], bound_addresses [127.0.0.1:9200], [:::1]:9200
[2016-10-09 12:34:00.623][INFO ][discovery ] [Spirit of '76] elasticsearch/e
H0GcXLRJOZtZStgEKmg
[2016-10-09 12:34:04.739][INFO ][cluster.service ] [Spirit of '76] new_master [Spi
rit of '76] [eH0GcXLRJOZtZStgEKmg] [127.0.0.1] [127.0.0.1:9200], reason: zen-disco-join(elec
ted_as_master, [0] joins received)
[2016-10-09 12:34:04.807][INFO ][gateway ] [Spirit of '76] recovered [0] i
ndices into cluster_state
[2016-10-09 12:34:05.191][INFO ][http ] [Spirit of '76] publish_address
[127.0.0.1:9200], bound_addresses [127.0.0.1:9200], [:::1]:9200
[2016-10-09 12:34:05.191][INFO ][node ] [Spirit of '76] started
```



#10 Visualize using ErasticSearch + Kibana

3. Convert packet dissection data into Elasticsearch friendly JSON file

tshark -T ek -r stream.pcapng > packet.json

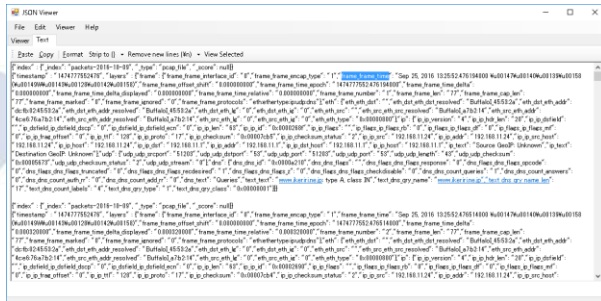
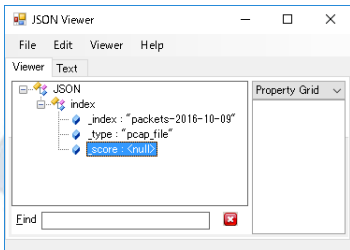
1: Read stream.pcapng, convert EK (Elasticsearch friendly JSON file)

2: Redirect output stream to a file named packet.json

コマンドプロンプト

```
C:\Users\megumi\Desktop>tshark -T ek -r stream.pcapng > packet.json
```

```
C:\Users\megumi\Desktop>_
```



#10 Visualize using ErasticSearch + Kibana

4. Entry packet dissection data (JSON) in Elasticsearch and check data

curl -XPOST url @filename : use POST method to send data to server url
curl -XPOST http://localhost:9200/_bulk --data-binary @packet.json

```
C:\Users\megumi\Desktop>curl -XPOST http://localhost:9200/_bulk --data-binary @packet.json
{"took":5748,"errors":false,"items":[{"create":{"_index":"packets-2016-10-09","type":"pcap_eC-Ci","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cj","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Ck","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cl","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cm","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cn","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Co","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cp","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cq","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cr","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cs","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Ct","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cu","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cv","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cw","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cx","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cy","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-Cz","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201},"create":{"_index":"packets-2016-10-09","type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-C0","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"status":201}}]}
```

```
C:\Users\megumi\Desktop>curl -XPOST
http://localhost:9200/_bulk --data-binary @packet.json
{"took":5748,"errors":false,"items":[{"create":{"_index":"p
ackets-2016-10-
09","_type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-
Ci","_version":1,"_shards":{"total":2,"successful":1,"faile
d":0},"status":201}},...
,{"create":{"_index":"packets-2016-10-
09","_type":"pcap_file","_id":"AVenqQppWAnL3ZFeC-
C0","_version":1,"_shards":{"total":2,"successful":1,"fail
ed":0},"status":201}}]}
```

#10 Visualize using ErasticSearch + Kibana

4. Entry packet dissection data (JSON) in Elasticsearch and check data

Access http://localhost:9200/_search?pretty (`_search` means all index (pretty output)) and check data entry

```
localhost:9200/_search?pretty
localhost:9200/_search?pretty

{
  "took": 40,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 19,
    "max_score": 1.0,
    "hits": [
      {
        "_index": "packets-2016-10-09",
        "_type": "pcap_file",
        "_id": "AVenqQpqWANL3ZFeC-Ck",
        "_score": 1.0,
        "_source": {
          "timestamp": "147477554499",
          "layers": {
            "frame": {
              "frame_frame_interface_id": "0",
              "frame_frame_encap_type": "1",
              "frame_frame_line": "8ap 25, 2016 13:25:54.499748000 Wu00147Wu00140",
              "frame_frame_offset_shift": "0.000000000",
              "frame_frame_line_epoch": "147477554.499748000",
              "frame_frame_line_delta": "2.023234000",
              "frame_frame_line_delta_displayed": "2.023234000",
              "frame_frame_line_relative": "2.023554000",
              "frame_frame_number": "3",
              "frame_frame_len": "118",
              "frame_frame_cap_len": "118",
              "frame_frame_marked": "0",
              "frame_frame_ignored": "0",
              "frame_frame_protocols": "eth:ethertype:ip:udp:dns"
            }
          }
        }
      }
    ]
  }
}
```

```
{
  "took": 40,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 19,
    "max_score": 1.0,
    "hits": [ {
      "_index": "packets-2016-10-09",
      "_type": "pcap_file",
      "_id": "AVenqQpqWANL3ZFeC-Ck",
      "_score": 1.0,
      "_source": {
        "timestamp": "147477554499",
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.11.24	192.168.11.1	DNS	77	Standard query 0xa210 A www.ikeriri.n...
2	0.000320	192.168.11.24	192.168.11.1	DNS	77	Standard query 0x27c9 AAAA www.ikeriri.n...
3	2.023234	192.168.11.1	192.168.11.24	DNS	116	Standard query response 0xa210 A www.ikeriri.n...
4	0.000808	192.168.11.1	192.168.11.24	DNS	168	Standard query response 0x27c9 AAAA www.ikeriri.n...
5	0.000796	192.168.11.24	asashina.ikeriri.ne.jp	TCP	66	57013->80 [SYN] Seq=0 Win=8192 Len=0 M...
6	0.013221	asashina.ikeriri.ne.jp	192.168.11.24	TCP	62	80->57013 [SYN, ACK] Seq=0 Ack=1 Win=84...
7	0.000139	192.168.11.24	asashina.ikeriri.ne.jp	TCP	54	57013->80 [ACK] Seq=1 Ack=1 Win=65044 ...
8	0.000272	192.168.11.24	asashina.ikeriri.ne.jp	HTTP	452	GET /wireshark/cheer.html HTTP/1.1
9	0.024883	asashina.ikeriri.ne.jp	192.168.11.24	HTTP	1215	HTTP/1.1 200 OK (text/html)
10	0.006790	192.168.11.24	asashina.ikeriri.ne.jp	HTTP	437	GET /wireshark/chiyodanyan.jpg HTTP/1...
11	0.054820	asashina.ikeriri.ne.jp	192.168.11.24	TCP	1468	[TCP segment of a reassembled PDU]
12	0.001371	asashina.ikeriri.ne.jp	192.168.11.24	TCP	1468	[TCP segment of a reassembled PDU]
13	0.000003	asashina.ikeriri.ne.jp	192.168.11.24	HTTP	96	HTTP/1.1 200 OK (JPEG JFIF image)
14	0.000115	192.168.11.24	asashina.ikeriri.ne.jp	TCP	54	57013->80 [ACK] Seq=782 Ack=4032 Win=6...
15	0.051055	192.168.11.24	asashina.ikeriri.ne.jp	HTTP	398	GET /favicon.ico HTTP/1.1
16	0.016414	asashina.ikeriri.ne.jp	192.168.11.24	TCP	1468	[TCP segment of a reassembled PDU]
17	0.001206	asashina.ikeriri.ne.jp	192.168.11.24	HTTP	296	HTTP/1.1 200 OK (image/x-ico)
18	0.000127	192.168.11.24	asashina.ikeriri.ne.jp	TCP	54	57013->80 [ACK] Seq=1126 Ack=5688 Win=...
19	1.496935	192.168.11.1	192.168.11.24	TCP	74	32830->80 [SYN] Seq=0 Win=5840 Len=0 M...

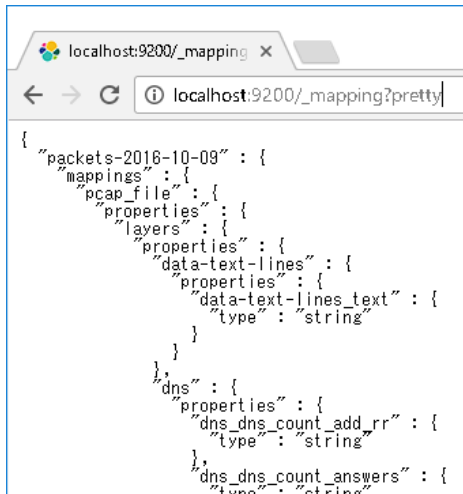
#10 Visualize using ErasticSearch + Kibana

4. Entry packet dissection data (JSON) in Elasticsearch and check data

If you failed to entry data , use curl “curl -XDELETE http://localhost:9200/*”

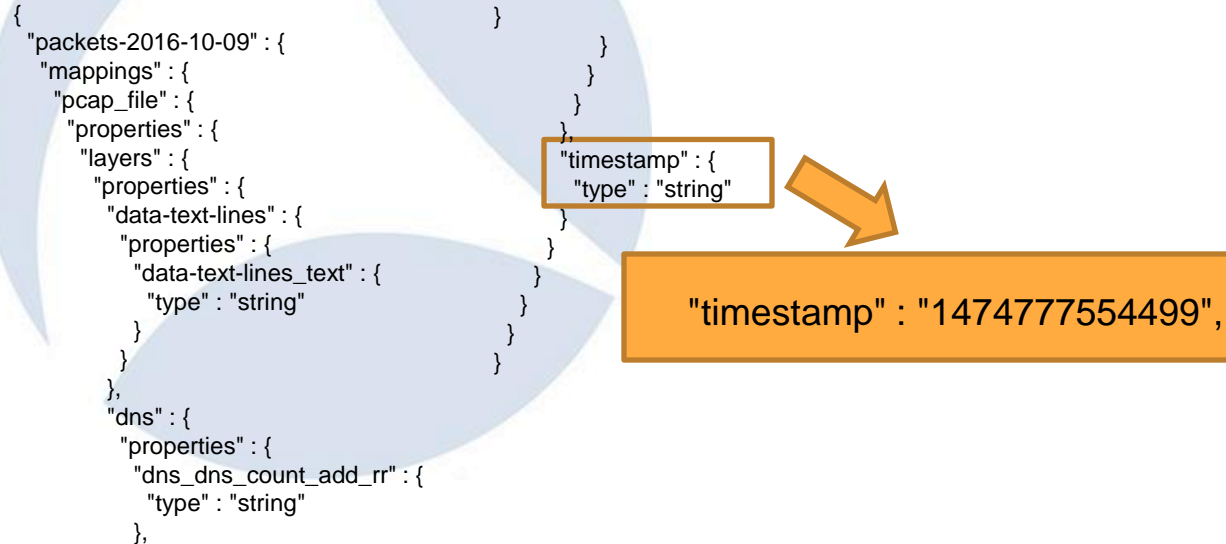
```
C:\Users\megumi\Desktop>curl -XDELETE http://localhost:9200/*  
{"acknowledged":true}
```

Check data mappings, open browser “http://localhost:9200/_mapping?pretty”



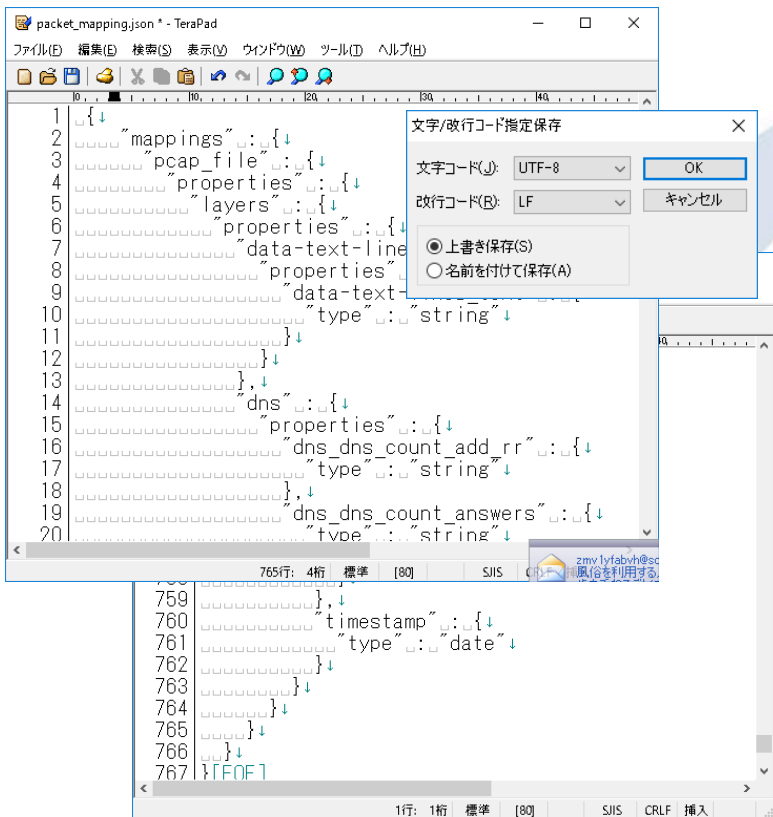
```
localhost:9200/_mapping x  
localhost:9200/_mapping?pretty  
{  
  "packets-2016-10-09" : {  
    "mappings" : {  
      "pcap_file" : {  
        "properties" : {  
          "layers" : {  
            "properties" : {  
              "data-text-lines" : {  
                "properties" : {  
                  "data-text-lines_text" : {  
                    "type" : "string"  
                  }  
                }  
              }  
            }  
          }  
        }  
      }  
    }  
  },  
  "dns" : {  
    "properties" : {  
      "dns_dns_count_add_rr" : {  
        "type" : "string"  
      }  
      "dns_dns_count_answers" : {  
        "type" : "string"  
      }  
    }  
  }  
}
```

```
{  
  "packets-2016-10-09" : {  
    "mappings" : {  
      "pcap_file" : {  
        "properties" : {  
          "layers" : {  
            "properties" : {  
              "data-text-lines" : {  
                "properties" : {  
                  "data-text-lines_text" : {  
                    "type" : "string"  
                  }  
                }  
              }  
            }  
          }  
        }  
      }  
    }  
  },  
  "dns" : {  
    "properties" : {  
      "dns_dns_count_add_rr" : {  
        "type" : "string"  
      }  
    }  
  }  
}
```



#10 Visualize using ErasticSearch + Kibana

5. Modify the mapping, re-entry packet dissection data (JSON)



http://localhost:9200/_mapping?pretty

Save mapping as “packet_mapping.json”

Delete header { “**packets-2016-10-09**” :

Modify mapping as

```
  "timestamp": {
    "type": "date"
```

Check character/return code UTF-8 / LF

Delete “curl -XDELETE http://localhost:9200/*”

Enter mapping as “curl -XPOST

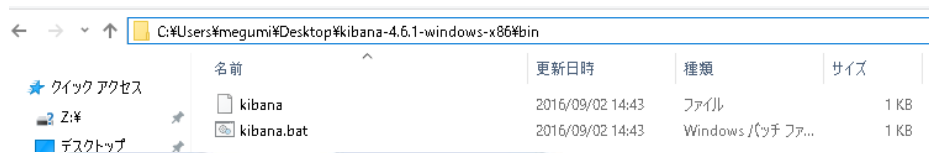
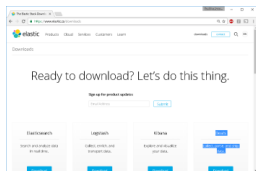
http://localhost:9200/packets-2016-10-09 --data-binary @packet_mapping.json”

Re-enter original data

curl -XPOST http://localhost:9200/_bulk --data-binary @packet.json

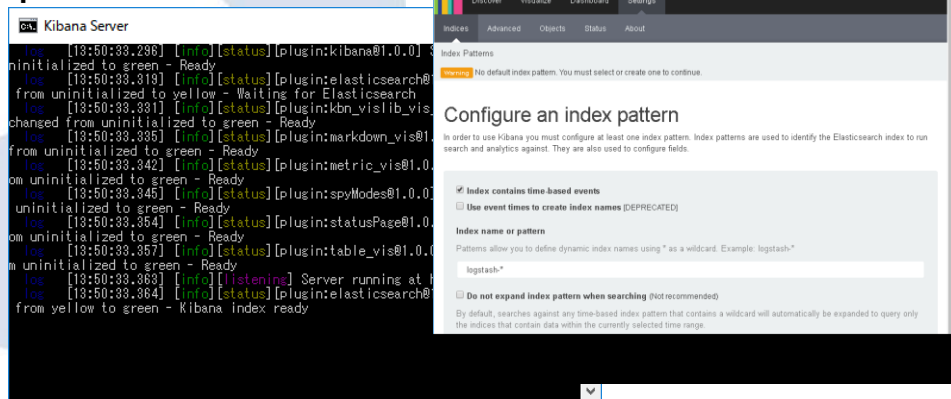
#10 Visualize using ErasticSearch + Kibana

6. Download Kibana and start server, check <http://localhost:5601>



<https://www.elastic.co/downloads> C:\Users\megumi\Desktop\kibana-4.6.1-windows-x86\bin\kibana.bat

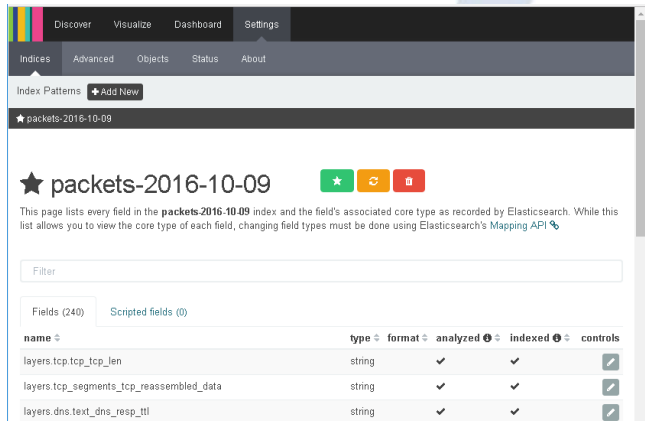
1. Access <https://www.elastic.co/downloads>
2. Download kibana-4.6.1-windows-x86.zip
3. Extract zip and open bin folder
4. Execute kibana.bat
5. Check “Kibana index ready” in prompt
6. Open <http://localhost:5601>



#10 Visualize using ErasticSearch + Kibana

7. Access Kibana and set index

1. Access `http://localhost:5601`
2. Check “index contains time-based events”
3. Set Index name or pattern as
“packets-2016-10-09” or “packets-*”
4. Set Time-field name as “timestamp”
5. Click “Create” button



Configure an index pattern

In order to use Kibana you must configure at least one index pattern. search and analytics against. They are also used to configure fields.

- Index contains time-based events
- Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard.

packets-2016-10-09

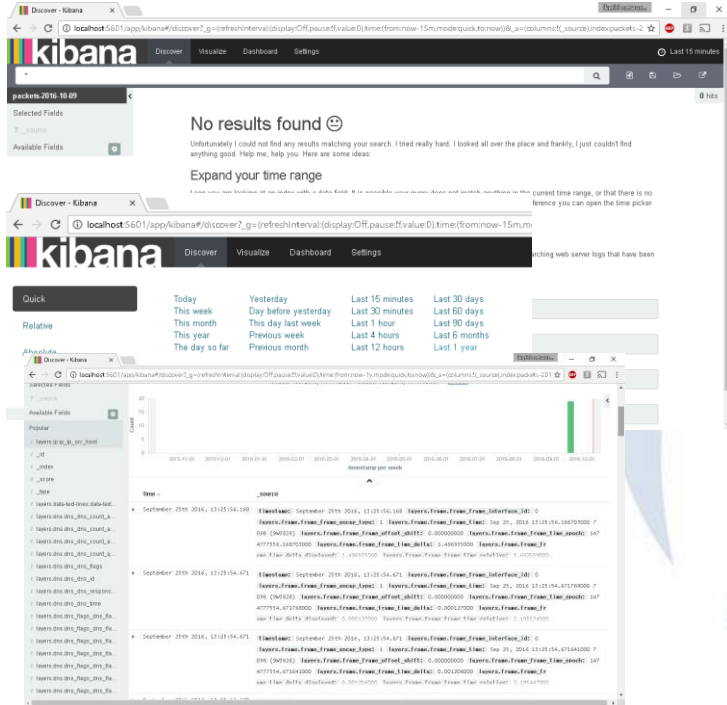
Time-field name [refresh fields](#)

timestamp

Create

#10 Visualize using ErasticSearch + Kibana

8. Search packet in full-text, visualize the packet and enjoy big data !



1. At First, use time picker to select the time of packets (just use “Last 1 year” is a good way)
2. Check histogram and left pane
3. Select layers.ip_ip_dst in left pane, click “add” and click “Visualize”, see and save the graph as name “IP”

This image shows the 'Visualize' page in Kibana. The visualization title is 'layers.ip_ip_dst'. The 'Quick Count' shows 19/19 records. The visualization is a bar chart showing the distribution of IP destinations. The top three results are:

IP Destination	Percentage
192.168.11.24	52.8%
211.5.104.181	38.8%
192.168.11.1	10.5%

The visualization is saved with the name 'IP Destination'. The search query `*geoip*` is visible in the search bar.

#10 Visualize using ErasticSearch + Kibana

8. Search packet in full-text, visualize the packet and enjoy big data !

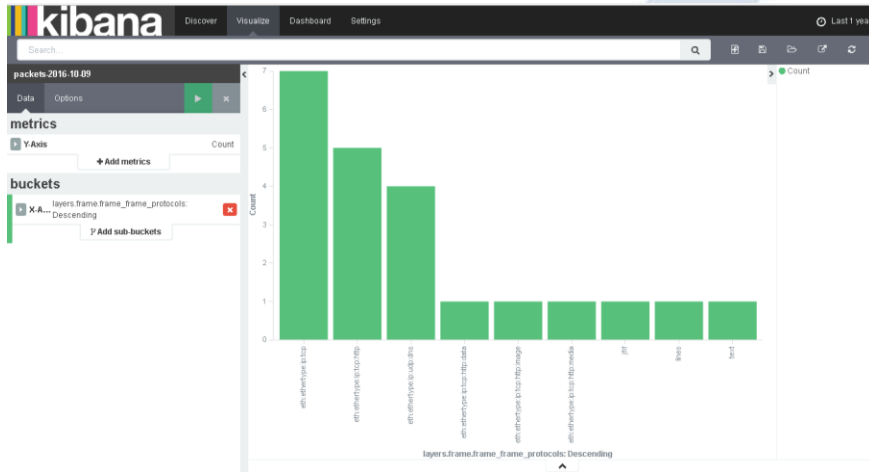
1. Select field `ip.ip_ip_src`, visualize and save the visualization as “IP Source”
2. Select “Visualize”, “Pie chart”, and “From a new search”
check “Select buckets type”, click “Split slices”, select “Terms” in Aggregation list box, choose field “`layers.frame.frame_frame_len`”, Apply changes, save the visualization as “Frame length”
3. Using “`layers.frame.frame_protocol`” and create pie chart, save as “Protocols”
4. Click “Dashboard” and set layout of these 4 Visualization



#10 Visualize using ErasticSearch + Kibana

8. Search packet in full-text, visualize the packet and enjoy big data !

Just a few step,
We can create grate visualization
of packets, and enjoy big data !!



Finish



Use Wireshark

SharkFest'17 US · Carnegie Mellon University · June 19–22, 2017

Thank you very much for your listening
Use Wireshark !

Thank you !
どうもありがとうございます !

