Spicy

A Framework For Dissecting All Your Data

zeek

corelight

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
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
12:24:12.08
12:24:12.08                                                              16
12:24:12.09
12:24:12.09                                                              16
12:24:12.09
```

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

Protocol Dissection - tcpdump

```
# tcpdump -n -r tftp_r
12:24:11.972852 IP 192                                    0 RRQ "rfc1350.txt" octet
12:24:12.077243 IP 192                                    DP, length 516
12:24:12.081790 IP 192                                    DP, length 4
12:24:12.086300 IP 192                                    DP, length 516
12:24:12.08
12:24:12.08                                               uly 1992          16
12:24:12.09
12:24:12.09                                                                 16
12:24:12.09
```

RFC 1350

```
2
--
| 0
--
```

```
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 fileame */
        ND_PRINT("\"");
        ui = nd_printztn(ndo, bp, length,
                         ndo->ndo_snapend);
        ND_PRINT("\"");
        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;
```

print-tftp.c

zeek

corelight

Protocol Dissection - tcpdump

```
# tcpdump -n -r tftp_r
12:24:11.972852 IP 192                                    0 RRQ "rfc1350.txt" octet
12:24:12.077243 IP 192                                    DP, length 516
12:24:12.081790 IP 192                                    DP, length 4
12:24:12.086300 IP 192                                    DP, length 516
12:24:12.08
12:24:12.08                                               uly 1992          16
12:24:12.09
12:24:12.09                                                                 16
12:24:12.09
```

RFC 1350

```
          2
--------
| 0
--------
```

```
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 fileame */
        ND_PRINT("\"");
        ui = nd_printztn(ndo, bp, length,
                         ndo->ndo_snapend);
        ND_PRINT("\"");
        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;
```

200 LOC total

zeek

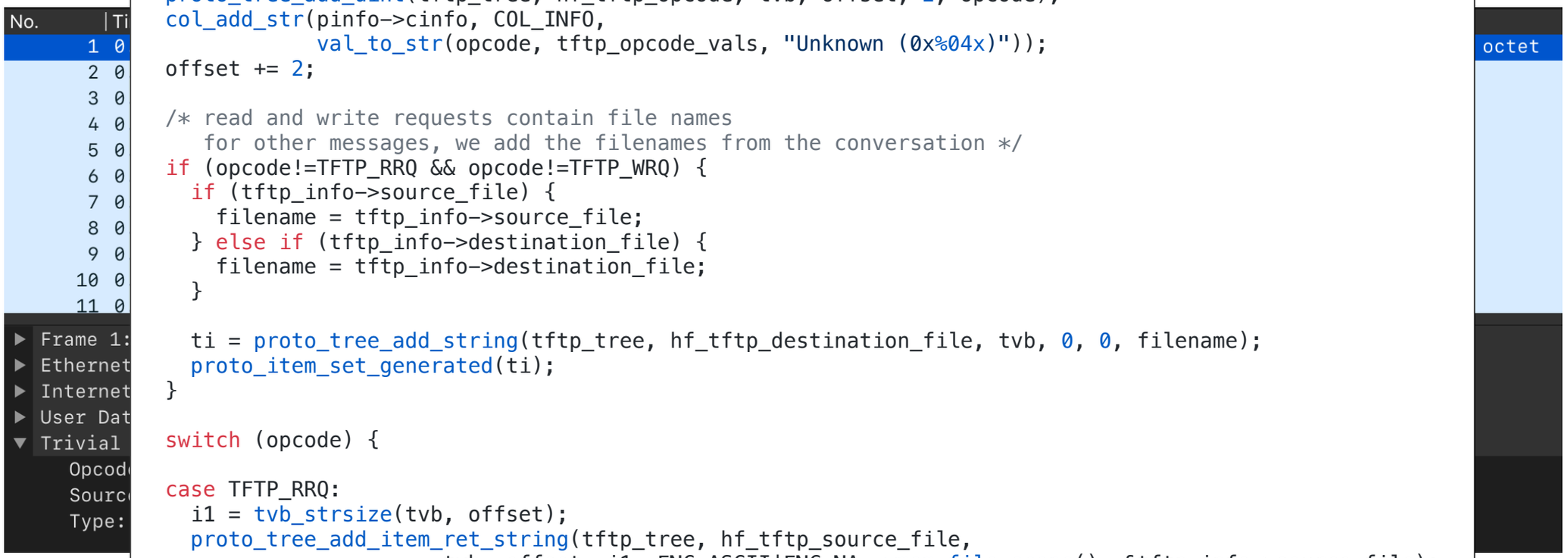
corelight

Protocol Dissection - Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.253	192.168.0.10	TFTP	62	Read Request, File: rfc1350.txt, Transfer type: octet
2	0.104391	192.168.0.10	192.168.0.253	TFTP	558	Data Packet, Block: 1
3	0.108938	192.168.0.253	192.168.0.10	TFTP	60	Acknowledgement, Block: 1
4	0.113448	192.168.0.10	192.168.0.253	TFTP	558	Data Packet, Block: 2
5	0.116109	192.168.0.253	192.168.0.10	TFTP	60	Acknowledgement, Block: 2
6	0.116143	192.168.0.10	192.168.0.253	TFTP	558	Data Packet, Block: 3
7	0.118794	192.168.0.253	192.168.0.10	TFTP	60	Acknowledgement, Block: 3
8	0.118823	192.168.0.10	192.168.0.253	TFTP	558	Data Packet, Block: 4
9	0.121531	192.168.0.253	192.168.0.10	TFTP	60	Acknowledgement, Block: 4
10	0.121564	192.168.0.10	192.168.0.253	TFTP	558	Data Packet, Block: 5
11	0.124141	192.168.0.253	192.168.0.10	TFTP	60	Acknowledgement, Block: 5

```
▶ 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)
▶ Internet Protocol Version 4, Src: 192.168.0.253, Dst: 192.168.0.10
▶ User Datagram Protocol, Src Port: 50618, Dst Port: 69
▼ Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: rfc1350.txt
    Type: octet
```

Protocol Dissection - Wireshark



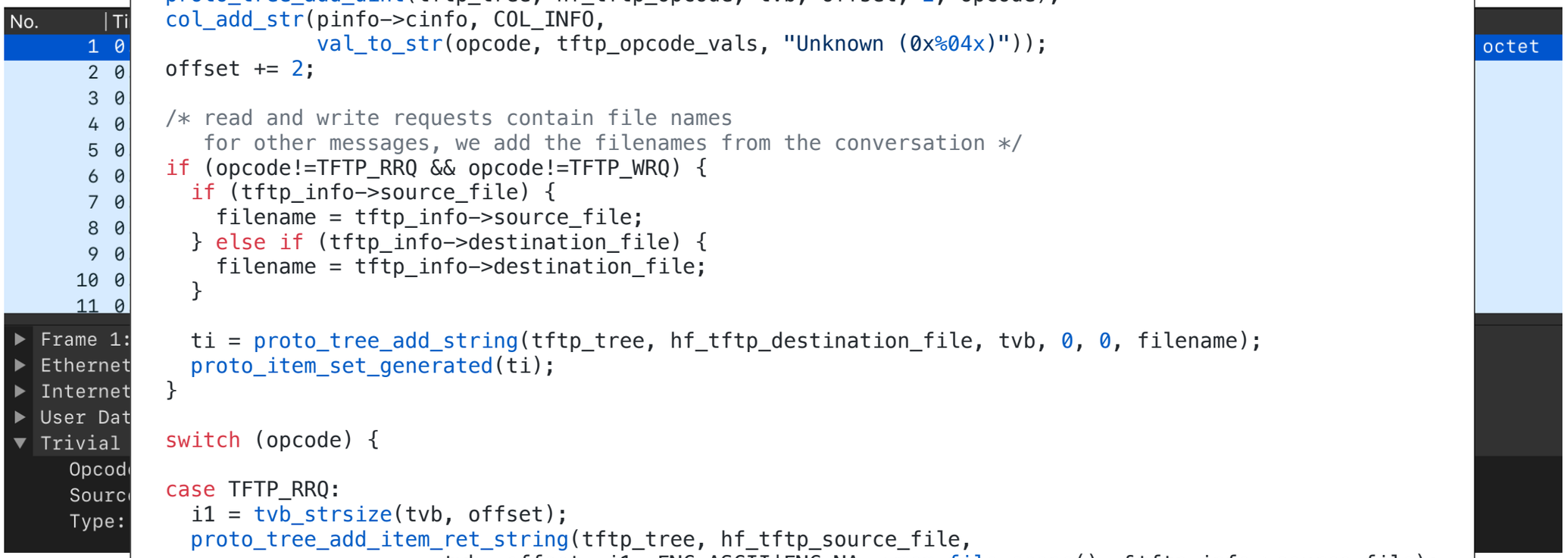

```
/* Opcode */

  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;
```

packet-tftp.c

zeek

corelight

Protocol Dissection - Wireshark




```
/* 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;
```

990 LOC total

Protocol Dissection - Zeek (FTP)

Protocol Dissection - Zeek (FTP)

```
# zeek -r ftp/ipv4.trace
# cat ftp.log
#ts                uid                id.orig_h         id.orig_p  id.resp_h         id.resp_p  user       command  arg
1329843175.680248   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  PASV     -
1329843179.815947   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  PASV     -
1329843179.926563   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  RETR     ftp://199.233.217.249/./robots.txt
1329843194.040188   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  PORT     141,142,220,235,131,46
1329843197.672179   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  PORT     141,142,220,235,147,203
1329843197.727769   CHhAvVGS1DHFjwGM9  141.142.220.235   50003      199.233.217.249   21         anonymous  RETR     ftp://199.233.217.249/./robots.txt
```

Protocol Dissection - Zeek (FTP)

```
# zeek -r ftp/ipv4.trace
# cat ftp.log
#ts                  uid
1329843175.680248    CHhAv
1329843179.815947    CHhAv
1329843179.926563    CHhAv
1329843194.040188    CHhAv
1329843197.672179    CHhAv
1329843197.727769    CHhAv
```

```
.7.249/./robots.txt
131,46
147,203
.7.249/./robots.txt
```

```
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);
```

FTP.cc

zeek

corelight

Protocol Dissection - Zeek (FTP)

```
# zeek -r ftp/ipv4.trace
# cat ftp.log
#ts                  uid
1329843175.680248    CHhAv
1329843179.815947    CHhAv
1329843179.926563    CHhAv
1329843194.040188    CHhAv
1329843197.672179    CHhAv
1329843197.727769    CHhAv
```

```
.7.249/./robots.txt
131,46
147,203
.7.249/./robots.txt
```

```
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);
```

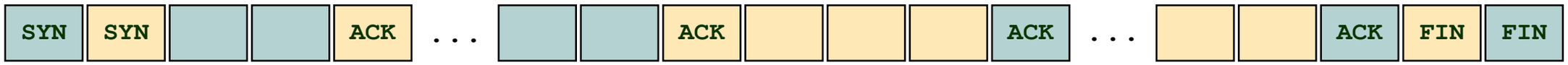
zeek

335 LOC total

corelight

Dissecting Protocols Is Hard

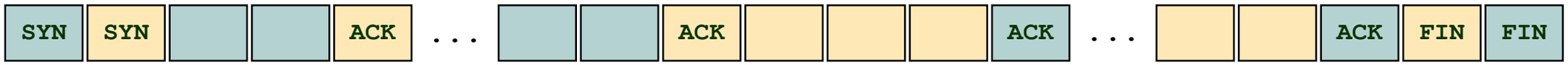
(... and no fun)



zeek

corelight

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



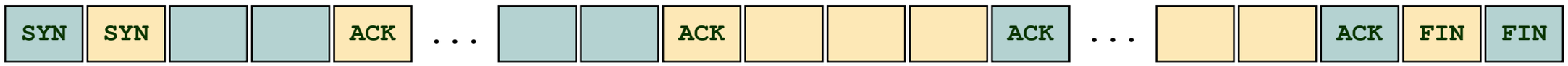
Must be robust

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

zeek

corelight

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



Must be robust

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

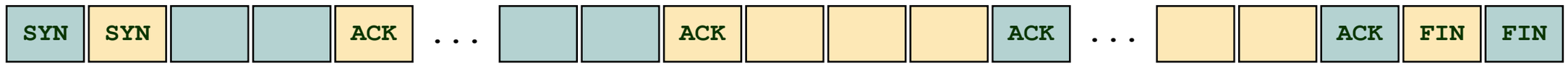
Must be efficient

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

zeek

corelight

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 …)

zeek

corelight

Dissector vulnerabilities are endemic

Dissector vulnerabilities are endemic

CVE Details
The ultimate security vulnerability datasource

WIRESHARK

Wireshark » Wireshark : Vulnerability Statistics

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2006	14	12	1												
2007	25	25	4	5											
2008	23	22	1	1							3				
2009	25	23	3	4											
2010	15	11	3	6	1										4
2011	31	29	4	9	1							1			2
2012	46	42	4	15							1				1
2013	82	82		18	1										1
2014	29	29	2	17	2										1
2015	33	33		5	1										
2016	95	89		31								1			
2017	58	2		4											
2018	79	1		12											
2019	21			3											
Total	576	400	22	130	6						4	2			9
% Of All		69.4	3.8	22.6	1.0	0.0	0.0	0.0	0.0	0.0	0.7	0.3	0.0	0.0	

zeek

corelight

Dissector vulnerabilities are endemic

zeek System for detecting network intruders in real-time
3.0.11 security Σ=0 3.0.11
Maintainer: leres@FreeBSD.org

FRESHports

zeek

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

Date	By	Description
07 Oct 2020 21:29:54 **3.0.11**	leres	security/zeek: Update to 3.0.11 to fix memory leaks and potential DOS: 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 MFH: 2020Q4 Security: 769a4f60-9056-4c27-89a1-1758a59a21f8
10 Sep 2020 00:15:49 **3.0.10**	leres	security/zeek: Update to 3.0.10 to fix memory leaks and potential DOS: https://github.com/zeek/zeek/releases/tag/v3.0.10 - Fix memory leak caused by re-entering AYIYA 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 MFH: 2020Q3 Security: 2c92fdd3-896c-4a5a-a0d8-52acee69182d
28 Jul 2020 01:09:39 **3.0.8**	leres	security/zeek: Update to 3.0.8 and address various vulnerabilities: https://github.com/zeek/zeek/releases/tag/v3.0.8 - Fix potential DNS analyzer stack overflow - Fix potential NetbiosSSN analyzer stack overflow Other fixes: - Fix DHCP Client ID Option misformat for Hardware Type 0 - Fix/allow copying/cloning of opaque of Broker::Store - Fix ConnPolling memory over-use

8 of 11 3.0.x updates fixed dissector issues!

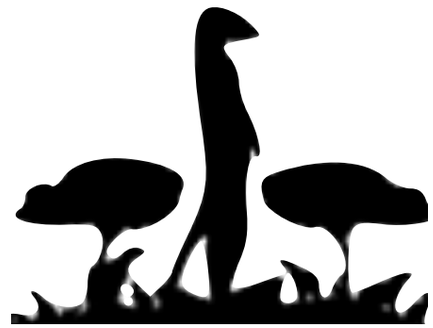
Can we make it easier to develop robust dissectors?

(... and maybe even share them??)

Language Support

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

WIRESHARK



Suricata

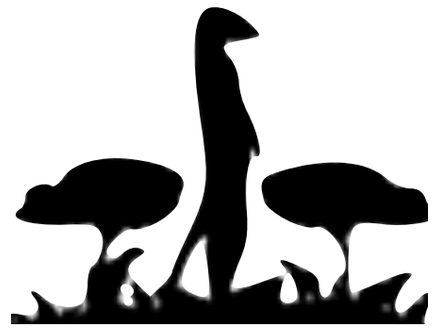
zeek

corelight

Language Support

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

WIRESHARK



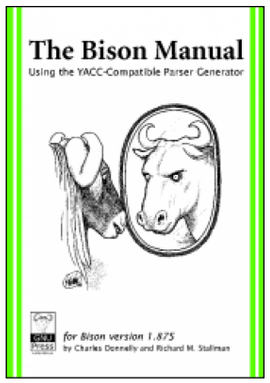
Suricata

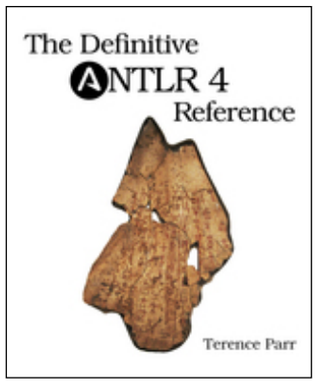
zeek

corelight

Meanwhile, in another domain …

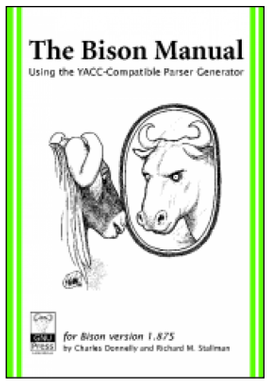
There are powerful tools for *generating* parsers from *declarative* specifications.

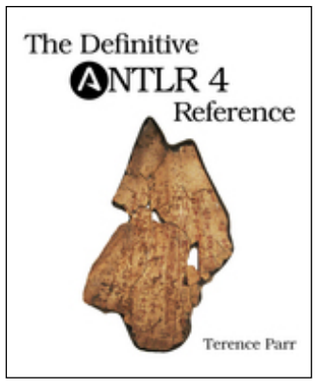




Meanwhile, in another domain …

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

zeek

corelight

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

```
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)

binpac → **class binpac::ConnectionAnalyzer** → Zeek

zeek

corelight

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

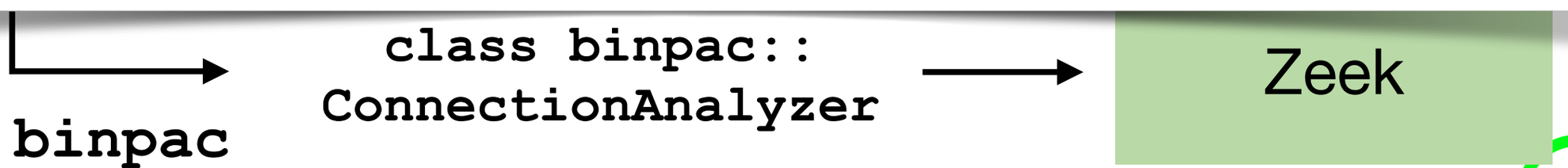
However, BinPAC solves the challenge only partially.

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`

corelight

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`

corelight

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

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

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

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";
};
```

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";
};
```

tftp-switch.spicy

```
# 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";
};
```

tftp-switch.spicy

```
# 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";
};
```

tftp.spicy

35 LOC

corelight

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";
};
```

tftp.spicy

35 LOC

```
# cat rrq.dat | spicy-dump tftp.spicy
```

A complete RFC 1350 Dissector

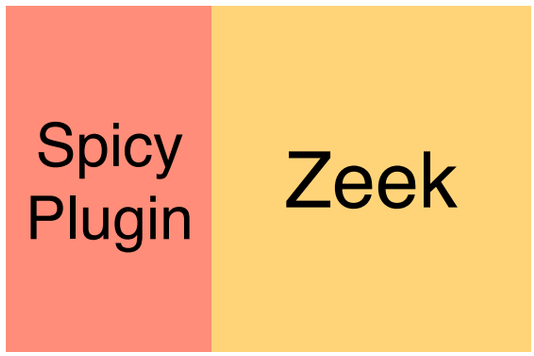
```
module TFTP;

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";
};
```

tftp.spicy

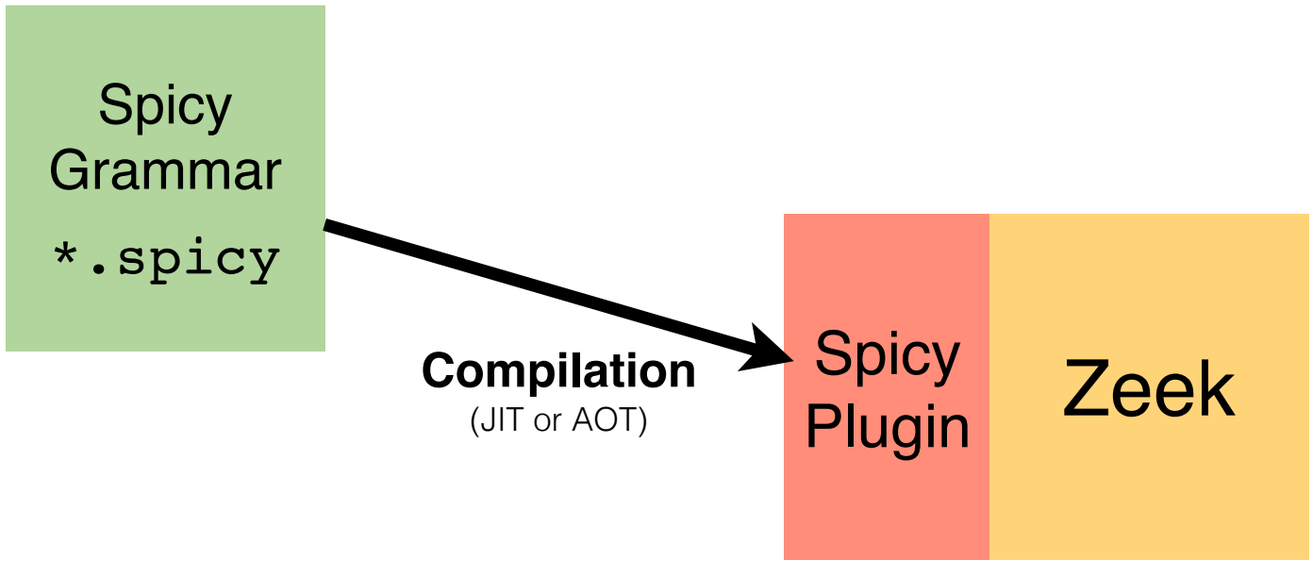
35 LOC

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

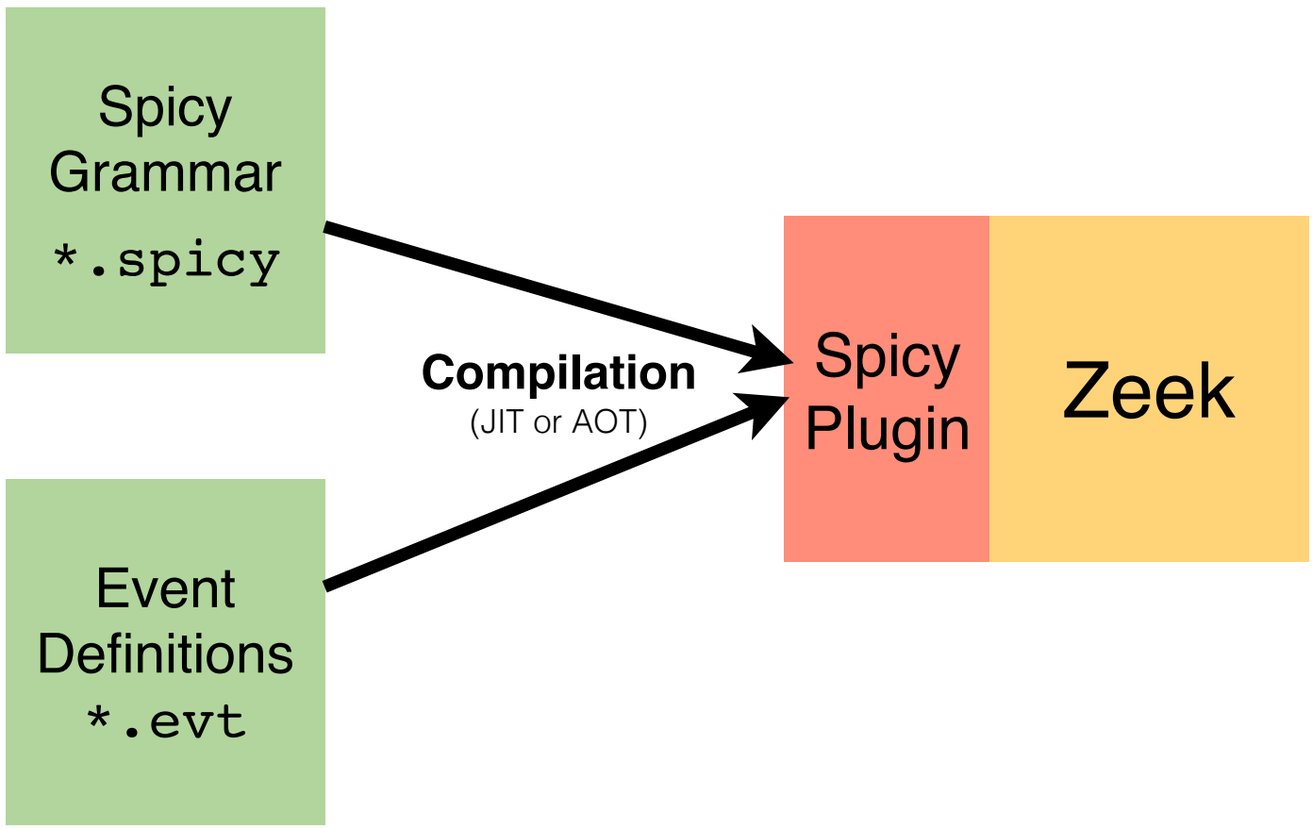
Zeek Integration



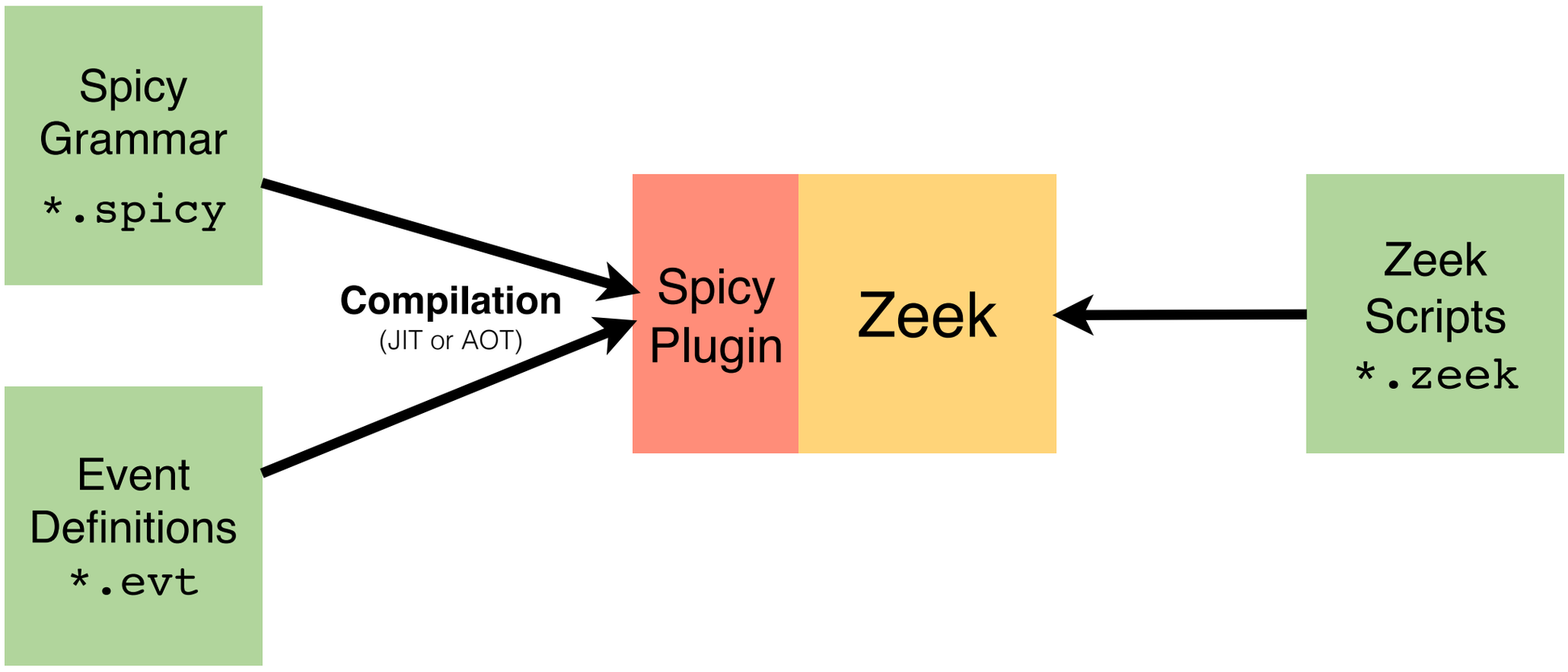
Zeek Integration



Zeek Integration



Zeek Integration



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";
};
```

tftp.spicy

corelight

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

corelight

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

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

corelight

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

```
# 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;
}
```

tftp.zeek

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

```
# 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;
}
```

tftp.zeek

```
# zeek -r tftp_rrq.pcap tftp.hlto tftp.zeek
```

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

```
# 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;
}
```

tftp.zeek

```
# 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
```

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";
};
```

tftp.spicy

```
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);
```

tftp.evt

```
# 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;
}
```

tftp.zeek

```
# 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
```

POC Wireshark 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";
};
```

tftp.spicy

corelight

POC Wireshark 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";
};
```

tftp.spicy

```
# 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

# cp spicy.so /usr/local/lib/wireshark/plugins/3-3/epan
```

corelight

POC Wireshark 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) {
```

```
# 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
```

tftp_rrq.pcap

Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.253	192.168.0.10	TFTP …	62	[$op=Opcode::RRQ, $rrq=[$filename=b"rfc1350.txt", $mode=b"octe
2	0.104391	192.168.0.10	192.168.0.253	TFTP …	558	[$op=Opcode::DATA, $rrq=(not set), $wrq=(not set), $data=[$nur
3	0.108938	192.168.0.253	192.168.0.10	TFTP …	60	[$op=Opcode::ACK, $rrq=(not set), $wrq=(not set), $data=(not s
4	0.113448	192.168.0.10	192.168.0.253	TFTP …	558	[$op=Opcode::DATA, $rrq=(not set), $wrq=(not set), $data=[$nur
5	0.116109	192.168.0.253	192.168.0.10	TFTP …	60	[$op=Opcode::ACK, $rrq=(not set), $wrq=(not set), $data=(not se

```
▶ 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)
▶ Internet Protocol Version 4, Src: 192.168.0.253, Dst: 192.168.0.10
▶ User Datagram Protocol, Src Port: 50618, Dst Port: 69
▼ TFTP Packet
    op: RRQ
  ▼ rrq
      filename: rfc1350.txt
      mode: octet
```

```
0000  00 50 8d d7 8b 43 00 0b  be 18 9a 40 08 00 45 00   ·P···C·· ···@··E·
0010  00 30 00 00 00 00 ff 11  39 65 c0 a8 00 fd c0 a8   ·0······ 9e·····
0020  00 0a c5 ba 00 45 00 1c  3e 20 00 01 72 66 63 31   ·····E·· > ··rfc1
0030  33 35 30 2e 74 78 74 00  6f 63 74 65 74 00        350.txt· octet·
```

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";
};
```

tftp.spicy

corelight

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";
};
```

tftp.spicy

```
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;
}
```

tftp.spicy, continued

POC Wireshark Integration, Take 2

```
# 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 = uni
    op: uint16 &convert=
    switch ( self.op ) {
        Opcode::RRQ   ->
        Opcode::WRQ   ->
        Opcode::DATA  ->
        Opcode::ACK   ->
        Opcode::ERROR ->
        };
};

type Opcode = enum { RRQ
                     ACK

type Request = unit(is_r
    filename: bytes &unt
    mode:     bytes &unt
};

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

type Acknowledgement = unit {
    num: uint16;
};

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

tftp.spicy

```
.rrq;
.wrq;
.data;
.ack;
.error;
essage>";

me, 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;
}
```

tftp.spicy, continued

POC Wireshark Integration, Take 2

```
# 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
```

```
module TFTP;

public type Packet = uni
    op: uint16 &convert=
    switch ( self.op ) {
        Opcode::RRQ   ->
        Opcode::WRQ   ->
        Opcode::DATA  ->
        Opcode::ACK   ->
        Opcode::ERROR ->
        };
};

type Opcode = enum { RRQ
                     ACK

type Request = unit(is_r
    filename: bytes &unt
```

```
.rrq;
.wrq;
.data;
.ack;
.error;
essage>";
```

```
me, self.mode);
```

tftp_rrq.pcap

Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.253	192.168.0.10	TFTP …	62	Read Request, File: rfc1350.txt, Transfer type: octet
2	0.104391	192.168.0.10	192.168.0.253	TFTP …	558	Data Packet, Block: 1
3	0.108938	192.168.0.253	192.168.0.10	TFTP …	60	Acknowledgement, Block: 1
4	0.113448	192.168.0.10	192.168.0.253	TFTP …	558	Data Packet, Block: 2
5	0.116109	192.168.0.253	192.168.0.10	TFTP …	60	Acknowledgement, Block: 2

```
▶ 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)
▶ Internet Protocol Version 4, Src: 192.168.0.253, Dst: 192.168.0.10
▶ User Datagram Protocol, Src Port: 50618, Dst Port: 69
▼ TFTP Packet
    op: RRQ
  ▼ rrq
      filename: rfc1350.txt
      mode: octet
```

```
0000  00 50 8d d7 8b 43 00 0b  be 18 9a 40 08 00 45 00   ·P···C·· ···@··E·
0010  00 30 00 00 00 00 ff 11  39 65 c0 a8 00 fd c0 a8   ·0······ 9e·····
0020  00 0a c5 ba 00 45 00 1c  3e 20 00 01 72 66 63 31   ·····E·· > ··rfc1
0030  33 35 30 2e 74 78 74 00  6f 63 74 65 74 00        350.txt· octet·
```

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

Older, but mostly still valid, Spicy paper: http://www.icir.org/robin/papers/acsac16-spicy.pdf