Wireshark ohne Netzwerk

Augsburger Linux-Infotag
22. März 2014

Martin Kaiser

What?

- Wireshark is the standard tool for capturing and analyzing TCP/IP network traffic
  - supports many protocols
  - runs on different platforms
  - allows for fine-grained filtering
- this is useful for all kinds of captured data - not only when it comes from a TCP/IP network

Overview

- getting external data into Wireshark
- useful Wireshark features
- possible approaches
- examples
- adding a new protocol

About me

- writing embedded software for Digital TVs
- involved in creating the CI+ Pay-TV standard
- Wireshark Core Developer
- http://www.kaiser.cx

Getting external data into Wireshark

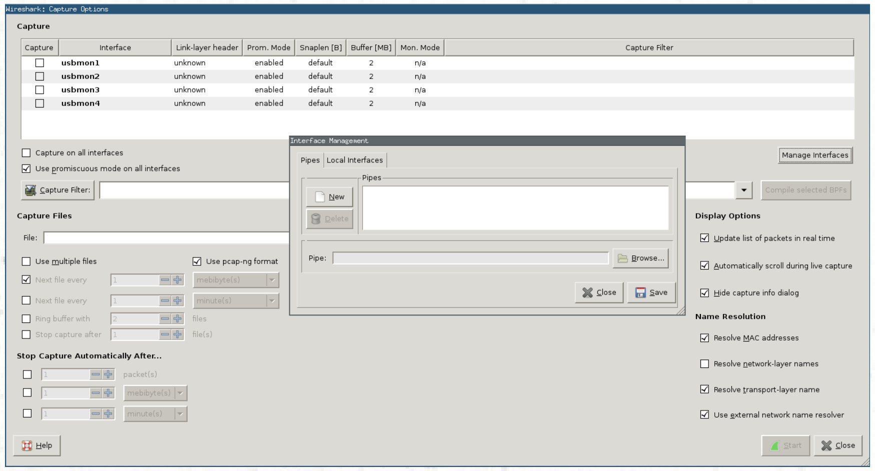
- existing data logger, driver software
- output data
  - bytes in a text file
  - some proprietary format
- how can I read and analyze such data with Wireshark?
  - offline
  - or ideally in real time

Wireshark

- monitoring of communication
- everything is packet-based
- only passive monitoring
  - no replay, no data injection, etc.

GUI: Interface list

- interface list
  - select multiple interfaces for capturing
  - refresh the interface list



Wireshark: Capture Options

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]	Buffer [MB]	Mon. Mode	Capture Filter
☐	**usbmon1**	unknown	enabled	default	2	n/a	
☐	**usbmon2**	unknown	enabled	default	2	n/a	
☐	**usbmon3**	unknown	enabled	default	2	n/a	
☐	**usbmon4**	unknown	enabled	default	2	n/a	

☐ Capture on all interfaces
☑ Use promiscuous mode on all interfaces

Capture Files

File:

☐ Use multiple files ☑ Use pcap-ng format
☑ Next file every 1 mebibyte(s)
☐ Next file every 1 minute(s)
☐ Ring buffer with 2 files
☐ Stop capture after 1 file(s)

Stop Capture Automatically After...

☐ 1 packet(s)
☐ 1 mebibyte(s)
☐ 1 minute(s)

Display Options

☑ Update list of packets in real time
☑ Automatically scroll during live capture
☑ Hide capture info dialog

Name Resolution

☑ Resolve MAC addresses
☐ Resolve network-layer names
☑ Resolve transport-layer name
☑ Use external network name resolver

Interface Management

Pipes | Local Interfaces

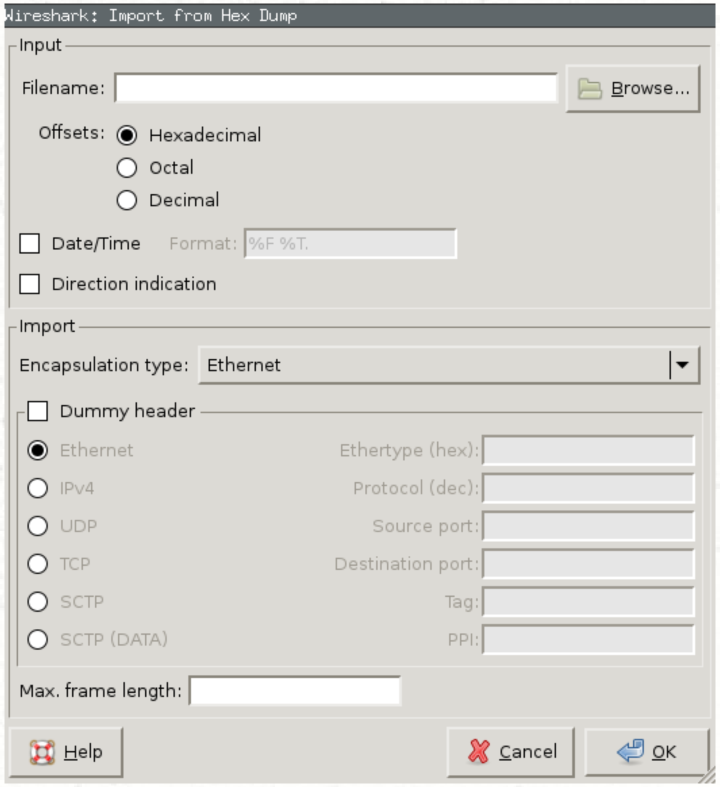
Pipes

Pipe:

- named pipes
  - GUI dialogue to add a pipe to the interface list
    - not permanent
  - similar to *wireshark -i <name>*

GUI: Import from hex dump

- *File / Import from Hex Dump...*
  - input file contains raw bytes and offsets
  - select a Data Link Type (DLT)
- similar to the command line tool *text2pcap*



Wireshark: Import from Hex Dump

Input

Filename: [ ] Browse...

Offsets: [✓ Hexadecimal] Octal Decimal

☐ Date/Time Format: %F %T.

☐ Direction indication

Import

Encapsulation type: Ethernet

☐ Dummy header

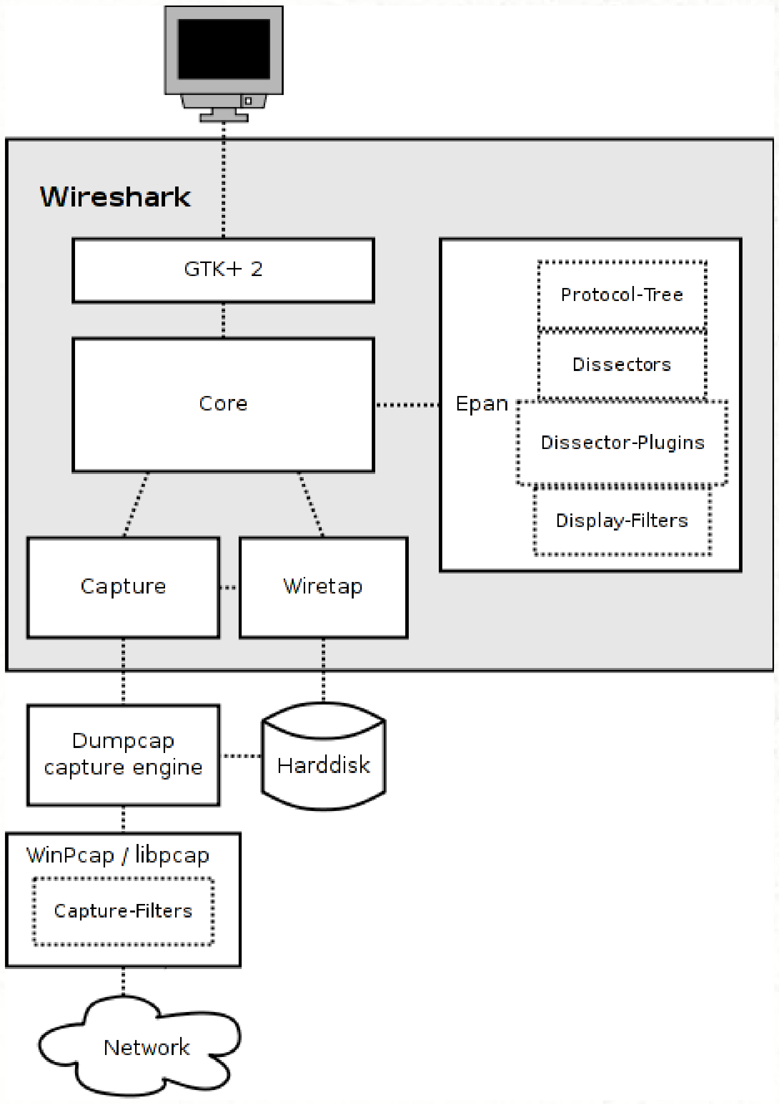
[✓ Ethernet] IPv4 UDP TCP SCTP SCTP (DATA)

Ethertype (hex): [ ]
Protocol (dec): [ ]
Source port: [ ]
Destination port: [ ]
Tag: [ ]
PPI: [ ]

Max. frame length: [ ]

Help | Cancel | OK

Architecture of Wireshark



- libpcap, winpcap
- dumpcap
- Wireshark
  - communication with dumpcap via pipe
  - wiretap
  - GUI: GTK, Qt, command line
  - display filter engine
  - dissectors
  - ASN.1 engine
  - plugin interface
  - lua, python interfaces

Protocols

- approx. 1100 protocols in Wireshark 1.10
  - networking, USB, mobile phone, digital tv, building automation, ...
- *frame* is always the "lowest layer" protocol
- link between protocols
  - DLT == Data Link Type
    - *www.tcpdump.org/linktypes.html*
  - TCP, UDP port, ...
  - basically, any filterable expression can be used as a selector for the upper layer protocol
  - to go from protocol A to protocol B, the code for A must support selection of upper-layer protocols
- protocol preferences

File formats

- PCAP format
  - one global header
    - DLT
    - snaplen, ...
  - one record per captured packet
    - trimmed to snaplen
- PCAPng
  - multiple interfaces
  - packet comment, capture comment, ...

Possible approaches

- encapsulate your data into network packets
- add support for your logger's file format
- convert your output data into PCAP(ng)
  - offline
  - in real-time
- add an interface for your logger to the interface list

Encapsulate data into network packets

- encapsulate your data e.g. into UDP packets to localhost:6000
- major disadvantage: dummy ethernet, IP, UDP layers
- capture/display filter to ignore other packets
- discard incoming packets on localhost:6000
  - *socat -u UDP-RECV:6000 /dev/null*

Add support for your file format

- read-only support is easy
- detect your file type
- read a packet
  - sequential read
  - random access

Convert to PCAP - text2pcap

```
$ cat data.txt
0   0a 0b 0c
3   01 02 03
0   a0 a1

$ text2pcap -l 50 data.txt data.pcap     (-l <data link type>)
Input from: data.txt
Output to: data.pcap
Output format: PCAP
Wrote packet of 6 bytes.
Wrote packet of 2 bytes.
Read 2 potential packets, wrote 2 packets (64 bytes).
```

Convert to PCAP - in real time

- modify your data logger's software to output PCAP
  - libpcap API (ANSI C)
    - bindings for perl, python, ...
- send PCAP output to a named pipe
  - named pipe supports only PCAP (not PCAPng)
  - pipe read blocks until the PCAP header was read
- start / stop capture
  - both in Wireshark and in the data logger software
  - they're not synchronised

Extcap

- new interface in Wireshark's interface list
  - provided by a separate executable (so-called *extcap*)
- Wireshark calls your extcap for
  - interface detection
  - interface configuration
  - start / stop capturing
- extcap drives the data logger, sends captured data to a named pipe
- will be available in the next major release

Example: capturing USB data

- works only if supported by OS / libpcap
- Linux
  - *modprobe usbmon*
  - make */dev/usbmon** readable for the wireshark user
- Windows: USBPcap
  - *http://desowin.org/usbpcap/*

Example: HDCP

- HDCP uses an I^{2}C bus on two pins of an HDMI cable
- logging hardware writes a text file
- DLT for I^{2}C
  - *www.tcpdump.org/linktypes.html*
  - needs a protocol-specific header
- use *text2pcap* for converting to a PCAP file

Example: Digital TV

- MPEG2 Transport Stream (TS)
  - contains multiple TV programs
  - a sequence of 188 byte packets
  - packet header includes a packet identifier (PID)
    - all packets with the same PID are one Elementary Stream (ES)
    - an ES may contain audio, video or one of several tables with additional infos

Example: Digital TV (II)

- read-only support for MPEG2 TS files (which contain only the raw TS packets)
- TS header, audio, video, tables are implemented as protocols
  - protocol selection based on PID

Extcap demo

A protocol dissector

- where do I attach it to?
  - DLT, TCP/UDP port, USB class, ...
- the dissector is called for each matching packet
  - parameters: *tvbuff, pinfo, tree*
- dissect your data
- generate tree entries, subtrees
- populate the columns
- create filterable items

Thank you for your attention.

Questions?

Slides at www.kaiser.cx

Freedigitalphotos/Master Isolated Images