Spicy
A Framework For Dissecting All Your Data
Protocol Dissection - tcpdump

# tcpdump -n -r tftp_rrq.pcap
12:24:11.972052  IP 192.168.0.253.50618 > 192.168.0.10.tftp: 20 RRQ "rfc1350.txt" octet
12:24:12.077243  IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
12:24:12.081790  IP 192.168.0.253.50618 > 192.168.0.10.monp: UDP, length 4
12:24:12.086300  IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
12:24:12.088961  IP 192.168.0.253.50618 > 192.168.0.10.monp: UDP, length 4
12:24:12.088995  IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
12:24:12.091646  IP 192.168.0.253.50618 > 192.168.0.10.monp: UDP, length 4
12:24:12.091675  IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
12:24:12.094383  IP 192.168.0.253.50618 > 192.168.0.10.monp: UDP, length 4
Protocol Dissection - tcpdump

```
# tcpdump -n -r tftp_rrq.pcap
12:24:11.972852 IP 192.168.0.253.50618 > 192.168.0.10.tftp: 20 RRQ "rfc1350.txt" octet
12:24:12.077243 IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
12:24:12.081790 IP 192.168.0.253.50618 > 192.168.0.10.monp: UDP, length 4
12:24:12.086300 IP 192.168.0.10.monp > 192.168.0.253.50618: UDP, length 516
```

RFC 1350

TFTP Revision 2                    July 1992

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5-1: RRQ/WRQ packet
Protocol Dissection - tcpdump

```c
if (length < 2)
    goto trunc;
opcode = GET_BE_U_2(bp);
cp = tok2str(op2str, "tftp-#\%u", opcode);
ND_PRINT("", %s", cp);
/* Bail if bogus opcode */
if (*cp == 't')
    return;
bp += 2;
length -= 2;
switch (opcode) {
case RRQ:
    if (length == 0)
        goto trunc;
    ND_PRINT(" ");
    /* Print filename */
    ND_PRINT("\n");
    ui = nd_printztn(ndo, bp, length, 
                    ndo->ndo_snapend);
    ND_PRINT("\n");
    if (ui == 0)
        goto trunc;
    bp += ui;
    length -= ui;
    /* Print the mode - RRQ and WRQ only */
    if (length == 0)
        goto trunc; /* no mode */
    ND_PRINT(" ");
    ui = nd_printztn(ndo, bp, length, 
                    ndo->ndo_snapend);
    if (ui == 0)
        goto trunc;
    bp += ui;
    length -= ui;
```
if (length < 2)
    goto trunc;
opcode = GET_BE_U_2(bp);
cp = tok2str(op2str, "tftp-"#u", opcode);
ND_PRINT("", "", cp);
/* Bail if bogus opcode */
if (*cp == 't')
    return;
bp += 2;
length -= 2;
switch (opcode) {
case RRQ:
    if (length == 0)
        goto trunc;
    if (ui == 0)
        goto trunc;
    bp += ui;
    length -= ui;
    ND_PRINT("\"\"");
    ui = nd_printztn(ndo, bp, length,
        ndo->ndo_snapend);
    ND_PRINT("\"\"");
    if (ui == 0)
        goto trunc;
    if (length == 0)
        goto trunc; /* no mode */
    /* Print the mode - RRQ and WRQ only */
    bp += ui;
    length -= ui;
    ND_PRINT(" ");
    ui = nd_printztn(ndo, bp, length,
        ndo->ndo_snapend);
    if (ui == 0)
        goto trunc;
    bp += ui;
    length -= ui;
Protocol Dissection - Wireshark

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td></td>
<td>62 Read Request, File: rfc1350.txt, Transfer type: octet</td>
</tr>
<tr>
<td>2</td>
<td>0.184391</td>
<td>192.168.0.10</td>
<td>192.168.0.253</td>
<td>TFTP</td>
<td>558</td>
<td>Data Packet, Block: 1</td>
</tr>
<tr>
<td>3</td>
<td>0.188938</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td>60</td>
<td>Acknowledgement, Block: 1</td>
</tr>
<tr>
<td>4</td>
<td>0.113448</td>
<td>192.168.0.10</td>
<td>192.168.0.253</td>
<td>TFTP</td>
<td>558</td>
<td>Data Packet, Block: 2</td>
</tr>
<tr>
<td>5</td>
<td>0.116109</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td>60</td>
<td>Acknowledgement, Block: 2</td>
</tr>
<tr>
<td>6</td>
<td>0.116143</td>
<td>192.168.0.10</td>
<td>192.168.0.253</td>
<td>TFTP</td>
<td>558</td>
<td>Data Packet, Block: 3</td>
</tr>
<tr>
<td>7</td>
<td>0.118794</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td>60</td>
<td>Acknowledgement, Block: 3</td>
</tr>
<tr>
<td>8</td>
<td>0.118823</td>
<td>192.168.0.10</td>
<td>192.168.0.253</td>
<td>TFTP</td>
<td>558</td>
<td>Data Packet, Block: 4</td>
</tr>
<tr>
<td>9</td>
<td>0.121531</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td>60</td>
<td>Acknowledgement, Block: 4</td>
</tr>
<tr>
<td>10</td>
<td>0.121564</td>
<td>192.168.0.10</td>
<td>192.168.0.253</td>
<td>TFTP</td>
<td>558</td>
<td>Data Packet, Block: 5</td>
</tr>
<tr>
<td>11</td>
<td>0.124161</td>
<td>192.168.0.253</td>
<td>192.168.0.10</td>
<td>TFTP</td>
<td>60</td>
<td>Acknowledgement, Block: 5</td>
</tr>
</tbody>
</table>

- Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
- Ethernet II, Src: Cisco_18:9a:40 (00:0b:be:18:9a:40), Dst: AbitComp_d7:8b:43 (00:50:8d:d7:8b:43)
- User Datagram Protocol, Src Port: 50618, Dst Port: 69
- Trivial File Transfer Protocol
  Opcode: Read Request (1)
  Source File: rfc1350.txt
  Type: octet
/* Opcode */

opcode = tvb_get_ntohs(tvb, offset);
proto_tree_add_uint(tftp_tree, hf_tftp_opcode, tvb, offset, 2, opcode);
col_add_str(pinfo->cinfo, COL_INFO, val_to_str(opcode, tftp_opcode_vals, "Unknown (0x%04x)");
offset += 2;

/* read and write requests contain file names
   for other messages, we add the filenames from the conversation */
if (opcode!=TFTP_RRQ && opcode!=TFTP_WRQ) {
    if (tftp_info->source_file) {
        filename = tftp_info->source_file;
    } else if (tftp_info->destination_file) {
        filename = tftp_info->destination_file;
    }
    ti = proto_tree_add_string(tftp_tree, hf_tftp_destination_file, tvb, 0, 0, filename);
    proto_item_set_generated(ti);
}

switch (opcode) {
    case TFTP_RRQ:
        i1 = tvb_strsize(tvb, offset);
        proto_tree_add_item_ret_string(tftp_tree, hf_tftp_source_file, tvb, offset, i1, ENC_ASCII|ENC_NA, wmem_file_scope(), &tftp_info->source_file);
        /* we either have a source file name (for read requests) or a
           destination file name (for write requests)
           when we set one of the names, we clear the other */
        tftp_info->destination_file = NULL;
        col_append_fstr(pinfo->cinfo, COL_INFO, ", File: %s",
            tvb_format_stringzpad(tvb, offset, i1));
        offset += i1;
        i1 = tvb_strsize(tvb, offset);
        proto_tree_add_item(tftp_tree, hf_tftp_transfer_type, tvb, offset, i1, ENC_ASCII|ENC_NA);
        col_append_fstr(pinfo->cinfo, COL_INFO, ", Transfer type: %s",
            tvb_format_stringzpad(tvb, offset, i1));
        offset += i1;
        tftp_dissect_options(tvb, pinfo, offset, tftp_tree, opcode, tftp_info);
        break;
/* Opcode */
opcode = tvb_get_ntohs(tvb, offset);
proto_tree_add_uint(tftp_tree, hf_tftp_opcode, tvb, offset, 2, opcode);
col_add_str(pinfo->cinfo, COL_INFO, val_to_str(opcode, tftp_opcode_vals, "Unknown (0x%04x")));
offset += 2;

/* read and write requests contain file names
for other messages, we add the filenames from the conversation */
if (opcode!=TFTP_RRQ && opcode!=TFTP_WRQ) {
    if (tftp_info->source_file) {
        filename = tftp_info->source_file;
    } else if (tftp_info->destination_file) {
        filename = tftp_info->destination_file;
    }
    ti = proto_tree_add_string(tftp_tree, hf_tftp_destination_file, tvb, 0, 0, filename);
    proto_item_set_generated(ti);
}

switch (opcode) {
    case TFTP_RRQ:
        i1 = tvb_strsize(tvb, offset);
        proto_tree_add_item_ret_string(tftp_tree, hf_tftp_source_file, tvb, offset, i1, ENC_ASCII|ENC_NA, wmem_file_scope(), &tftp_info->source_file);
        /* we either have a source file name (for read requests) or a 
         * destination file name (for write requests)
         * when we set one of the names, we clear the other */
        tftp_info->destination_file = NULL;
        col_append_fstr(pinfo->cinfo, COL_INFO, ", File: %s", tvb_format_stringzpad(tvb, offset, i1));
        offset += i1;
        i1 = tvb_strsize(tvb, offset);
        proto_tree_add_item(tftp_tree, hf_tftp_transfer_type, tvb, offset, i1, ENC_ASCII|ENC_NA);
        col_append_fstr(pinfo->cinfo, COL_INFO, ", Transfer type: %s", tvb_format_stringzpad(tvb, offset, i1));
        offset += i1;
        tftp_dissect_options(tvb, pinfo, offset, tftp_tree, opcode, tftp_info);
        break;
Protocol Dissection - Zeek (FTP)
# Protocol Dissection - Zeek (FTP)

```
# zeek -r ftp/ipv4.trace
# cat ftp.log

<table>
<thead>
<tr>
<th>ts</th>
<th>uid</th>
<th>id.orig_h</th>
<th>id.orig_p</th>
<th>id.resp_h</th>
<th>id.resp_p</th>
<th>user</th>
<th>command</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1329843175.680248</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>PASV</td>
<td></td>
</tr>
<tr>
<td>1329843179.815947</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>PASV</td>
<td></td>
</tr>
<tr>
<td>1329843179.926563</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>RETR</td>
<td>ftp://199.233.217.249/.robots.txt</td>
</tr>
<tr>
<td>1329843194.040188</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>PORT</td>
<td>141,142,220,235,131,46</td>
</tr>
<tr>
<td>1329843197.672179</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>PORT</td>
<td>141,142,220,235,147,203</td>
</tr>
<tr>
<td>1329843197.727769</td>
<td>CkVhAVG1DHFjwGm9</td>
<td>141.142.220.235</td>
<td>50032</td>
<td>199.233.217.249</td>
<td>21</td>
<td>anonymous</td>
<td>RETR</td>
<td>ftp://199.233.217.249/.robots.txt</td>
</tr>
</tbody>
</table>
```
Protocol Dissection - Zeek (FTP)

```cpp
EventHandlerPtr f;

if (orig)
{
    int cmd_len;
    const char* cmd;
    StringVal* cmd_str;

    line = util::skip_whitespace(line, end_of_line);
    util::get_word(end_of_line - line, line, cmd_len, cmd);
    line = util::skip_whitespace(line + cmd_len, end_of_line);

    if (cmd_len == 0)
    {
        // Weird("FTP command missing", end_of_line - orig_line, orig_line);
        cmd_str = new StringVal("<missing>");
    }
    else
    {
        cmd_str = (new StringVal(cmd_len, cmd))->ToUpper();
    }

    vl = {
        ConnVal(),
        IntrusivePtr{AdoptRef{}, cmd_str},
        make_intrusive<StringVal>(end_of_line - line, line),
    };

    f = ftp_request;
    ProtocolConfirmation();

    if (strncmp((const char*)cmd_str->Bytes(),
                "AUTH", cmd_len) == 0)
        auth_requested = std::string(line, end_of_line - line);

    if (detail::rule_matcher)
        Conn() -> Match(zeek::detail::Rule::FTP, (const u_char*)cmd,
                        end_of_line - cmd, true, true, true, true);

    else { ... }
}

EnqueueConnEvent(f, std::move(vl));
return ForwardStream(length, data, orig);
```
EventHandlerPtr f;

if ( orig ) {
    int cmd_len;
    const char* cmd;
    StringVal* cmd_str;

    line = util::skip_whitespace(line, end_of_line);
    util::get_word(end_of_line - line, line, cmd_len, cmd);
    line = util::skip_whitespace(line + cmd_len, end_of_line);

    if ( cmd_len == 0 ) {
        // Weird("FTP command missing", end_of_line - orig_line, orig_line);
        cmd_str = new StringVal("<missing>");
    } else
        cmd_str = (new StringVal(cmd_len, cmd))->ToUpper();

    vl = {
        ConnVal(),
        IntrusivePtr{AdoptRef{}, cmd_str},
        make_intrusive<StringVal>(end_of_line - line, line),
    };

    f = ftp_request;
    ProtocolConfirmation();

    if ( strncmp((const char*) cmd_str->Bytes(),
                  "AUTH",
                  cmd_len) == 0 )
        auth_requested = std::string(line, end_of_line - line);

    if ( detail::rule_matcher )
        Conn() -> Match(zeek::detail::Rule::FTP, (const u_char*) cmd,
                        end_of_line - cmd, true, true, true, true);
    else { ... }
}

EnqueueConnEvent(f, std::move(vl));
return ForwardStream(length, data, orig);
Dissecting Protocols Is Hard

(... and no fun)
Dissecting Protocols Is Hard

Must be robust
Lots of “crud” in real-world networks
Cannot trust input
Dissecting Protocols Is Hard (... and no fun)

**Must be robust**
- Lots of “crud” in real-world networks
- Cannot trust input

**Must be efficient**
- 100,000s of concurrent connections
- Incremental, stateful processing with low latency & memory usage
Dissecting Protocols Is Hard
(... and no fun)

Must be robust
Lots of “crud” in real-world networks
Cannot trust input

Must be efficient
100,000s of concurrent connections
Incremental, stateful processing with low latency & memory usage

Must be complete
Leaving out parts of the protocol means blind spots
Protocols can be really complex (looking at you, SMB …)
Dissector vulnerabilities are endemic
Dissector vulnerabilities are endemic

---

 CVE Details
The ultimate security vulnerability datasource

---

**Wireshark » Wireshark : Vulnerability Statistics**

**Vulnerability Trends Over Time**

<table>
<thead>
<tr>
<th>Year</th>
<th># of Vulnerabilities</th>
<th>DoS</th>
<th>Code Execution</th>
<th>Overflow</th>
<th>Memory Corruption</th>
<th>Sql Injection</th>
<th>XSS</th>
<th>Directory Traversal</th>
<th>Http Response Splitting</th>
<th>Bypass something</th>
<th>Gain Information</th>
<th>Gain Privileges</th>
<th>CSRF</th>
<th>File Inclusion</th>
<th># of exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>25</td>
<td>25</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>23</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2009</td>
<td>25</td>
<td>23</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>15</td>
<td>11</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2011</td>
<td>31</td>
<td>29</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2012</td>
<td>46</td>
<td>42</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2013</td>
<td>82</td>
<td>82</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2014</td>
<td>29</td>
<td>29</td>
<td>2</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2015</td>
<td>33</td>
<td>33</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2016</td>
<td>95</td>
<td>89</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2017</td>
<td>58</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>79</td>
<td>1</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>21</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>576</td>
<td>400</td>
<td>130</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>% Of All</td>
<td>69.4</td>
<td>3.8</td>
<td>22.6</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dissector vulnerabilities are endemic

**zeek** System for detecting network intruders in real-time

3.0.11 **security**  
Maintainer: lcr4s@FreeBSD.org

---

Commit History - (may be incomplete; see SVNWeb link above for full details)

<table>
<thead>
<tr>
<th>Date</th>
<th>By</th>
<th>Description</th>
</tr>
</thead>
</table>
| 07 Oct 2020 21:29:54  | lcr4s  | security/zeek: Update to 3.0.11 to fix memory leaks and potential DOS:  
|              |        | [https://github.com/zeek/zeek/releases/tag/v3.0.11](https://github.com/zeek/zeek/releases/tag/v3.0.11) |  
|              |        | - A memory leak in multipart MIME code has potential for remote exploitation and cause for Denial of Service via resource exhaustion. |  
|              |        | Other fixes:  
|              |        | - Fix incorrect RSTOS0 conn_state determinations |  
|              |        | Reported by: Jon Siwek  
|              |        | MPH: 2020Q4  
|              |        | Security: 7694f6d0-9056-4c27-89e1-1758a59a21e8 |  
| 10 Sep 2020 00:15:49  | lcr4s  | security/zeek: Update to 3.0.10 to fix memory leaks and potential DOS:  
|              |        | [https://github.com/zeek/zeek/releases/tag/v3.0.10](https://github.com/zeek/zeek/releases/tag/v3.0.10) |  
|              |        | - Fix memory leak caused by re-entering AVIYA parsing |  
|              |        | - Fix memory leak caused by re-entering GTPv1 parsing |  
|              |        | Other fixes:  
|              |        | - Fix Input Framework 'change' events for 'set' destinations |  
|              |        | - Fix reported body-length of HTTP messages w/ sub-entities |  
|              |        | Reported by: Jon Siwek  
|              |        | MPH: 2020Q3  
|              |        | Security: 2c92f9d3-896c-a5a-a0d8-52acceee9182d |  
| 28 Jul 2020 01:09:30  | lcr4s  | security/zeek: Update to 3.0.8 and address various vulnerabilities:  
|              |        | [https://github.com/zeek/zeek/releases/tag/v3.0.8](https://github.com/zeek/zeek/releases/tag/v3.0.8) |  
|              |        | - Fix potential DNS analyzer stack overflow |  
|              |        | - Fix potential MethiosSSH analyzer stack overflow |  
|              |        | Other fixes:  
|              |        | - Fix DHCP Client ID Option misformat for Hardware Type 0 |  
|              |        | - Fix/allow copying/cloning of opaque of Broker1:Store |  
|              |        | - Fix ConnPolling memory over-use |  

---

8 of 11 3.0.x updates fixed dissector issues!
Can we make it easier to develop robust dissectors?

(... and maybe even share them??)
Projects have developed a variety of approaches to support writing dissectors.
Language Support

Projects have developed a variety of approaches to support writing dissectors.

- Wireshark
- Suricata
- Lua
- R
Meanwhile, in another domain ...

There are powerful tools for generating parsers from declarative specifications.
Meanwhile, in another domain …

There are powerful tools for generating parsers from declarative specifications.

```
exp: NUM          { $$ = $1;    }
| exp ' + ' exp   { $$ = $1 + $2;  }
| exp ' - ' exp   { $$ = $1 - $2;  }
| exp ' * ' exp   { $$ = $1 * $2;  }
| exp ' / ' exp   { $$ = $1 / $2;  }
```

zeek  → Yacc  ↓  yyparse()  →  Host Application
Meanwhile, in another domain …

There are powerful tools for generating parsers from declarative specifications.

These parsers aren’t suitable for Zeek, unfortunately.

No support for concurrent, incremental processing
No support for domain-specific idioms
A yacc for Protocols (2006)

binpac: A yacc for Writing Application Protocol Parsers

Ruoming Pang
Google, Inc.

Vern Paxson
International Computer Science Institute

Robin Sommer
International Computer Science Institute

Larry Peterson
Princeton University
binpac:: A yacc for Writing Application Protocol Parsers

Ruoming Pang  
Google, Inc.

Vern Paxson  
International Computer Science Institute

Robin Sommer  
International Computer Science Institute

Larry Peterson  
Princeton University

type ClientHello(rec: HandshakeRecord) = record {
  client_version: uint16;
  gmt_unix_time : uint32;
  random_bytes  : bytestring &length = 28;
  session_len   : uint8;
  session_id    : uint8[session_len];
  dtls_cookie   : case client_version of {
    DTLSv10, DTLSv12 -> cookie : ClientHelloCookie(rec);
    default          -> nothing: bytestring &length=0;
  };
} [...]

TLS v3 Client Hello  (Source: Zeek's TLS analyzer)
A yacc for Protocols (2006)

binpac: A yacc for Writing Application Protocol Parsers
Ruoming Pang
Google, Inc.
Robin Sommer
International Computer Science Institute
Vern Paxson
International Computer Science Institute
Larry Peterson
Princeton University

However, BinPAC solves the challenge only partially.

Remains limited to syntax, cannot express logic
Still needs custom C++ for logic & integration
Remains limited to application protocols & connection structure
Lacks support for higher-level idioms
New Zeek Project: Spicy Parser Generator
New Zeek Project: Spicy Parser Generator

High-level scripting language for writing robust & efficient protocol dissectors

https://github.com/zeek/spicy
New Zeek Project: Spicy Parser Generator

High-level scripting language for writing robust & efficient protocol dissectors

Declarative syntax, with hooks for imperative logic
Facilitates writing robust, stateful dissectors without a single line of C/C++

https://github.com/zeek/spicy
New Zeek Project: Spicy Parser Generator

High-level scripting language for writing robust & efficient protocol dissectors

Declarative syntax, with hooks for imperative logic
  Facilitates writing robust, stateful dissectors without a single line of C/C++

Built-in support for domain-specific idioms
  Types, byte ordering, reassembly, protocol detection (v2), error recovery (v2)

https://github.com/zeek/spicy
New Zeek Project: Spicy Parser Generator

High-level scripting language for writing robust & efficient protocol dissectors

Declarative syntax, with hooks for imperative logic
  Facilitates writing robust, stateful dissectors without a single line of C/C++

Built-in support for domain-specific idioms
  Types, byte ordering, reassembly, protocol detection (v2), error recovery (v2)

Facilitates composition and reuse
  Dissectors can imported, extended, and layered

https://github.com/zeek/spicy
New Zeek Project: Spicy Parser Generator

High-level scripting language for writing robust & efficient protocol dissectors

Declarative syntax, with hooks for imperative logic
Facilitates writing robust, stateful dissectors without a single line of C/C++

Built-in support for domain-specific idioms
Types, byte ordering, reassembly, protocol detection (v2), error recovery (v2)

Facilitates composition and reuse
Dissectors can imported, extended, and layered

Agnostic of host application
Generates safe C++ dissector code with simple API for easy integration
Employs application-specific glue generators for integration logic

https://github.com/zeek/spicy
### Figure 5-1: RRQ/WRQ packet

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Filename</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>string</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

RFC 1350 TFTP Revision 2 July 1992
RFC 1350  TFTP Revision 2  July 1992

2 bytes   string   1 byte   string   1 byte
------------------------------------------------
| Opcode  | Filename | 0      | Mode    | 0      |
------------------------------------------------

Figure 5-1: RRQ/WRQ packet

module TFTP;

class ReadRequest = unit {
    opcode: uint16;
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";

    on %done { print self; }
};

tftp-rrq.spicy
Protocol Dissection - Spicy

RFC 1350  TFTP Revision 2  July 1992

```
2 bytes     string     1 byte    string     1 byte
------------------------------------------------
| Opcode | Filename | 0 | Mode | 0 |
------------------------------------------------
```

Figure 5-1: RRQ/WRQ packet

module TFTP;

public type ReadRequest = unit {
  opcode:   uint16;
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";

  on %done { print self; }
};

tftp-rrq.spicy

Time for a demo!
Protocol Dissection - Spicy

RFC 1350  TFTP Revision 2  July 1992

2 bytes     string    1 byte     string    1 byte
------------------------------------------------
| Opcode | Filename | 0 | Mode | 0 |
------------------------------------------------

Figure 5-1: RRQ/WRQ packet

module TFTP;

public type ReadRequest = unit {
    opcode:   uint16;
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";

    on %done { print self; }
};
tftp-rrq.spicy

# cat rrq.dat | spicy-driver tftp-rrq.spicy
Protocol Dissection - Spicy

RFC 1350  TFTP Revision 2  July 1992

2 bytes     string    1 byte     string   1 byte
------------------------------------------------
| Opcode | Filename | 0 | Mode   | 0 |
------------------------------------------------

Figure 5-1: RRQ/WRQ packet

module TFTP;

public type ReadRequest = unit {
    opcode:  uint16;
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";

    on %done { print self; }
};

tftp-rrq.spicy

# cat rrq.dat | spicy-driver tftp-rrq.spicy
[$opcode=1, $filename=b"rfc1350.txt", $mode=b"octet"]
Making it more real
module TFTP;

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

public type Packet = unit {
    opcode: uint16 &convert=Opcode($$);

    switch ( self.opcode ) {
        Opcode::RRQ -> rrq: ReadRequest;
    };

    on %done { print self; }
};

type ReadRequest = unit {
    filename: bytes &until=b"\x00";
    mode: bytes &until=b"\x00";
};
Making it more real

```spicy
module TFTP;

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

public type Packet = unit {
    opcode: uint16 &convert=Opcode($$);

    switch ( self.opcode ) {
        Opcode::RRQ -> rrq: ReadRequest;
    };

    on %done { print self; }
};

type ReadRequest = unit {
    filename: bytes &until=b"\x00";
    mode: bytes &until=b"\x00";
};
```

# cat rrq.dat | spicy-driver tftp-switch.spicy
Making it more real

module TFTP;

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

public type Packet = unit {
    opcode: uint16 &convert=Opcode($$);

    switch ( self.opcode ) {
        Opcode::RRQ -> rrq: ReadRequest;
    };

    on %done { print self; }
};

type ReadRequest = unit {
    filename: bytes &until=b"\x00";
    mode: bytes &until=b"\x00";
};

cat rrq.dat | spicy-driver tftp-switch.spicy

[$opcode=Opcode::RRQ, $rrq=[$filename=b"rfc1350.txt", $mode=b"octet"]]
A complete RFC 1350 Dissector
A complete RFC 1350 Dissector

module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
                        ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:   uint16;
    data:  bytes &eod;
};

type Acknowledgement = unit {
    num:   uint16;
};

type Error = unit {
    code:  uint16;
    msg:   bytes &until=b"\x00";
};
A complete RFC 1350 Dissector

```rust
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
                     ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num: uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

# cat rrq.dat | spicy-dump tftp.spicy
TFTP::Packet {
    op: RRQ
    rrq: TFTP::Request {
        filename: rfc1350.txt
        mode: octet
    }
}
```

35 LOC
Zeek Integration

Spicy Plugin

Zeek
Zeek Integration

Spicy Grammar

Compilation
(JIT or AOT)

*.spicy

Spicy Plugin

Zeek
Zeek Integration

- Spicy Grammar *.spicy
- Event Definitions *.evt

Compilation (JIT or AOT) → Spicy Plugin → Zeek
Zeek Integration

Spicy Grammar
*.spicy

Event Definitions
*.evt

Compilation
(JIT or AOT)

Spicy Plugin

Zeek

Zeek Scripts
*.zeek
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  }
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
                         ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch (self.op) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num: uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
parse with TFTP::Packet,
port 69/udp;

on TFTP::Request if (is_read)
    -> event tftp::read_request($conn, self.filename, self.mode);
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
parse with TFTP::Packet,
port 69/udp;

on TFTP::Request if ( is_read )
-> event tftp::read_request($conn, self.filename, self.mode);

# spicyz -o tftp.hlto tftp.spicy tftp.evt
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
  parse with TFTP::Packet,
  port 69/udp;

on TFTP::Request if ( is_read )
  -> event tftp::read_request($conn, self.filename, self.mode);

# spicy -o tftp.hlto tftp.spicy tftp.evt

event tftp::read_request(c: connection, fname: string, mode: string) {
  print "[In Zeek] TFTP read request", c$id, fname, mode;
}

tftp.spicy
Zeek Integration

module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
parse with TFTP::Packet,
port 69/udp;
on TFTP::Request if ( is_read )
    -> event tftp::read_request($conn, self.filename, self.mode);

# spicyz -o tftp.hlto tftp.spicy tftp.evt

event tftp::read_request(c: connection, fname: string, mode: string) {
    print "[In Zeek] TFTP read request", c$id, fname, mode;
}

# zeek -r tftp_rrq.pcap tftp.hlto tftp.zeek
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
parse with TFTP::Packet,
port 69/udp;
on TFTP::Request if ( is_read )
    -> event tftp::read_request($conn, self.filename, self.mode);

# spicy -o tftp.hlto tftp.spicy tftp.evt

event tftp::read_request(c: connection, fname: string, mode: string) {
    print “[In Zeek] TFTP read request”, c$id, fname, mode;
}

# zeek -r tftp_rrq.pcap tftp.hlto tftp.zeek
[In Zeek] TFTP read request, [orig_h=192.168.0.253, orig_p=50618/udp, resp_h=192.168.0.10, resp_p=69/udp], rfc1350.txt, octet
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error:  Error;
  }
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};

protocol analyzer spicy::TFTP over UDP:
parse with TFTP::Packet,
port 69/udp;

on TFTP::Request if ( is_read )
  -> event tftp::read_request($conn, self.filename, self.mode);

# spicy -o tftp.hlto tftp.spicy tftp.evt

event tftp::read_request(c: connection, fname: string, mode: string) {
  print "[In Zeek] TFTP read request", c$id, fname, mode;
}

# zeek -r tftp_rrq.pcap tftp.hlto tftp.zeek
[In Zeek] TFTP read request, [orig_h=192.168.0.253, orig_p=50618/udp, resp_h=192.168.0.10, resp_p=69/udp], rfc1350.txt, octet

# zkg install spicy-tftp
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
    ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};
module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    }
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};
Module TFTP;

Public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch (self.op) {
        Opcode::RRQ  -> rrq:  Request(True);
        Opcode::WRQ  -> wrq:  Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK  -> ack:  Acknowledgement;
        Opcode::ERROR  -> error:  Error;
    };
};

Type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
                 ACK = 4, ERROR = 5 };

Type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

Type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

Type Acknowledgement = unit {
    num: uint16;
};

Type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

# make-wireshark-plugin \
tftp.spicy \
--parser TFTP::Packet \
--plugin_version 0.0.1 \
--plugin_want_major 3 --plugin_want_minor=3 \
--wireshark_include_dir=/usr/local/include/wireshark \
--wireshark_library_dir=/usr/local/lib \
--output spicy.so
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, 
  ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};
Poc Wireshark Integration, Take 2

module TFTP;

public type Packet = unit {
    op: uint16 &convert=Opcode($$);
    switch ( self.op ) {
        Opcode::RRQ   -> rrq:   Request(True);
        Opcode::WRQ   -> wrq:   Request(False);
        Opcode::DATA  -> data:  Data;
        Opcode::ACK   -> ack:   Acknowledgement;
        Opcode::ERROR -> error: Error;
    }
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3,
                        ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
    filename: bytes &until=b"\x00";
    mode:     bytes &until=b"\x00";
};

type Data = unit {
    num:  uint16;
    data: bytes &eod;
};

type Acknowledgement = unit {
    num: uint16;
};

type Error = unit {
    code: uint16;
    msg:  bytes &until=b"\x00";
};

on TFTP::Packet::%print {
    switch ( self.op ) {
        case TFTP::Opcode::RRQ:   return "%s" % self.rrq;
        case TFTP::Opcode::WRQ:   return "%s" % self.wrq;
        case TFTP::Opcode::DATA:  return "%s" % self.data;
        case TFTP::Opcode::ACK:   return "%s" % self.ack;
        case TFTP::Opcode::ERROR: return "%s" % self.error;
        default:                  return "<unknown message>";
    }
}

on TFTP::Request::%print {
    return "%s Request, File: %s, Transfer type: %s"
                  % ((is_read ? "Read" : "Write"), self.filename, self.mode);
}

on TFTP::Data::%print {
    return "Data Packet, Block: %u" % self.num;
}

on TFTP::Acknowledgement::%print {
    return "Acknowledgement, Block: %u" % self.num;
}

on TFTP::Error::%print {
    return "Error, Code: %u" % self.code;
}
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  };
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, 
                        ACK = 4, ERROR = 5 };


type Request = unit(is_read: bool) {
  filename: bytes &until=b"\x00";
  mode:     bytes &until=b"\x00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};


type Acknowledgement = unit {
  num: uint16;
};


type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\x00";
};

on TFTP::Packet::%print {
  switch ( self.op ) {
    case TFTP::Opcode::RRQ:   return "%s" % self.rrq;
    case TFTP::Opcode::WRQ:   return "%s" % self.wrq;
    case TFTP::Opcode::DATA:  return "%s" % self.data;
    case TFTP::Opcode::ACK:   return "%s" % self.ack;
    case TFTP::Opcode::ERROR: return "%s" % self.error;
    default:                  return "<unknown message>";
  }
};

on TFTP::Request::%print {
  return "%s Request, File: %s, Transfer type: %s"
    % ((is_read ? "Read" : "Write"), self.filename, self.mode);
}

on TFTP::Data::%print {
  return "Data Packet, Block: %u" % self.num;
}

on TFTP::Acknowledgement::%print {
  return "Acknowledgement, Block: %u" % self.num;
}

on TFTP::Error::%print {
  return "Error, Code: %u" % self.code;
}

# make-wireshark-plugin \
  tftp-wireshark-all.spicy \ 
  --parser TFTP::Packet \ 
  --plugin_version 0.0.1 \ 
  --plugin_want_major 3 --plugin_want_minor=3 \ 
  --wireshark_include_dir=/usr/local/include/wireshark \ 
  --wireshark_library_dir=/usr/local/lib \ 
  --output spicy.so

# cp spicy.so /usr/local/lib/wireshark/plugins/3-3/epan
module TFTP;

public type Packet = unit {
  op: uint16 &convert=Opcode($$);
  switch ( self.op ) {
    Opcode::RRQ   -> rrq:   Request(True);
    Opcode::WRQ   -> wrq:   Request(False);
    Opcode::DATA  -> data:  Data;
    Opcode::ACK   -> ack:   Acknowledgement;
    Opcode::ERROR -> error: Error;
  }
};

type Opcode = enum { RRQ = 1, WRQ = 2, DATA = 3, ACK = 4, ERROR = 5 };

type Request = unit(is_read: bool) {
  filename: bytes &until=b"\00";
  mode:     bytes &until=b"\00";
};

type Data = unit {
  num:  uint16;
  data: bytes &eod;
};

type Acknowledgement = unit {
  num: uint16;
};

type Error = unit {
  code: uint16;
  msg:  bytes &until=b"\00";
};

# make-wireshark-plugin \
tftp-wireshark-all.spicy \
--parser TFTP::Packet \
--plugin_version 0.0.1 \
--plugin.want_major 3 --plugin.want_minor=3 \
--wireshark.include_dir=/usr/local/include/wireshark \
--wireshark.library_dir=/usr/local/lib \
--output spicy.so
Future Spicy Extensions
Future Spicy Extensions

Additional domain support
  Eg., built-in ASN.1 support
Future Spicy Extensions

Additional domain support
  Eg., built-in ASN.1 support

Error handling & recovery
  Resynchronize with input stream on dissector errors
Future Spicy Extensions

Additional domain support
  Eg., built-in ASN.1 support

Error handling & recovery
  Resynchronize with input stream on dissector errors

Rewriting traces by reversing dissection
  Protocol-aware editing for, e.g., anonymization
Future Spicy Extensions

Additional domain support
  Eg., built-in ASN.1 support

Error handling & recovery
  Resynchronize with input stream on dissector errors

Rewriting traces by reversing dissection
  Protocol-aware editing for, e.g., anonymization

Further application domains
  File and log parsing
Spicy — a Lingua Franca?
Spicy — a Lingua Franca?

We are working to make Spicy production ready for Zeek. Currently in beta, aiming for initial stable release by end of year. Will maintain Spicy as a core part of Zeek.
Spicy — a Lingua Franca?

We are working to make Spicy production ready for Zeek.
Currently in beta, aiming for initial stable release by end of year
Will maintain Spicy as a core part of Zeek.
Spicy — a Lingua Franca?

We are working to make Spicy production ready for Zeek.
  Currently in beta, aiming for initial stable release by end of year
  Will maintain Spicy as a core part of Zeek.

We believe that Spicy could be interesting to other projects, too
  The Wireshark integration could be much extended, allowing for more control
  Integrations need to be written just once, to provide the glue
Spicy — a Lingua Franca?

We are working to make Spicy production ready for Zeek.

Currently in beta, aiming for initial stable release by end of year
Will maintain Spicy as a core part of Zeek.

We believe that Spicy could be interesting to other projects, too

The Wireshark integration could be much extended, allowing for more control
Integrations need to be written just once, to provide the glue

Could Spicy enable open source communities to share dissectors?
We are interested in feedback!

https://github.com/zeek/spicy

https://docs.zeek.org/projects/spicy

#spicy on the Zeek Slack
We are interested in feedback!

https://github.com/zeek/spicy

https://docs.zeek.org/projects/spicy

#spicy on the Zeek Slack