

Wireshark Dissectors – Basic

June 15, 2010

Gerald Combs

Lead Developer | Wireshark

SHARKFEST '10

Stanford University

June 14-17, 2010

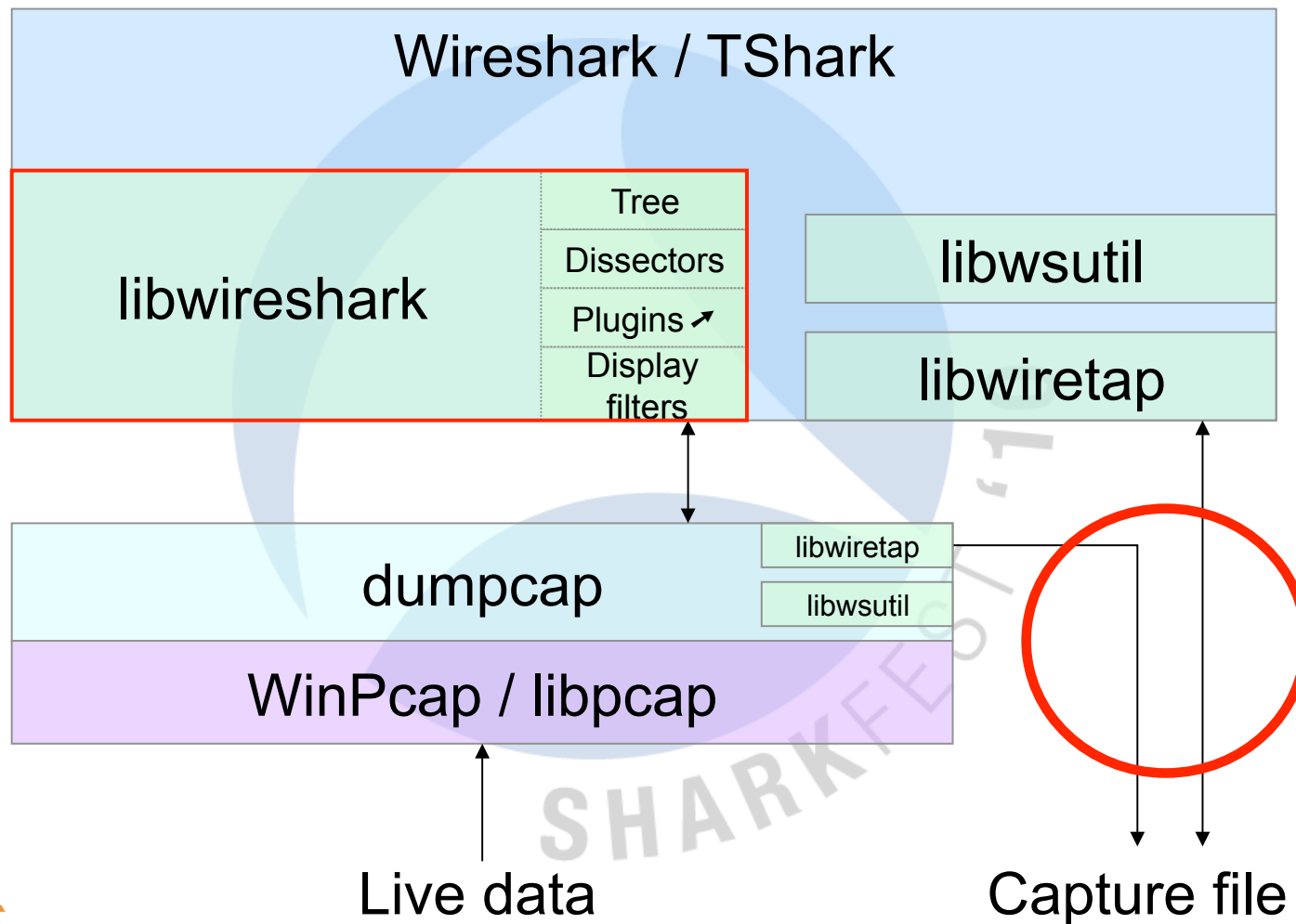


SHARKFEST '10

Wireshark Development

- Distributed
- Plain old boring C
- Multi-platform
- Multi-interface
- GPL 2-or-later
- SVN (at least until tomorrow afternoon)
- Rapid development pace

Application Architecture



Build Requirements

- Source code (of course)
- Visual C++ || gcc || SunPro C++ (|| LLVM?)
- Libraries: WinPcap/libpcap, GLib, GTK+, zlib, GNUTLS, c-ares, libsmi, ...
- Support tools: Python, Perl, Linux/UNIX shell

Build Requirements - Windows

- Visual C++ 6.0 to **2008**
- Python 2.4+ (No 3.0 yet)
- Optional: NSIS, TortoiseSVN

- Not optional: Cygwin



Getting the Code

- Subversion

<http://anonsvn.wireshark.org/wireshark/trunk>

<http://anonsvn.wireshark.org/wireshark/trunk-1.4>

- Tar files

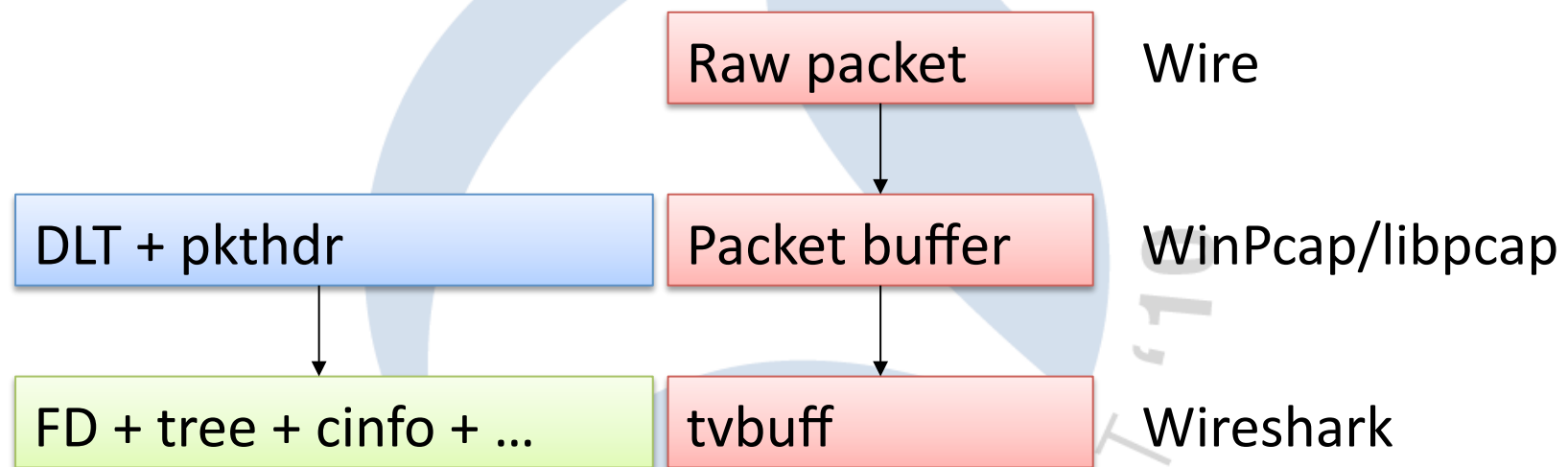
<http://www.wireshark.org/download/src>

- Want to send us a patch? `svn diff`

Source Directory Overview

<i>root</i>	CLI applications, common code
<i>doc</i>	READMEs, man pages
<i>docbook</i>	Guides
<i>epan</i>	Dissection
<i>dissectors</i>	Built-in dissectors
<i>gtk</i>	User interface
<i>packaging</i>	Platform installers
<i>plugins</i>	Plugin dissectors
<i>wiretap</i>	File formats
<i>wsutil</i>	Shared utility routines

Packet Data + Metainformation



Core Data Structures

- You get:
 - tvbuff: Protocol data access
 - packet_info, frame_data: Packet meta-info
 - proto_tree: Detail tree
- You provide:
 - header_field_info

UI Element Origins

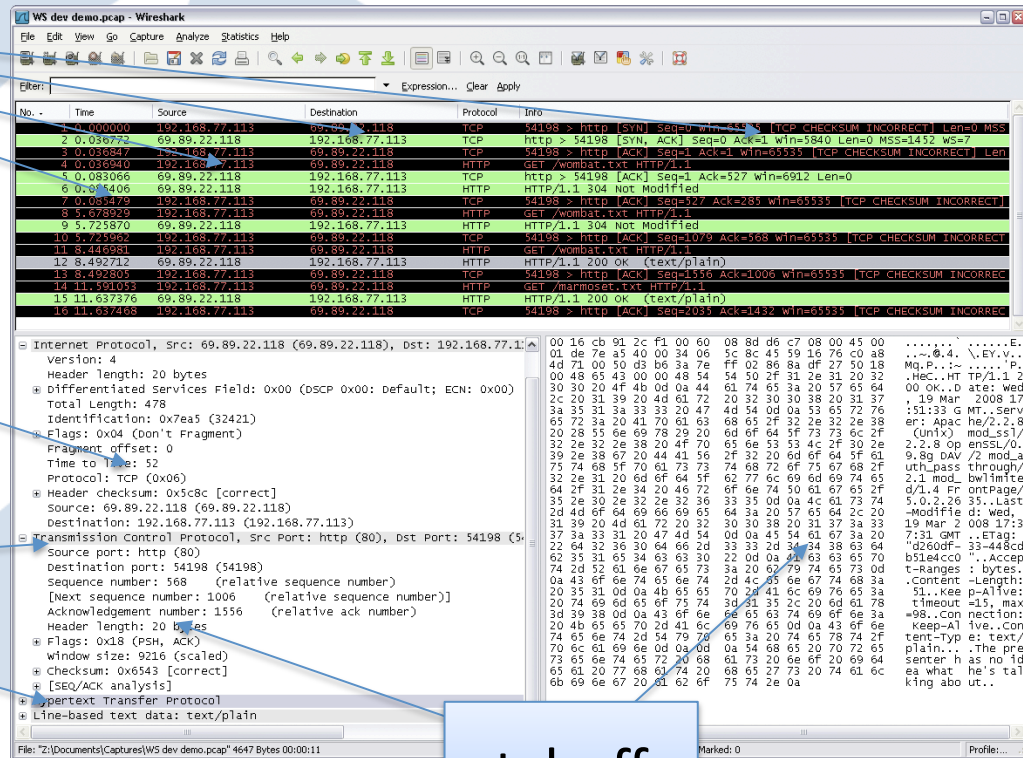
col_set_str()

proto_tree_add_*()

hfinfo

dissector_add()

tvbuff

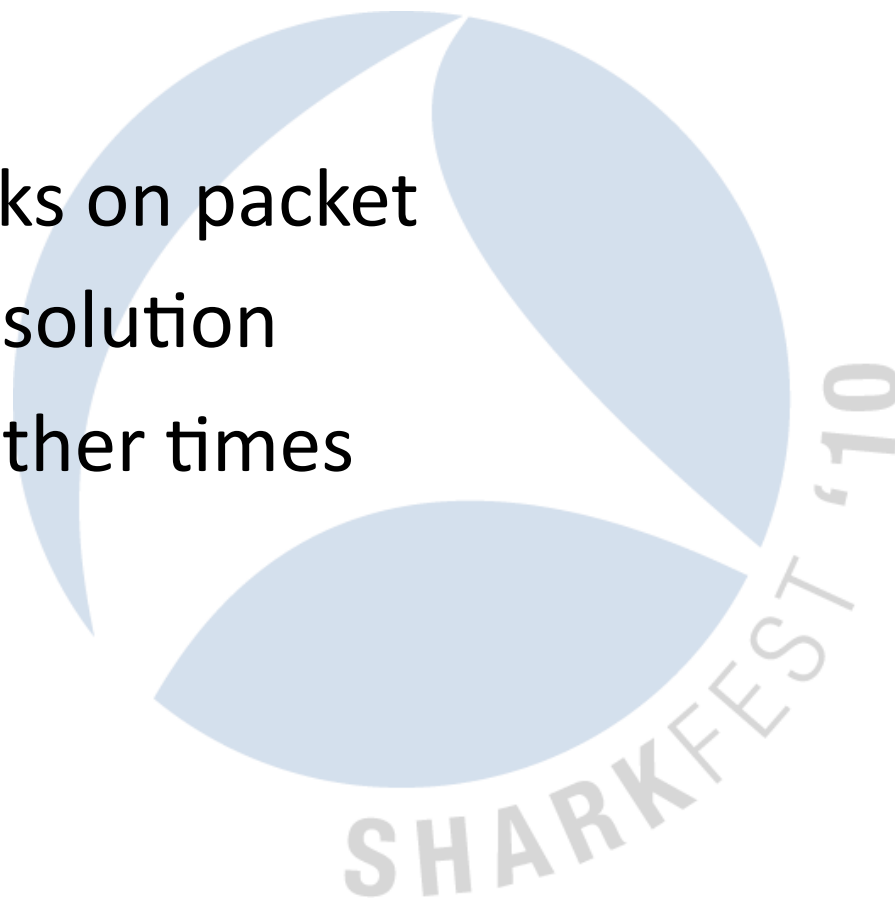


Getting Called

- Normal
 `dissector_add`
- Heuristic
 `heur_dissector_add`
- On the fly
 `dissector_add_handle`
 `find_dissector + call_dissector`

When Do You Get Called?

- File load
- User clicks on packet
- Name resolution
- Lots of other times



DNS Dissection Call Stack

```
dissect_dns_udp()      /* packet-dns.c */
decode_udp_ports()    /* packet-udp.c */
dissect_udp()         /* packet-udp.c */
dissect_ip()          /* packet-ip.c */
dissect_eth_common()  /* packet-eth.c */
dissect_frame()       /* packet-frame.c */
dissect_packet()      /* Add top-level structs */
process_packet()      /* DLT from wiretap */
main()
```

Parts of a Dissector

```
/* packet-xml.c
 * Routine for XML encapsulation of XML over 802.2 LLC
 * (and other protocols using SNAP 8)
 *
 * $Id$
 *
 * Wireshark - Network traffic analyzer
 * By Gerald Combs <gerald@wireshark.org>
 * Copyright 1998 Gerald Combs
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include 
#include "packet.h"
#include "epan/types.h"

static int proto_xml = -1;
static int hf_xml_type_ethertype = -1;
static int hf_xml_type_vlan_tag = -1;
static gint ett_xml = -1;

static const value_string proto_xml_type_vals[] = {
    { 0, NULL }
};

static dissector_table_t ethertype_subdissector_table;
static dissector_handle_t proto_xml_handle;
static dissector_handle_t data_handle;

/* Apparently XML had some scheme for encapsulating XML in 802.2 LLC
 * using a DSAP and SSAP of 0x00, and putting a 2-byte field that appeared
 * to contain at least for IFF, the Ethertype value for IFF.
 *
 * We assume that the value there is an Ethertype value, except for
 * the IEEE spanning tree protocol, which also uses a DSAP and SSAP
 * of 0x00 but has, at least in one capture, 0x0000 as the type
 * field. We handle that specially.
 *
 * IEEE 802.2 also has some packets on 802.11 with a DSAP and SSAP of 0x00,
 * but with random stuff that appears neither to be XML nor IEEE
 * spanning tree.
 */
static void
dissect_xml(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{
    proto_tree *subtree = NULL;
    proto_tree *t1;
    guint16 type;
    tvbuff_t *next_tvb;

    call_dissector(proto_xml_handle, tvb, pinfo, tree);

    if (tree) {
        t1 = proto_tree_add_item(tree, proto_xml, tvb, 0, 4, FALSE);
        subtree = proto_tree_add_subtree(t1, ett_xml, tvb, 0, 4, FALSE);

        type = tvb_get_ntohs(tvb, 0);
        next_tvb = tvb_get_subset_remaining(tvb, 2);
        if (type == 0x0000) {
            /* If the type is 0x0000, it is either a IEEE 802.2 type or a IEEE 802.11
             * type. */
            if (type == 0x0000) {
                /* If the type is 0x0000, it is either a IEEE 802.2 type or a IEEE 802.11
                 * type. */
                call_dissector(proto_xml_handle, next_tvb, pinfo, tree);
            } else {
                /* If the type is 0x0000, it is either a IEEE 802.2 type or a IEEE 802.11
                 * type. */
                call_dissector(proto_xml_handle, next_tvb, pinfo, tree);
            }
        } else {
            /* If the type is not 0x0000, it is either a IEEE 802.2 type or a IEEE 802.11
             * type. */
            call_dissector(proto_xml_handle, next_tvb, pinfo, tree);
        }
    }
}

void
proto_register_xml(void)
{
    static hf_register_info hf[] = {
        /* Registered here but handled in ethertype.c */
        { &hf_xml_type_ethertype,
          { "Type", "Xmlencaps", PF_ETHERTYPE, BASE_HEX,
            VALS(etype_vals), 0x00, NULL, HFILL } },
        { &hf_xml_type_vlan_tag,
          { "Type", "Xmlencaps", PF_ETHERTYPE, BASE_HEX,
            VALS(etype_vals), 0x00, NULL, HFILL } },
    };

    static gint *ett[] = {
        &ett_xml,
    };

    proto_xml = proto_register_protocol("XML Encapsulation", "XML", "xml");
    proto_register_field_array(proto_xml, hf, array_length(hf));
    proto_register_subtree_arrays(ett, array_length(ett));
}

void
proto_reg_handoff_xml(void)
{
    dissector_handle_t our_xml_handle;
    dissector_handle_t data_handle;
    dissector_handle_t data_handle;

    ethertype_subdissector_table = find_dissector_table("ethertype");
    our_xml_handle = create_dissector_handle(dissect_xml, proto_xml);
    dissector_add("l3.disp", 0x00, our_xml_handle);
}
```

Legal

Globals (value strings)

Dissection Routines

Registration Routines

Header Field Info



Registration

packet-arp.c

```
void
proto_register_arp(void)
{
    proto_arp = proto_register_protocol("Address Resol
    "ARP/RARP", "a

}

void
proto_reg_handoff_arp(void)
{
    dissector_handle_t arp_handle;

    arp_handle = find_dissector("arp");

    dissector_add("ethertype", ETHERTYPE_ARP, arp_hand
    dissector_add("ethertype", ETHERTYPE_REVARP, arp_h
    dissector_add("arcnet.protocol_id", ARCNET_PROTO_A
    dissector_add("arcnet.protocol_id", ARCNET_PROTO_A
    dissector_add("arcnet.protocol_id", ARCNET_PROTO_R
}
```

register.c

```
void
register_all_protocols(register_cb cb, gpointer clie
{
    {extern void proto_register_1722 (void); if(cb) (*
    {extern void proto_register_arp (void); if(cb) (*c
    {extern void proto_register_zrtp (void); if(cb) (*
}

void
register_all_protocol_handoffs(register_cb cb, gpoin
{
    {extern void proto_reg_handoff_1722 (void); if(cb)
    {extern void proto_reg_handoff_arp (void); if(cb)
    {extern void proto_reg_handoff_zrtp (void); if(cb)
}
```

tv_buff: Testy, virtual buffers

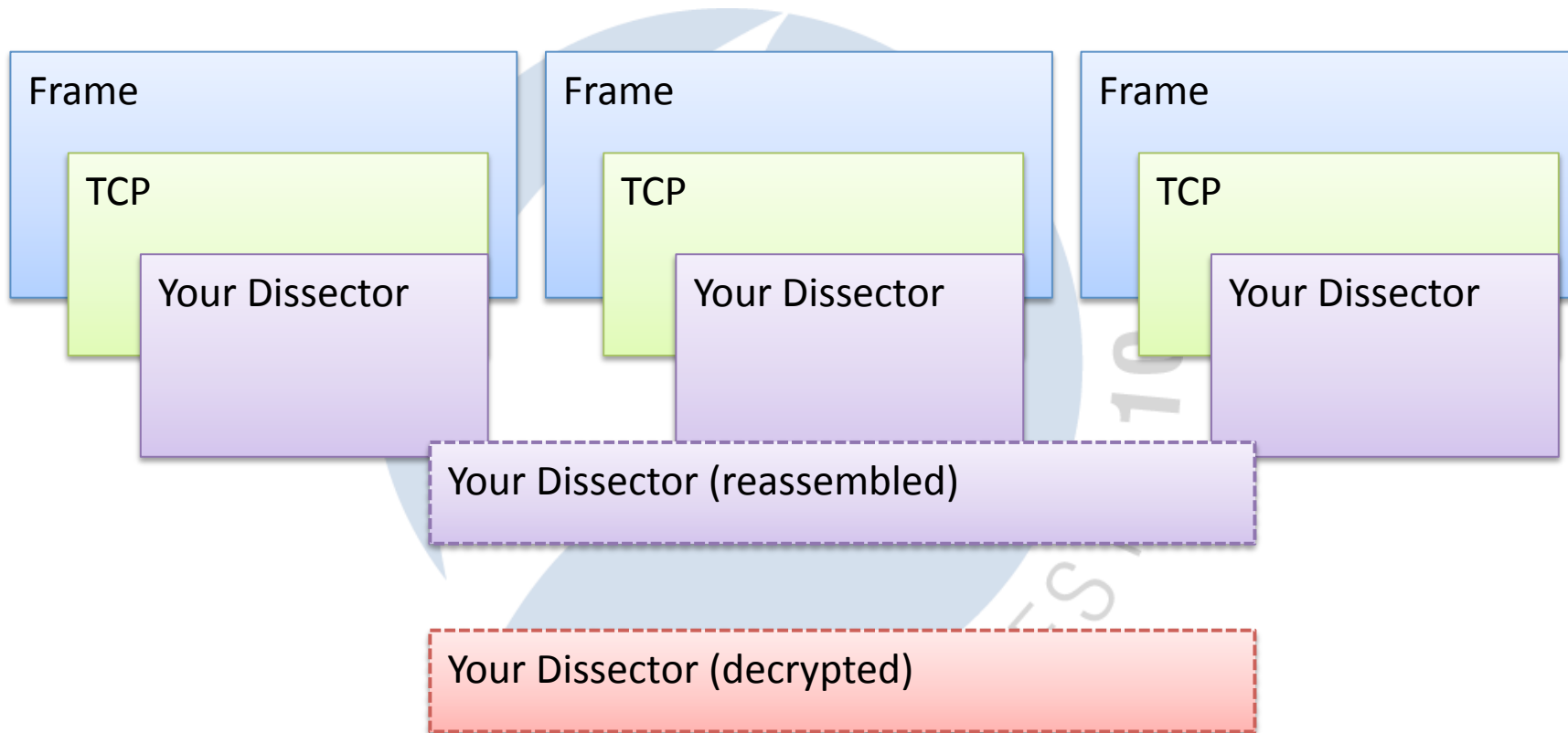
- Data buffers & extraction
- tvb_get_...
 - guint8
 - ntohs, ntohs24, ntohs1, ntohs64
 - letohs, ...
 - ipv4, ipv6
 - string, ...
- tvb_mem...

Make your own!
Impress your friends!

epan/
tvbuff.h

Chain them
together!

tv_buff Examples



packet_info & frame_data

- High and low-level metainfo
- epan/packet_info.h, epan/frame_data.h
- Frame data: Wire information
 - Length, timestamps, DLT
- Packet Info: State information
 - Addresses, ports, reassembly, protocol data

header_field_info

- Describes protocol elements
- Data type, filter name, descriptions
- Enums - Value/Range/TF Strings
- epan/proto.h
- Unique

hfinfo Examples

```
{&hf_ieee80211_ff_block_ack_timeout,  
  {"Block Ack Timeout", "wlan_mgt.fixed.batimeout",  
   FT_UINT16, BASE_HEX, NULL, 0, NULL, HFILL }},  
  
{&hf_smb_file_type,  
  { "File Type", "smb.file_type", FT_UINT16, BASE_DEC,  
   VALS(filetype_vals), 0, "Type of file", HFILL }},  
  
{&hf_ieee80211_ff_block_ack_params_amsdu_permitted,  
  {"A-MSDUs", "wlan_mgt.fixed.baparams.amsdu",  
   FT_BOOLEAN, 16, TFS (&ff_block_ack_params_amsdu_permitted_flag),  
   0x0001, "A-MSDU Permitted in QoS Data MPDUs", HFILL }},
```

proto_tree

- Detail tree (middle pane)
- Might be NULL
- epan/proto.h
- Can be hidden or “generated”
- Avoid proto_tree_add_text()

Adding Tree Items

- Names
- Data type
- Tree position
- Packet position

```
proto_item *ti;  
proto_tree *sub_tree = NULL;  
  
ti = proto_tree_add_item(tree, ...);  
sub_tree = proto_item_add_subtree(ti, ...);  
  
proto_tree_add_item(sub_tree, ...);  
proto_tree_add_uint_format(sub_tree, ...);
```

Naïve Dissection

```
typedef struct _my_pdu_header_t {  
    guint8 version;  
    guint16 type;  
    guint16 code;  
    guint16 len;  
} my_pdu_header_t;
```

```
my_pdu_header_t *hdr;
```

```
hdr = (my_pdu_header_t *) tvb_get_ptr(tvb, 0, sizeof(my_pdu_header_t));
```

```
proto_tree_add_text(tree, tvb, 0, 1, "Version: %u", hdr->ver);
```

```
proto_tree_add_text(tree, tvb, 1, 2, "Type: %u", hdr->type);
```

```
proto_tree_add_text(tree, tvb, 3, 2, "Code: %u", hdr->code);
```

```
proto_tree_add_text(tree, tvb, 5, 2, "Len: %u", hdr->len);
```

tvb_get_ptr()

Perception



Reality



Adding Raw Data

```
/* Just plain wrong */  
proto_tree_add_text(tree, tvb, 0, 50, tvb_get_ptr(tvb, 0, 50));  
  
/* Still bad */  
proto_tree_add_text(tree, tvb, 0, 50, "%s", tvb_get_ptr(tvb, 0, 50));  
  
/* Best */  
proto_tree_add_text(tree, tvb, 0, 50, "%s", tvb_format_text(tvb, 0, 50));  
proto_tree_add_item(tree, hf_my_ft_bytes_item, ...);
```

Columns

- epan/column-utils.h
- Accessed via pinfo
- Enums for each type (COL_PROTOCOL, COL_INFO)
- Don't use `check_col` any more

```
col_set_str(pinfo->cinfo, COL_PROTOCOL, "TCP");  
col_set_str(pinfo->cinfo, COL_INFO, "[TCP segment of a reassembled PDU]");
```

Memory management

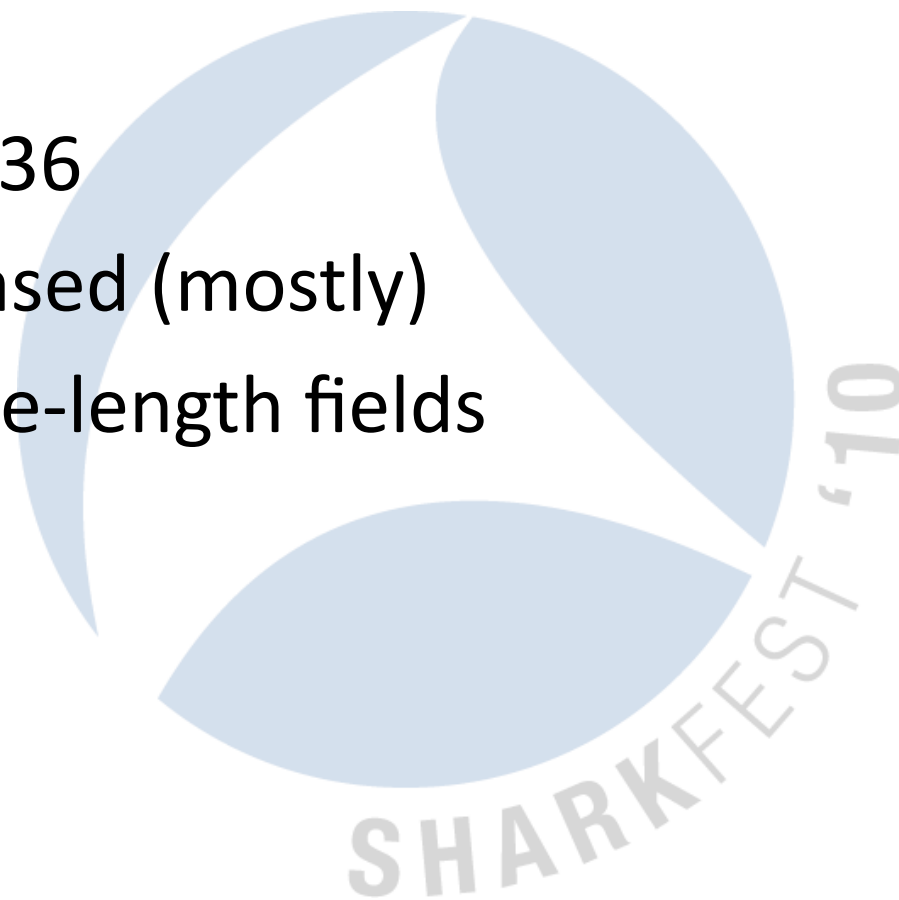
- Manual: GLib
 - `g_malloc()`, `g_free()`
- Automatic: `epan/emem.h`
 - `ep_alloc()`
 - `se_alloc()`
 - Strings (static & growable)
 - Binary trees
 - Stacks
 - Fast!

Let's Make a Dissector!

- Copy from README.developer or existing dissector
- Place in epan/dissectors (built-in)
- Add to DISSECTOR_SRC in Makefile.common
- More initial work for plugins

Gopher Protocol

- RFC 1436
- Text-based (mostly)
- Variable-length fields



Gopher Directory Entities

- Line-based
- Last line empty

Field	Length
Type	1
Item name	0 – 70
ASCII Tab	1
Selector	0 – 255
ASCII Tab	1
Host	0 – 255?
ASCII Tab	1
Port	0 – 5
ASCII CR-LF	2

Challenges

- "Simplicity is intentional"
...so what does this packet contain?
- What are our minimum lengths?

Example

Minimal Gopher Dissector

Value Strings

- Map integers to strings
- `val_to_str()`, `match_strval()`
- Also range strings, T/F strings, string strings, and bitfields

```
static const value_string auth_vals[] = {  
    {0, "Authentication Request"},  
    {1, "Authentication Response"},  
    {847, "Look! Ice cream man!"},  
    {0, NULL}  
};
```

Example

Directory Dissection

Dissector Deficiencies

- Primitive state tracking
- No reassembly
- Limited response support
- Wire data doesn't match spec (GASP!)

Strings

- GLib internals
 - `g_str*()`, `g_string_*()`;
- `ep_str*()` and `tvb_get_str*()`
- `epan/strutil.h`, `epan/to_str.h`
- `ep_strbuf*()`

ep_strbuf Example

```
emem_strbuf_t *flags_strbuf = ep_strbuf_new_label("<None>");
const gchar *fstr[] = {"FIN", "SYN", "RST", "PSH", "ACK", "URG", "ECN", "CWR"};
gboolean first_flag = TRUE;

for (i = 0; i < 8; i++) {
    bpos = 1 << i;
    if (tcph->th_flags & bpos) {
        if (first_flag) {
            ep_strbuf_truncate(flags_strbuf, 0);
        }
        ep_strbuf_append_printf(flags_strbuf, "%s%s", first_flag ? "" : ", ",
                               fstr[i]);
        first_flag = FALSE;
    }
}
```

Plugins

- Live in /plugins
- Separate DLL / shared object
- For each plugin:
 1. Load it
 2. Look for init routines
 3. Run them

Plugin or Built-in?

	Plugin	Built-in
Licensing	Proprietary?	GPLv2
Complexity	Some	Minimal
Distribution	DLL	Application

Creating a Plugin

- doc/README.plugins
- /plugins/xyzzzy
 - Dissector source
 - Makefiles
 - Boilerplate
- plugin.c wrapper auto-generated

Distributing Your Code

- Can you?
- Should you?



Contributing Your Code

1. Fuzz!
2. Run check scripts (checkAPIs.pl)
3. Generate a patch
`svn diff > ~/Desktop/banana.patch`
4. Patch + sample capture →
bugs.wireshark.org

Fuzzing Example

```
cd wireshark-gtk2
../tools/fuzz-test.sh /tmp/*.pcap

../tools/fuzz-test.sh: line 56: ulimit: cpu time: cannot modify limit: Invalid argument
Running ./tshark with args: -nVxr (forever)

Starting pass 1:
  c:\cygwin\tmp\buildbot.test.pcap: OK
Starting pass 2:
  c:\cygwin\tmp\buildbot.test.pcap: OK
...
...
```

Common Mistakes

- Not using `proto_tree_add_item`
- Extracting packet data yourself
- Assuming packets are processed in order
- Setting columns and offsets inside `if (tree)`
- `tvb_reported_length_remaining()`
- Creating a bunch of tvbuffs
- `abort` or `g_assert` in your dissector

Further Information

- <http://anonsvn.wireshark.org/wireshark/trunk>
- <http://www.wireshark.org/develop.html>
- Wireshark Developer's Guide
- doc/README.developer
- wireshark-dev@wireshark.org
- Next session