

Spicy

Generating Robust Parsers for Protocols & File Formats

Benjamin Banner

Corelight

hack.lu 2024-10-23

Our use case: Zeek



<https://zeek.org>

<https://github.com/zeek/zeek>

Zeek taps into network traffic and logs *interesting information*.

- used and deployed in large networks for 20+ years
- core developed in C++
- extensible via its own scripting language, or C++ code
- released under a permissive license

basic-auth-with-colon.trace

Apply a display filter ... <3%>

Nc	Time	Source	Destination	Protocol	Info
1	0.000000	172.24.133.205	172.24.133.205	TCP	43090 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1724347962 TSecr=0 WS=128
2	0.000120	172.24.133.205	172.24.133.205	TCP	8000 → 43090 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1724347962 TSecr=
3	0.000202	172.24.133.205	172.24.133.205	TCP	43090 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1724347962 TSecr=1724347962
4	0.000278	172.24.133.205	172.24.133.205	HTTP	GET / HTTP/1.1
5	0.000478	172.24.133.205	172.24.133.205	TCP	8000 → 43090 [ACK] Seq=1 Ack=187 Win=65408 Len=0 TSval=1724347962 TSecr=1724347962
6	0.000563	172.24.133.205	172.24.133.205	TCP	8000 → 43090 [PSH, ACK] Seq=187 Ack=187 Win=65536 Len=155 TSval=1724347963 TSecr=1724347962 [TCP
7	0.000661	172.24.133.205	172.24.133.205	TCP	43090 → 8000 [ACK] Seq=187 Ack=156 Win=65408 Len=0 TSval=1724347963 TSecr=1724347963
8	0.000737	172.24.133.205	172.24.133.205	HTTP	HTTP/1.0 200 OK (text/html)
9	0.000838	172.24.133.205	172.24.133.205	TCP	43090 → 8000 [ACK] Seq=187 Ack=799 Win=64768 Len=0 TSval=1724347963 TSecr=1724347963
	0.000921	172.24.133.205	172.24.133.205	TCP	8000 → 43090 [FIN, ACK] Seq=799 Ack=187 Win=65536 Len=0 TSval=1724347964 TSecr=1724347963
	0.001176	172.24.133.205	172.24.133.205	TCP	43090 → 8000 [FIN, ACK] Seq=187 Ack=800 Win=65536 Len=0 TSval=1724347964 TSecr=1724347964
	0.001259	172.24.133.205	172.24.133.205	TCP	8000 → 43090 [ACK] Seq=800 Ack=188 Win=65536 Len=0 TSval=1724347964 TSecr=1724347964

> Frame 4: 238 bytes on wire (1904 bits), 238 bytes captured (1
 > Internet Protocol Version 4, Src: 172.24.133.205, Dst: 172.24
 > Transmission Control Protocol, Src Port: 43090, Dst Port: 800
 > Hypertext Transfer Protocol
 > GET / HTTP/1.1\r\n
 Host: 172.24.133.205:8000\r\n
 User-Agent: python-requests/2.31.0\r\n
 Accept-Encoding: gzip, deflate\r\n
 Accept: */*\r\n
 Connection: keep-alive\r\n
 > Authorization: Basic dGVzdDox0jM0\r\n
 Credentials: test:1:34
 \r\n
 [Response in frame: 8]
 [Full request URI: http://172.24.133.205:8000/]

0000	45 00 00 ee 44 fa 40 00	40 06 91 44 ac 18 85 cd	E . . . D @ . . @ . D
0010	ac 18 85 cd a8 52 1f 40	00 b5 8b dc c5 70 36 b0 R @ p6 .
0020	80 18 02 00 eb df 00 00	01 01 08 0a 66 c7 76 3a f : v .
0030	66 c7 76 3a 47 45 54 20	2f 20 48 54 54 50 2f 31	f : v : GET / HTTP/1
0040	2e 31 0d 0a 48 6f 73 74	3a 20 31 37 32 2e 32 34	. 1 . Host : 172.24
0050	2e 31 33 33 2e 32 30 35	3a 38 30 30 30 0d 0a 55	. 133.205 :8000 . U
0060	73 65 72 2d 41 67 65 6e	74 3a 20 70 79 74 68 6f	ser-Agen t: pytho
0070	6e 2d 72 65 71 75 65 73	74 73 2f 32 2e 33 31 2e	n-reques ts/2.31.
0080	30 0d 0a 41 63 63 65 70	74 2d 45 6e 63 6f 64 69	0 . Accep t-Encodi
0090	6e 67 3a 20 67 7a 69 70	2c 20 64 65 66 6c 61 74	ng: gzip , deflat
00a0	65 0d 0a 41 63 63 65 70	74 3a 20 2a 2f 2a 0d 0a	e . Accep t: */*
00b0	43 6f 6e 65 63 74 69	6f 6e 3a 20 6b 65 65 70	Connecti on: keep
00c0	2d 61 6c 69 76 65 0d 0a	41 75 74 68 6f 72 69 7a	-alive . Authoriz
00d0	61 74 69 6f 6e 3a 20 42	61 73 69 63 20 20 64 47	ation: Basic dG
00e0	56 7a 64 44 6f 78 4f 6a	4d 30 0d 0a 0d 0a	VzdDox0j M0

Frame (238 bytes) Basic Credentials (9 bytes) Packets: 12 Profile: Default

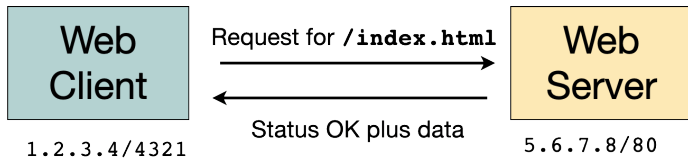
HTTP Authorization header (http.authorization), 36 bytes

\$ zEEK -r basic-auth-with-colon.pcap

```
conn.log:
{
  "uid": "Cmo3v02TYe9eXWzx58",
  "ts": 1724831789.595534,
  "id.orig_h": "172.24.133.205",
  "id.orig_p": 43090,
  "id.resp_h": "172.24.133.205",
  "id.resp_p": 8000,
  "proto": "tcp",
  "service": "http",
  "duration": 0.0011758804321289062,
  "orig_bytes": 186,
  "resp_bytes": 798,
  "conn_state": "SF",
  "local_orig": true,
  "local_resp": true,
  "missed_bytes": 0,
  "history": "ShADadFf",
  "orig_pkts": 6,
  "orig_ip_bytes": 506,
  "resp_pkts": 6,
  "resp_ip_bytes": 1118
}
```

```
http.log:
{
  "uid": "Cmo3v02TYe9eXWzx58",
  "ts": 1724831789.595812,
  "id.orig_h": "172.24.133.205",
  "id.orig_p": 43090,
  "id.resp_h": "172.24.133.205",
  "id.resp_p": 8000,
  "trans_depth": 1,
  "method": "GET",
  "host": "172.24.133.205:8000",
  "uri": "/",
  "version": "1.0",
  "user_agent": "python-requests/2.31.0",
  "request_body_len": 0,
  "response_body_len": 643,
  "status_code": 200,
  "status_msg": "OK",
  "tags": [],
  "username": "test",
  "resp_fuids": [ "FM4Ls72L4REzbA6llg" ],
  "resp_mime_types": [ "text/html" ]
}
```

```
files.log:
{
  "fuid": "FM4Ls72L4REzbA6llg",
  "ts": 1724831789.596271,
  "uid": "Cmo3v02TYe9eXWzx58",
  "id.orig_h": "172.24.133.205",
  "id.orig_p": 43090,
  "id.resp_h": "172.24.133.205",
  "id.resp_p": 8000,
  "source": "HTTP",
  "depth": 0,
  "analyzers": [],
  ...
}
```



↳ `connection_established(1.2.3.4/4321⇒5.6.7.8/80)`

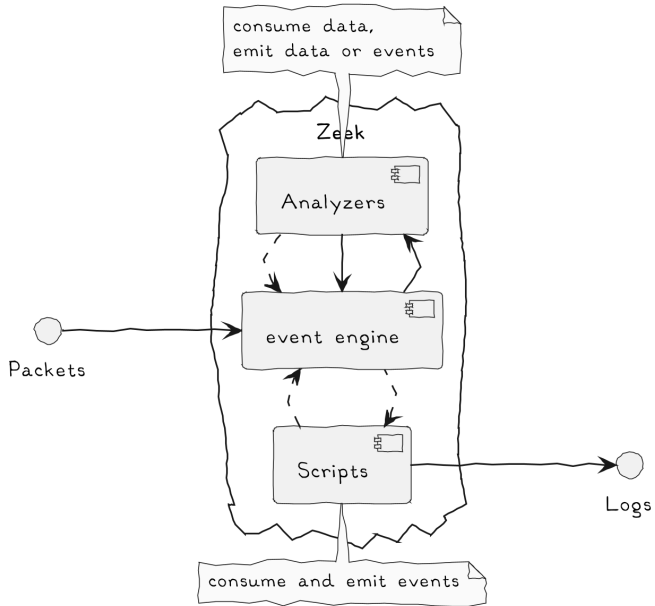
`GET /index.html HTTP/1.1 ...` TCP stream reassembly for originator

↳ `http_request(1.2.3.4/4321⇒5.6.7.8/80, "GET", "/index.html")`

`200 OK ...` TCP stream reassembly for responder

↳ `http_reply(1.2.3.4/4321⇒5.6.7.8/80, 200, "OK", data)`

↳ `connection_finished(1.2.3.4/4321, 5.6.7.8/80)`



Adding new analyzers to Zeek

- historically required C++ implementation
- C++ has a significant barrier of entry
- code can be pretty low level hiding business logic
- C++ is not a safe language

High-level language for parsing: Spicy



Goals

- robust language with support for both syntax and logic (no need for C++!)
- efficient enough to parse many concurrent connections with low latency and memory overhead
- transparent integration in Zeek
- allow creating parsers reusable outside of Zeek ecosystem

Project

- started as ICSI research prototype
- actively used in the Zeek ecosystem to write packet, protocol and file analyzers
- <https://github.com/zeek/spicy>, #spicy

General structure

- Out-of-the-box Spicy provides parsing for some basic types: bytes, `/(reg)(ular)(expressions)/`, integer, real, addr, vector.
- A Spicy parser consists of a unit with zero or more fields, e.g.,

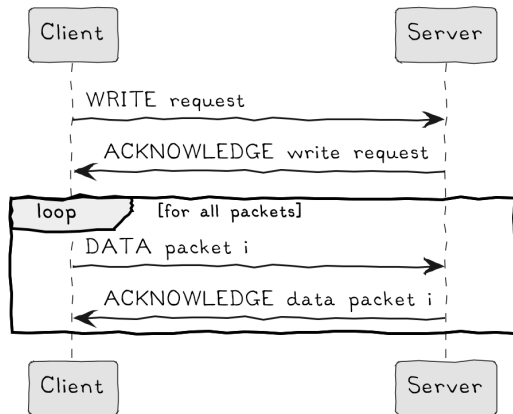
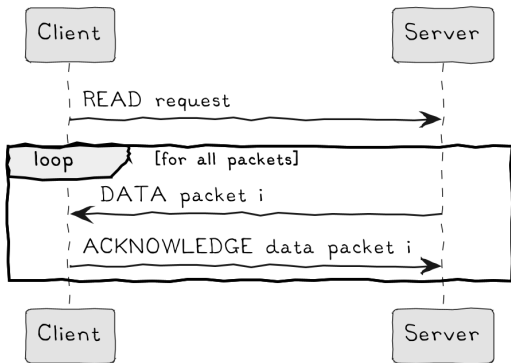
```
type MyInt = unit {  
    value: int8;  
};
```

Unit types can be used for fields.

- Procedural code can be executed via hooks which can be attached to various parsing stages, e.g.,

```
type X = unit {  
    len: uint64 {  
        print "extracted len=%llu" % self.len;  
    }  
    data: bytes &size=self.len;  
  
    on %done {  
        print "parsed X=%s" % self;  
    }  
};
```

Trivial File Transfer Protocol (TFTP)



Full parser at <https://github.com/zeek/spicy-tftp>.

Figure 5-1: RRQ/WRQ packet

#

2 bytes string 1 byte string 1 byte

-----

| Opcode | Filename | 0 | Mode | 0 |

-----

```
public type Packet = unit {
```

```
  op: uint16
```

```
  rrq: ReadRequest;
```

```
  # Hook triggered once this unit has finished parsing.
```

```
  on %done { print self; }
```

```
};
```

```
type ReadRequest = unit {
```

```
  filename: bytes &until=b"\x00";
```

```
  mode:        bytes &until=b"\x00";
```

```
};
```

Figure 5-1: RRQ/WRQ packet

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

```
public type Packet = unit {  
  op: uint16 &convert=Opcode($$);  
  switch ( self.op ) {  
    Opcode::RRQ   -> rrq: Request(True);  
    Opcode::WRQ   -> wrq: Request(False);  
    # TODO: Add handling for other opcodes.  
  };  
  
  # Hook triggered once this unit has finished parsing.  
  on %done { print self; }  
};  
  
type Request = unit(is_read: bool) { # Parameter is implicitly in`  
  filename: bytes &until=b"\x00";  
  mode:     bytes &until=b"\x00";  
};  
  
type Opcode = enum { RRQ = 0x01, WRQ = 0x02, DATA = 0x03,  
  ACK = 0x04, ERROR = 0x05 };
```

Lookahead parsing

Lookahead parsing allows to dynamically react to the seen data.

```
type X = unit {  
  : (b"A")[]; # Extract unknown number of literal 'A' bytes.  
  end_marker: skip b"\x00";  
};
```

.....

```
type Message1 = unit {  
  : skip uint8(1);  
  # ... parse data.  
};
```

```
type Message2 = unit {  
  : skip uint8(2);  
  # ... parse data.  
};
```

```
public type X = unit {  
  switch {  
    -> m1: Message1;  
    -> m2: Message2;  
    # For unknown message types  
    # consume all data.  
    -> : skip bytes &eod;  
  };  
};
```

Recovering from input gaps/parse errors

```
type X = unit {
  a: b"begin-of-X";
  b: bytes &size=10;
};

type Foo = unit {
  xs: (X &synchronize)[];
};

on Foo::%synced {
  # If needed validate and/or
  # fix internal state.
  if (unit_is_fine)
    # Continue regular parsing.
    confirm;
  else
    # Try to synchronize again.
    reject;
}
```

```
type Bar = unit {
  a: A;
  b: B;
  c: C &synchronize;
  d: D;
};
```

More low-level manual backtracking is also supported.

Reshaping data

```
# csv_naive.spicy
module csv_naive;

public type CSV = unit {
  rows: Row[];
};

type Row = unit {
  cols: bytes &until=b"\n" &convert=$$.split(b",");
} &convert=self.cols;

on CSV::%done {
  print self;
}

.....

$ printf '1,2,3\nA,B,C\n' | spicy-driver csv_naive.spicy
[$rows=[[b"1", b"2", b"3"], [b"A", b"B", b"C"]]]
```

Extensibility

```
// mylibrary.cc  
namespace MyLibrary {  
std::string rotl3(const std::string& in) {  
    // ...  
} // MyLibrary
```

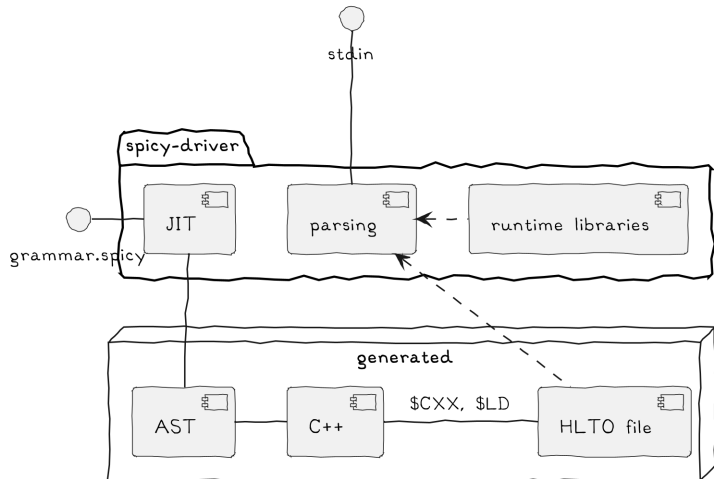
```
# mylibrary.spicy  
module MyLibrary;
```

```
public function rotl3(s: string) : string &cxxname="MyLibrary::rotl3";
```

```
# example.spicy  
module Example;  
import MyLibrary;  
const data = "Hello, world!";  
print "%s' -> '%s' -> '%s'" % (data,  
                               MyLibrary::rotl3(data),  
                               MyLibrary::rotl3(MyLibrary::rotl3(data)));
```

```
.....  
$ spicyc -j rotl3.spicy mylibrary.spicy mylibrary.cc  
'Hello, world!' -> 'Uryyb, jbeyq!' -> 'Hello, world!'
```


Spicy processing pipeline



```
# foo.spicy
# ...
public type X = unit {
  a: uint8;
  b: bytes &eod;
};
```

```
.....

$ spicyc -c foo.spicy
namespace <internal> {
struct X /*: ...*/ {
  // ...
  static auto parse3(::hilti::rt::ValueReference<::spicy::rt::ParsedUnit>& gunit,
    ::hilti::rt::ValueReference<::hilti::rt::Stream>& data,
    const std::optional<::hilti::rt::stream::View>& cur,
    const std::optional<::spicy::rt::UnitContext>& context)
    -> ::hilti::rt::stream::View;

  std::optional<::hilti::rt::integer::safe<uint8_t>> a{};
  std::optional<::hilti::rt::Bytes> b{};
};
}
```

ParsedUnit can be reflected on.

tftp_rrq.pcap

Apply a display filter ... <3%/>

No.	Time	Source	Destination	Protoc	Length	Info
1	0.000000	192.168.0.253	192.168.0.10	TFTP	62	Read Request, File: rfc1350.txt, Transfer t
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 7: 60 bytes on wire (480 bits), 60 bytes captured (480 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: 3445

▼ Trivial File Transfer Protocol

 Opcode: Acknowledgement (4)

 [Source File: rfc1350.txt]

 Block: 3

```

0000  00 50 8d d7 8b 43 00 0b  be 18 9a 40 08 00 45 00  .P...C...@..E.
0010  00 20 00 03 00 00 ff 11  39 72 c0 a8 00 fd c0 a8  .9r.....
0020  00 0a c5 ba 0d 75 00 0c  aa 47 00 04 00 03 00 00  .u..G...
0030  00 00 00 00 00 00 00 00  00 00 00 00

```

Block number (tftp.block), 2 bytes Packets: 99 - Displayed: 99 (100.0%) Profile: Default

Features not touched on

- standard library
- *preprocessing* data through filter abstraction
- support for *reassembly* with sink abstraction
- unit context to attach additional information (e.g., to *share state between both sides of connection*)
- *control data* a field is parsed from with attributes
- *random access* support to completely change input position ...

Use in Zeek

- Spicy knows nothing about Zeek
- Zeek-specific *spicy-plugin* glue layer adapts types and matches up Spicy hooks with Zeek events:

```
# tftp.evt
protocol analyzer spicy::TFTP over UDP:
    parse with TFTP::Packet,
    port 69/udp;

import TFTP;

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

- parser developers typically write a Spicy grammar and an EVT file starting from a template, and implement additional analyses in Zeek scripts

Spicy enabled more users to add parsers to Zeek.

New parsers in Zeek by default started with Spicy today.

Consider starting your Zeek analyzer from Spicy *zkg* templates.

What would you look at if parsing had less intellectual overhead in your project?

- automation protocols
- industrial control protocols
- over Bluetooth/USB?
- file formats
- ...



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

<https://static.zeek.org/spicy-course>

#spicy on Zeek Slack

<https://github.com/zeek/spicy>

Sample STUN analyzer

<https://github.com/corelight/zeek-spicy-stun>

```
$ for f in *(spicylevtlzeeklsig); do echo $f $(rg -v '^(#!$)' $f | wc -l); done | column -t
analyzer.spicy      160
analyzer.evt        70
zeek_analyzer.spicy 9
dpd.sig             10
consts.zeek         79
main.zeek           179
__load__.zeek       3
```


Sample STUN analyzer

<https://github.com/corelight/zeek-spicy-stun>

```
const MAGIC_COOKIE = /\x21\x12\xA4\x42/;
public type STUNPacket = unit {
  var M: uint16;
  var C: uint16;

  msgtype: bitfield(16) {
    m0: 0..3;
    c0: 4;
    m1: 5..7;
    c1: 8;
    m2: 9..13;
    zeros: 14..15 &requires=($$ == 0);
  } {
    self.M = self.msgtype.m0 + 16*self.msgtype.m1 + 128*self.msgtype.m2;
    self.C = self.msgtype.c0 + 2*self.msgtype.c1;
  }

  msg_length: uint16;
  : MAGIC_COOKIE;
  trans_id: bytes &size=12;
  attributes: Attribute(self.M, self.C, self.trans_id)[];
};
```