SharkFest'25 EUROPE

Dissector Developer Notes

Jaap Keuter – Core Developer

Intro / $ whoami

- Embedded software engineer in Telecom
- User, contributor and then core developer of Wireshark
- Mainly focus on dissectors
- Contributor to various other Open Source projects
- Besides enjoying various air sports

Intro / who are you ?

First some questions to get to know each other:

- Who has the Wireshark source code? Cloned or tarball?
- Who's writing dissectors? In C or Lua?
- Who's developing on which OS?
- Who has read the development documentation?

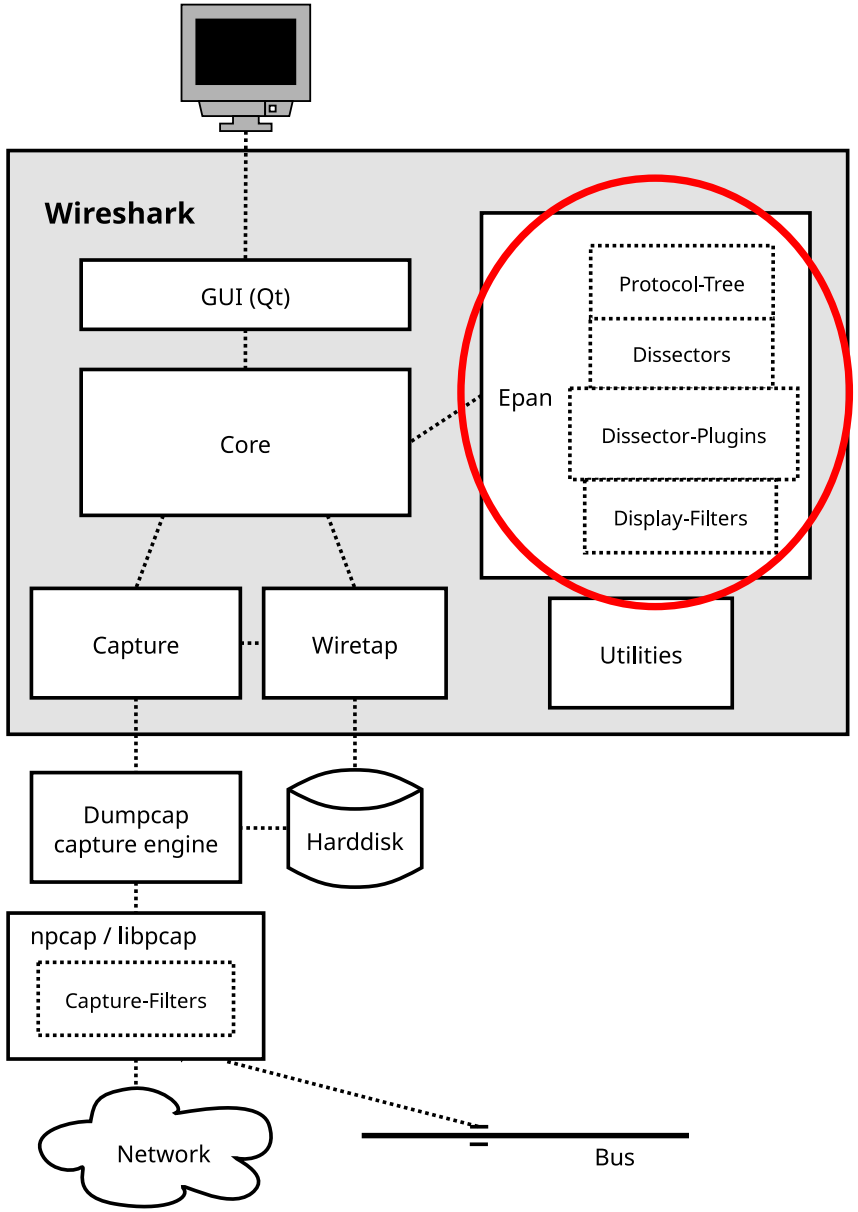
What will we discuss ?

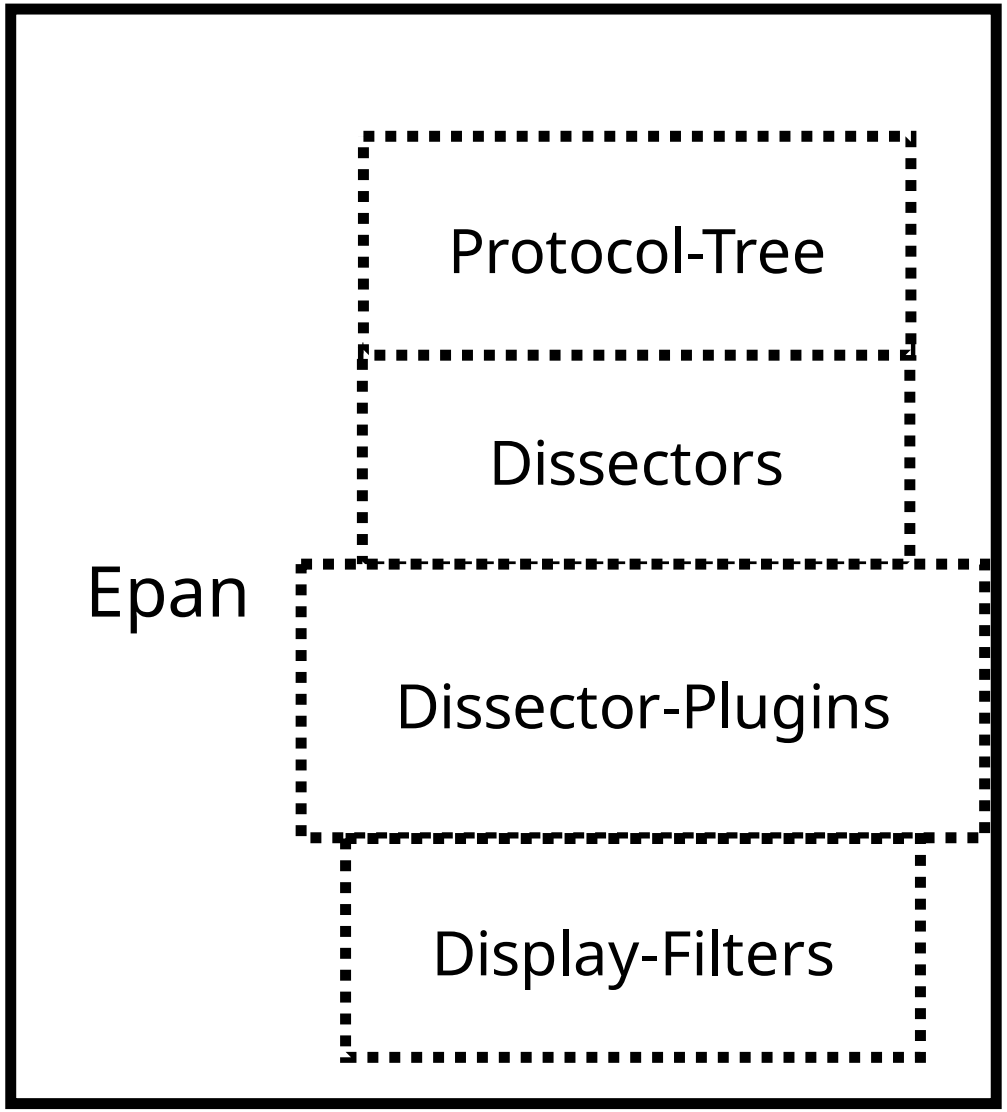
Dissectors.

Not just the EPAN API's, but what lays beyond them.

How do we think about dissector design?

Getting our bearings





What do we want to achieve ?


```
   13   2004-05-13 12:17:09.864896 Xerox_00:00:00 dialin-145
   14   2004-05-13 12:17:09.945011 fe:ff:20:00:.. 65.208.228
   15   2004-05-13 12:17:10.125270 Xerox_00:00:00 dialin-145
   16   2004-05-13 12:17:10.205385 fe:ff:20:00:.. 65.208.228
   17   2004-05-13 12:17:10.225414 fe:ff:20:00:.. 145.253.2.
   18   2004-05-13 12:17:10.295515 Xerox_00:00:00 dialin-145
   19   2004-05-13 12:17:10.325558 Xerox_00:00:00 dialin-145
   20   2004-05-13 12:17:10.686076 fe:ff:20:00:.. 65.208.228
   21   2004-05-13 12:17:10.806249 fe:ff:20:00:.. 65.208.228
   22   2004-05-13 12:17:10.806249 Xerox_00:00:00 dialin-145
   23   2004-05-13 12:17:10.946451 fe:ff:20:00:.. 65.208.228
```

```
Frame 1: Packet, 62 bytes on wire (496 bits), 62 bytes capt
Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst:
Internet Protocol Version 4, Src: 145.254.160.237 (145.254.
Transmission Control Protocol, Src Port: tip2 (3372), Dst P
    Source Port: tip2 (3372)
    Destination Port: Web (80)
    [Stream index: 0]
    [Stream Packet Number: 1]
  - [Conversation completeness: Complete, WITH_DATA (31)]
      ..0. .... = RST: Absent
      ...1 .... = FIN: Present
      .... 1... = Data: Present
      .... .1.. = ACK: Present
      .... ..1. = SYN-ACK: Present
      .... ...1 = SYN: Present
      [Completeness Flags: ·FDASS]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 951057939
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0111 .... = Header Length: 28 bytes (7)
  - Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    - .... .... ..1. = Syn: Set
```

Source Port (tcp.srcport), 2 bytes

~ : tshark — Konsole

File Edit View Bookmarks Plugins Settings Help

```
        [Source or Destination GeoIP Country: United States]
        [Destination GeoIP ISO Two Letter Country Code: US]
        [Source or Destination GeoIP ISO Two Letter Country Code: US]
        [Destination GeoIP AS Number: 17338]
        [Source or Destination GeoIP AS Number: 17338]
        [Destination GeoIP AS Organization: AOScloud, LLC.]
        [Source or Destination GeoIP AS Organization: AOScloud, LLC.]
        [Destination GeoIP Latitude: 37.751]
        [Source or Destination GeoIP Latitude: 37.751]
        [Destination GeoIP Longitude: -97.822]
        [Source or Destination GeoIP Longitude: -97.822]
    [Stream index: 0]
Transmission Control Protocol, Src Port: tip2 (3372), Dst Port: Web (80), Seq: 0, Len: 0
    Source Port: tip2 (3372)
    Destination Port: Web (80)
    [Stream index: 0]
    [Stream Packet Number: 1]
    [Conversation completeness: Complete, WITH_DATA (31)]
        ..0. .... = RST: Absent
        ...1 .... = FIN: Present
        .... 1... = Data: Present
        .... .1.. = ACK: Present
        .... ..1. = SYN-ACK: Present
        .... ...1 = SYN: Present
        [Completeness Flags: ·FDASS]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 951057939
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0111 .... = Header Length: 28 bytes (7)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Accurate ECN: Not set
        .... 0... .... = Congestion Window Reduced: Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
            [Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]
                [Connection establish request (SYN): server port 80]
                [Severity level: Chat]
                [Group: Sequence]
```

Dissector entry point

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Who’s calling ?

Who is calling `dissect_PROTOABBREV()` ?

For this we need to register the dissector with EPAN.

Dissector registration


```
/* Register the protocol with EPAN. */
void
proto_register_PROTOABBREV(void)
{
	proto_PROTOABBREV = proto_register_protocol("PROTONAME",
		"PROTOSHORTNAME", "PROTOFILTERNAME");

	PROTOABBREV_handle = register_dissector("PROTOABBREV",
		dissect_PROTOABBREV, proto_PROTOABBREV);
}
```

Who's calling ? (2)

Now that EPAN knows about `dissect_PROTOABBREV()` when does it call us?

For this we need to setup dissector handoff.

Dissection handoff


```c
#define PROTOABBREV_UDP_PORT 10000

/* Register for handoff to the dissector. */
void
proto_reg_handoff_PROTOABBREV(void)
{
      dissector_add_uint("udp.port",
            PROTOABBREV_UDP_PORT, PROTOABBREV_handle);
}
```

Dissector entry point (2)

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Testy, Virtual Buffer

epan/tvbuff.h

```
*
* Testy, Virtual(-izable) Buffer of uint8_t*'s
*
* "Testy" -- the buffer gets mad when an attempt is made to access data
*            beyond the bounds of the buffer. An exception is thrown.
*
* "Virtual" -- the buffer can have its own data, can use a subset of
*              the data of a backing tvbuff, or can be a composite of
*              other tvbuffs.
*
* Copyright (c) 2000 by Gilbert Ramirez <gram@alumni.rice.edu>
*
```

Access to packet data

epan/tvbuff.h

```
WS_DLL_PUBLIC uint8_t
tvb_get_uint8(tvbuff_t *tvb, const int offset);
```

Besides this one there are access functions for any imaginable type in a TVB. Use them!

Dissector entry point (3)

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Packet info struct

epan/packet_info.h

```
typedef struct _packet_info {
    <insane amount of parameters>
} packet_info;
```

Frame data struct

epan/frame_data.h

```
typedef struct _frame_data {
    <less insane amount of parameters>
} frame_data;
```

Dissector entry point (4)

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

The protocol tree

epan/proto.h

```
WS_DLL_PUBLIC proto_item *
proto_tree_add_item(proto_tree *tree,
    int hfindex, tvbuff_t *tvb,
    const int start, int length,
    const unsigned encoding);
```

Header field

doc/packet-PROTOABBREV.c

```
static hf_register_info hf[] = {
    { &hf_FIELDABBREV,
      { “FIELDNAME”, “FIELDFILTERNAME”,
        FT_FIELDTYPE, FIELDDISPLAY, FIELDCONVERT, BITMASK,
        “FIELDDESCR”, HFILL }
    }
};
```

Converting values

Often when creating your dissections you want to convert a number into a representative string. But can you trust the number read from the TVB to be valid?

Setup a `value_string` array and make sure to terminate that with a `{0, NULL}` tuple. Then use the `value_string` conversion functions, or stick it in the header field.

Dissector entry point (5)

doc/packet-PROTOABBREV.c

```c
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Out of band data

During dissection we want to pass out-of-band data between dissectors. If this is not part of packet_info, then the data parameter allows for this.

Since, in most cases, this is an unused parameter, use the “_U_” attribute to tell the compiler to ignore it.

Dissector entry point (6)

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
    /* create display subtree for the protocol */
    ti = proto_tree_add_item(tree, proto_PROTOABBREV, tvb, 0, -1,
ENC_NA);

    PROTOABBREV_tree = proto_item_add_subtree(ti, ett_PROTOABBREV);
```

Dissector design considerations

Packet dissection order

In what order are packets dissected, i.e., in what order is my dissector being called?

Wireshark: First sequential, then in random order.

Tshark: Once sequential, twice sequential with “-2” option.

Ergo: you can’t use static variables!

State across packets

`doc/README.request_response_tracking`

Packets are often not dissected in isolation. They can depend on data in earlier packets.

How to keep track of which packets belong together?

Conversations: An association defined by endpoint tuples, e.g., side A and B: IPv4 address + UDP port#

Streaming vs datagram

With datagram protocols (e.g., UDP) you know that you are getting a Protocol Data Unit worth of data.

How about streaming protocols (e.g. TCP) ?

You cannot expect the TCP dissector to give you your complete Protocol Data Units!

TCP PDU finder

epan/dissectors/packet-tcp.h

```
WS_DLL_PUBLIC void
tcp_dissect_pdus(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
    bool proto_desegment, unsigned fixed_len,
    unsigned (*get_pdu_len)(packet_info *, tvbuff_t *, int, void *),
    dissector_t dissect_pdu, void *dissector_data);
```

What to do with errors?

Remember that Wireshark gets put into action when things don't work. That may be when there's a protocol error.

To help the user, always try to show as much as possible.

What to do with errors? (2)

Use the safety of the EPAN facilities to cover for errors, e.g., TVB, value_string, etc.

Always check the validity of values read from the TVB before using it for loop counts, shifts, etc. These EPAN can't protect you against.

What we didn’t cover

- Columns
- Preferences
- Generated and hidden fields
- Per packet data
- Request and response tracking
- Heuristics
- Taps and Statistics
- Memory management
- Endianness conversion
- Encoding conversion
- ….

More info to be found

- In the documentation
  - Developers Guide
  - doc/README.*
  - Doxygen: https://www.wireshark.org/docs/wsar_html/
- In the source code

Feedback is much appreciated

https://conference.wireshark.org/sharkfest-25-europe-2025/talk/SSSB3Q/feedback/