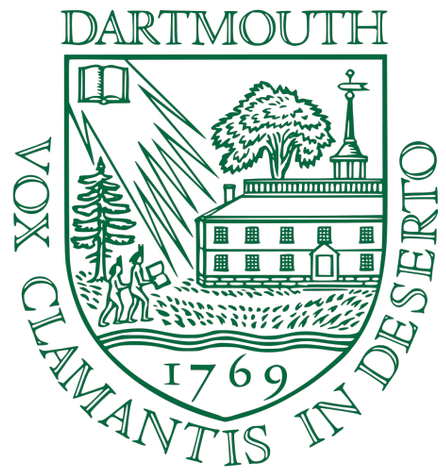
Input Handling Done Right

Building Hardened Parsers using Language-Theoretic Security

Prashant Anantharaman
Michael Millian
Sergey Bratus

Dartmouth College

SecDev 2017
24th September, 2017



Logistics

- Find the material at http://langsec.org/secdev.html
- Get the docker image –
  https://hub.docker.com/r/prashantbarca/hammer-parser

`docker run -it prashantbarca/hammer-parser:secdevfinal`

Contents

- Parsers and Security
- LangSec Viewpoint
- Parser combinators
- Looking at a DNS parser
- Looking at an HTTP parser
- Testing with libfuzzer

1. Parsers and Security

Parsers are dangerous & important

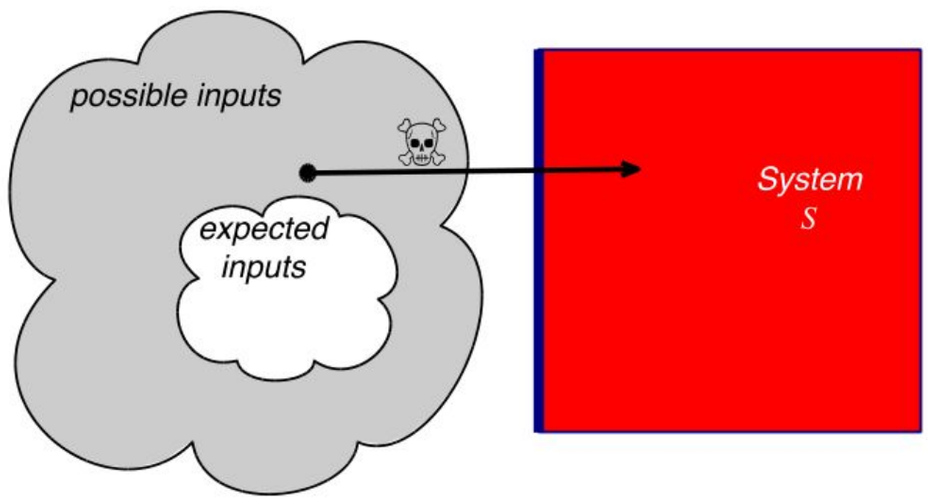
- Parsers are the front line of application security
- Parser must assure **pre-conditions** of subsequent code
- Parsers done right actually deliver *assurance*, make it possible to prove *correctness*
- Parsers done wrong are attacker's engine of unexpected computation

  Hence:

  - It should be clear what a parser expects & enforces
  - *"Parser code should look like the grammar"*

Data format is code's destiny

- A parser takes input data and builds a structural representation of the input
- "Validating input" is judging what effect it will have on code
- Some formats are too hard to parse



What goes wrong in parsing

- Object boundaries in message cross or overlap
- Objects are embedded in other objects incorrectly
- Objects that should appear in a given position aren't there
- Objects appear in a position that isn't expected

- Pre-conditions expected by the rest of the code are not met
- Code's behavior on input is *not predictable*
  - Buffer overflows, memory corruption, exploitation

Heartbleed is a "shotgun parser" bug

```
hbtype = *p++;
n2s(p, payload);
pl = p;
```

Heartbeat sent to victim

SSLv3 record:

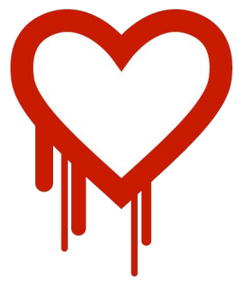
SSL3_RECORD

Length
4 bytes

HeartbeatMessage

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte

hbtype payload



```
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Where OpenSSL's parser went wrong

```
-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;
-
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);
```

```
+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
+               unsigned int write_length = 1 /* heartbeat type */ +
+                                           2 /* heartbeat length */ +
+                                           payload + padding;
                int r;
```

Ethereum DAO disaster

```
1  contract investmentBank {
2
3      function () public { //add balance
4          balance[msg.sender] += msg.value; //increment balance
5      }
6
7      //elision
8
9      ///Withdraw a sender's entire balance
10     function withdrawAll() public {
11         int r = msg.sender.call.value(balance[msg.sender])();
12         if (!r) { throw; } //have to check...
13         balance[msg.sender] = 0; //before deducting.
14     }
15
16 }
```

"To find out what it does, you need to run it"

Recursion is trouble

```
18 contract maliciousWallet {
19
20     c = address_of_an_investmentBank;
21
22     //elision
23
24     function seedBalance() {
25         investmentBank bank = investmentBank(c);
26         bank.call.value(100)(); //give 100 ether to bank
27     }
28
29     //default function, called when someone sends us ether
30     function () public {
31         investmantBank bank = investmentBank(c); //instantiate reference
32         c.withdrawAll();
33     }
34 }
```

CVE-2015-1427

"Sanitized" Groovy scripts in inputs + JVM Reflection = Pwnage

```
def banner():
    print """\x1b[1;32m
```


```
 Exploit for ElasticSearch , CVE-2015-1427   Version: %s\x1b[0m""" %(__version__)

def execute_command(target, command):
    payload = """{"size":1, "script_fields": {"lupin":{"script":
"java.lang.Math.class.forName(\\"java.lang.Runtime\\").getRuntime().exec(\\"%s\\").getText()"}}}""" %(command)
    try:
        url = "http://%s:9200/_search?pretty" %(target)
        r = requests.post(url=url, data=payload)
    except Exception, e:
        sys.exit("Exception Hit"+str(e))
    values = json.loads(r.text)
    f[redacted]ingjson = values['hits']['hits'][0]['fields']['lupin'][0]
    print f[redacted]ingjson.strip()


def exploit(target):
    print "{*} Spawning Shell on target... Do note, its only semi-interactive... Use it to drop a better
payload or something"
    while True:
        cmd = raw_input("~$ ")
```

"Ruby off Rails"

- "Why parse if we can **eval(user_input)**?"
- Oh so many. Joernchen of Phenoelit *Phrack 69:12*, Egor Homakov ("*Don't let YAML.load close to any user input*"), ...

- CVE-2016-6317, "*Mitigate by casting the parameter to a **string** before passing it to Active Record*"

"Shellshock" CVE-2014-6271
parse_and_execute(CGI_input)

```
/* Initialize the shell variables from the current environment.
   If PRIVMODE is nonzero, don't import functions from ENV or
   parse $SHELLOPTS. */
void
initialize_shell_variables (env, privmode)
     char **env;
     int privmode;
{
  [...]
  for (string_index = 0; string = env[string_index++]; )
    {

      [...]
      /* If exported function, define it now.  Don't import functions from
     the environment in privileged mode. */
      if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
      {
        [...]
        parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
        [...]
      }
}
```

Parsing & protocol anti-patterns

- "Shotgun parsers": input validity checks intermixed with processing code; no clear separation boundary
  - OpenSSL's Heartbleed, GNU TLS Hello bug, …
- Unnecessarily complex syntax (e.g., context-sensitive where context-free or regular would suffice)
  - Objects' interpretation & legality depends on sibling object contents
- Parser differentials (parsers disagree about message contents)
  - X.509 CA vs client bugs, Android Master Key bugs, …

Anti-patterns of C parsers

- Pointer arithmetic for stepping through variable length fields shouldn't be allowed.

```
-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;

        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
+               unsigned int write_length = 1 /* heartbeat type */ +
+                                           2 /* heartbeat length */ +
+                                           payload + padding;
                int r;
```

2. The LangSec Viewpoint

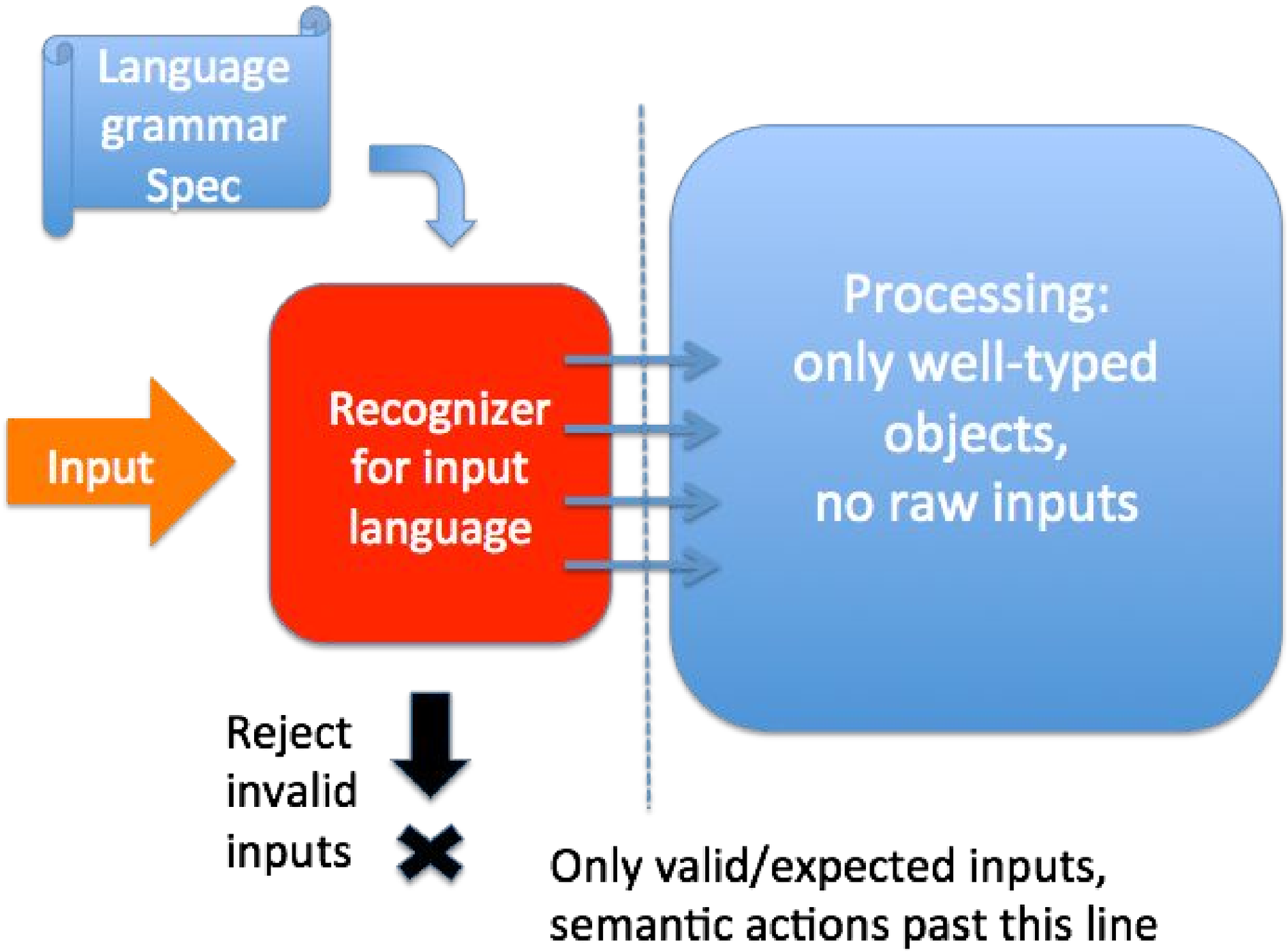
LangSec: Mission assurance for parsers

- Formal language theory: The fundamental underlying science for validating input and constraining input driven computation
- Seeks to prevent recurring programmer errors in protocols by identifying and eliminating problematic syntax & ambiguity

LangSec: Mission assurance for parsers

- LangSec identifies protocol/syntax features that make security an uphill battle:
  - specification is ambiguous: programmers disagree
  - validity check is too hard for a programmer to get right
  - several sources of *truth*
  - too much context needed to judge an object as valid or invalid

Part of a the solution: *Recognizer Pattern*



Solve language problems with a language approach

- Start with a grammar
  - If you don’t know what valid or expected syntax/content of a message is, how can you check it? Or interoperate?
  - If the protocol comes without a grammar, you need to derive one. It’s the only way! :(

- Write the parser to look like the grammar: succinct & *incrementally testable*

- Don’t start processing before you’re done parsing

Principles & Desiderata

- Valid or expected syntax must be clear from parser code at any point
- Code clearly reads as a specification of valid inputs
- Any code that accepts a syntax element must be testable on its own
- Toolkit should discourage troublesome syntax (If some feature is hard to express, it's likely trouble)

- Natural fit: **Parser Combinators**
  - What, in C? Yes! In C, Python, Ruby, PHP, .Net, Java, ...

3. Diving into the Code

Parser combinators: a natural choice

- Developed for Haskell, but good enough for everyone!
- Hammer parser construction kit: C/C++
- Bindings for Java, Python, Ruby, .NET, Go
- Three algorithmic parsing back-ends

- Freely available on GitHub:
  https://github.com/UpstandingHackers/hammer

Parser combinators at a glance (DNP3)

```
05 64 14 F3
01 00 00 04
0A 3B C0 C3
01 3C 02 06
3C 03 06 3C
04 06 3C 01
06 9A 12
```

```
start = h_token("\x05\x64");
len = h_int_range(h_uint8(), 5, 255);
ctrl = h_uint8();
dst = h_uint16();
src = h_int_range(h_uint16(), 0, 65519);
crc = h_uint16();
hdr = h_attr_bool(h_sequence(h_ignore(start),
    len, ctrl, dst, src, crc, NULL),
    validate_crc);
frame = h_attr_bool(h_sequence(hdr,
      h_optional(transport_frame),
      h_end_p(), NULL),validate_len);
```

Header

Sync	Length	Link Control	Destination Address	Source Address	CRC

Parser Combinators: code looks like the grammar

- Each syntax primitive (AST leaf) gets its own parser

```
HParser *seqno = h_bits(4, false);
HParser *bit   = h_bits(1, false);
...
```

- Primitive parsers are combined into higher-level structures with *combinators*

```
h_choice, h_many, h_many1, h_sequence…
```

- Define your own combinators (if you need to)

4. Parser Combinator

Demo

Combinators to try

C Primitives

1. *h_token(const uint8_t *str, const size_t len)
2. *h_ch(const uint8_t c)
3. *h_ch_range(const uint8_t lower, const uint8_t upper)
4. *h_bits(size_t len, bool sign)
5. *h_int64(void)
6. *h_in(const uint8_t *charset, size_t length)
7. *h_not_in(const uint8_t *charset, size_t length)
8. *h_end_p(void)

Combinators to try

Python Primitives

1. h.token(str, len)
2. h.ch(ch)
3. h.ch_range(c1,c2)
4. h.bits(len, sign)
5. h.int64()
6. h.in(charset)
7. h.not_in(charset)
8. h.end_p()

Combinators to try

C Combinators

1. *h_sequence(...) → A, B, C
2. *h_choice(...) → A | B | C
3. *h_many(const HParser *p) → A*
4. *h_many1(const HParser *p) → A+
5. *h_repeat_n(const HParser *p, const size_t n) → A{n}
6. *h_optional(const HParser *p) → A?
7. *h_sepBy1(const HParser *p, const HParser *sep)
   → A (*sep* A)+
8. *h_sepBy(const HParser *p, const HParser *sep)
   → A (*sep* A)*

Combinators to try

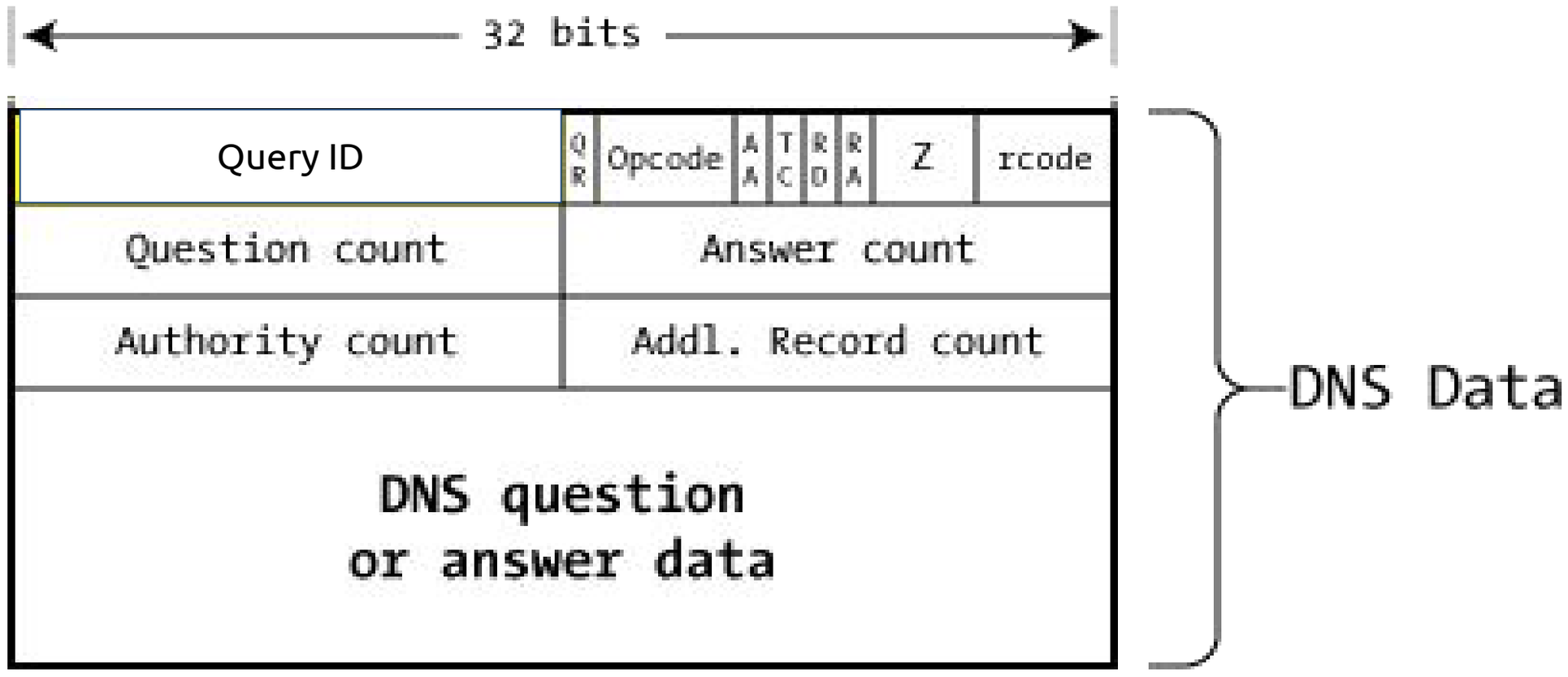
Python Combinators

1. h.sequence(...)
2. h.choice(...)
3. h.butnot(p1, p2)
4. h.many(p)
5. h.many1(p)
6. h.repeat_n(p, n)
7. h.optional(p)
8. h.sepBy1(p, sep)
9. h.sepBy(p, sep)

5. Going through the DNS Parser

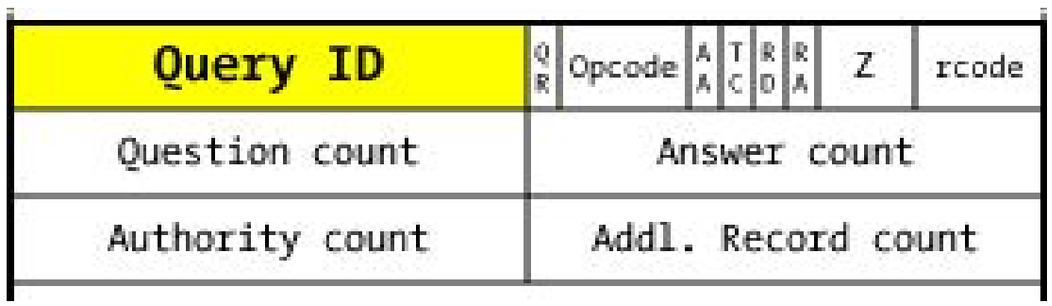
DNS packet format

```
H_RULE  (domain,   init_domain());
H_AVRULE(hdzero,   h_bits(3, false));
h_bits(16, false) // ID
```



DNS packet format

```
H_ARULE (header,   h_sequence(h_bits(16, false), // ID
                  h_bits(1, false),  // QR
                  h_bits(4, false),  // opcode
                  h_bits(1, false),  // AA
                  h_bits(1, false),  // TC
                  h_bits(1, false),  // RD
                  h_bits(1, false),  // RA
                  hdzero,            // Z
                  h_bits(4, false),  // RCODE
                  h_uint16(),        // QDCOUNT
                  h_uint16(),        // ANCOUNT
                  h_uint16(),        // NSCOUNT
                  h_uint16(),        // ARCOUNT
                  NULL));
```



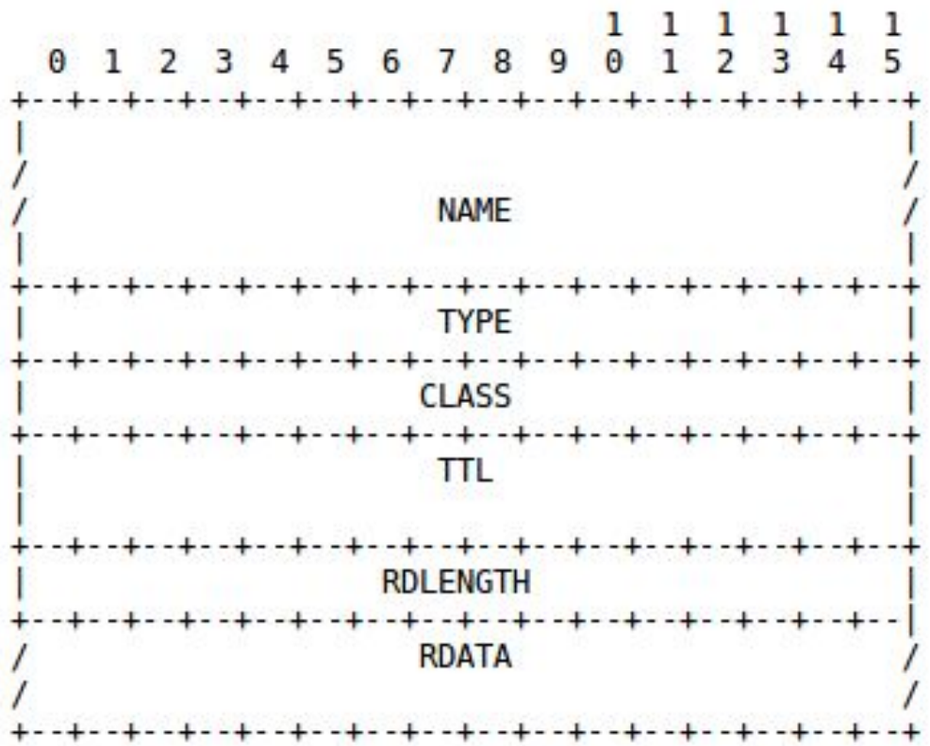
DNS using H_ARULE

```
HParsedToken* act_header(const HParseResult *p, void* user_data) {
  HParsedToken **fields = h_seq_elements(p->ast);
  dns_header_t header_ = {
      .id     = H_CAST_UINT(fields[0]),
      .qr     = H_CAST_UINT(fields[1]),
      .opcode = H_CAST_UINT(fields[2]),
      .aa     = H_CAST_UINT(fields[3]),
      .tc     = H_CAST_UINT(fields[4]),
      .rd     = H_CAST_UINT(fields[5]),
      .ra     = H_CAST_UINT(fields[6]),
      .rcode  = H_CAST_UINT(fields[7]),
      .question_count   = H_CAST_UINT(fields[8]),
      .answer_count     = H_CAST_UINT(fields[9]),
      .authority_count  = H_CAST_UINT(fields[10]),
      .additional_count = H_CAST_UINT(fields[11])
  };

  dns_header_t *header = H_ALLOC(dns_header_t);
  *header = header_;

  return H_MAKE(dns_header_t, header);
}
```

Query ID	QR	Opcode	AA	TC	RD	RA	Z	rcode
Question count	Answer count							
Authority count	Addl. Record count							

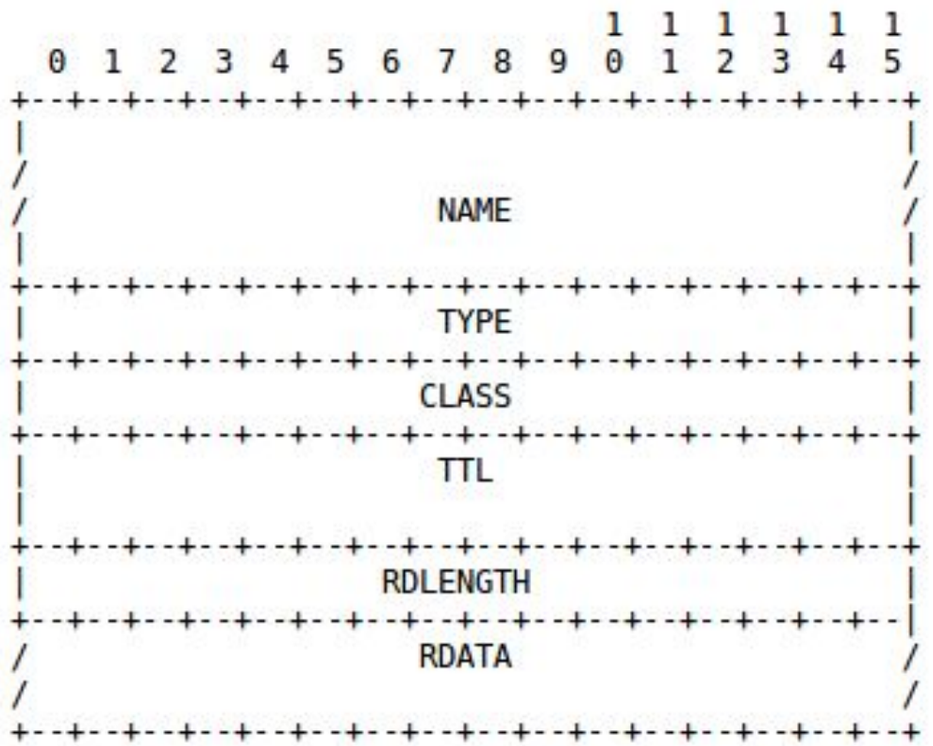
Resource Record Format

```
H_RULE  (type,    h_int_range(h_uint16(), 1, 16));
H_RULE  (class,   h_int_range(h_uint16(), 1, 4));
H_RULE  (len,     h_int_range(h_uint8(), 1, 255));
H_ARULE (label,   h_length_value(len, h_uint8()));
H_ARULE (question, h_sequence(name,type,class,NULL));
H_RULE  (rdata,   h_length_value(h_uint16(), h_uint8()));
H_ARULE (rr,      h_sequence(
                  domain,       // NAME
                  type,         // TYPE
                  class,        // CLASS
                  h_uint32(),   // TTL
                  rdata,        // RDLENGTH+RDATA
                  NULL));
H_AVRULE(message,  h_sequence(header,
                  h_many(question),
                  h_many(rr),
                  h_end_p(),
                  NULL));
```



DNS using H_VRULE

```
bool validate_message(HParseResult *p, void* user_data)
{
      if (TT_SEQUENCE != p->ast->token_type)
        return false;

      dns_header_t *header = H_FIELD(dns_header_t, 0);
      size_t qd = header->question_count;
      size_t an = header->answer_count;
      size_t ns = header->authority_count;
      size_t ar = header->additional_count;

      if (H_FIELD_SEQ(1)->used != qd)
        return false;
      if (an+ns+ar != H_FIELD_SEQ(2)->used)
        return false;

      return true;
}
```



H_RULES in Python

- Use h.action(h_parser, function())
  - e.g.

```
def validate_message():
    ...
message = h.sequence( header, h.many(question), h.many(rr), h.end_p() )
h.action(message, validate_message)
```

6. Building a parser for text-based protocols

Validating HTTP headers

```
GET /secdev/ HTTP/1.1
Host: langsec.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
(.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

Validating HTTP headers

```
GET /secdev/ HTTP/1.1
Host: langsec.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
(.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

Our Python Http Parser

```
def init_parser():
    cr = h.ch('\r')
    lf = h.ch('\n')
    # There is a \r\n at the end of each line. And \r\n\r\n at the end of the header.
    crlf = h.sequence(cr, lf)
    separator = h.ch(':')
    my_whitespace = h.in_("  \n\t\r\t\v\f")
    # URL may have only these characters.
    url_chars = h.many1(h.choice(h.in_("/?%+_=."), h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-"),
                                 h.ch_range('0', '9')))
    httpversion = h.choice(h.token("HTTP/1.1"), h.token("HTTP/2.0"))
    # Request and Response have different line #1.
    method_request = h.sequence(h.many(my_whitespace), h.choice(h.token("GET"), h.token("POST"),
                                 h.token("PUT"), h.token("UPDATE")),h.many1(my_whitespace), url_chars,
                                 h.many1(my_whitespace), httpversion, crlf)
    method_response = h.sequence(httpversion, h.many1(my_whitespace), h.choice(h.token("303 See Other"),
                                 h.token("200 OK"), h.token("401 Unauthorized")), crlf)
    # LHS: RHS format in the header
    lhs = h.many1(h.choice(h.ch_range('a', 'z'), h.ch_range('A', 'Z'), h.ch("-")))
    rhs = h.sequence(h.many(my_whitespace), h.many1(h.choice( h.ch_range('a', 'z'), h.ch_range('A', 'Z'),
                                 h.ch_range('0', '9'), h.in_(" ,\"/=\;\-\*\+.()_:?"))))
    line = h.sequence(lhs, separator, rhs, crlf)
    # Parser can either have request or response, and has many lines.
    parser = h.sequence(h.choice(method_request, method_response), h.many1(line))
    return parser
```

7. Testing with libfuzzer

Wiring hammer with libfuzzer

Libfuzzer does not want a main() method. It wants a method where a fuzzing target is set.

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  DoSomethingInterestingWithMyAPI(Data, Size);
  return 0;  // Non-zero return values are reserved for future use.
}
```

(From tutorial.libfuzzer.info)

Wiring hammer with libfuzzer (contd.)

We now need to define our own method that conforms to libfuzzer's input method.

```
void ParseInput(const uint8_t *Data, size_t Size)
{
    HParser *parser;
    // define hammer parser
    HParseResult *result = h_parse(parser, Data, Size);
    if(result)
    {
        // perform actions on the input
    }
}
```

Wiring hammer with libfuzzer (contd.)

We now need to define our own method that conforms to libfuzzer's input method.

```
void ParseInput(const uint8_t *Data, size_t Size)
{
    HParser *parser;
    // define hammer parser
    HParseResult *result = h_parse(parser, Data, Size);
    if(result)
    {
        // perform actions on the input
    }
}
```

Wiring hammer with libfuzzer (contd.)

We now need to define our own method that conforms to libfuzzer's input method.

```
void ParseInput(const uint8_t *Data, size_t Size)
{
    HParser *parser;
    // define hammer parser
    HParseResult *result = h_parse(parser, Data, Size);
    if(result)
    {
        // perform actions on the input
    }
}
```

Wiring hammer with libfuzzer (contd.)

We now need to define our own method that conforms to libfuzzer's input method.

```
void ParseInput(const uint8_t *Data, size_t Size)
{
    HParser *parser;
    // define hammer parser
    HParseResult *result = h_parse(parser, Data, Size);
    if(result)
    {
        // perform actions on the input
    }
}
```

Compiling with libfuzzer

clang++ -g -fsanitize=address
-fsanitize-coverage=trace-pc-guard **fuzz_me.cc** libFuzzer.a
-lhammer `pkg-config --libs --cflags glib-2.0`

Running the tests

./a.out -detect_leaks=0

```
==================================================================
==30013==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000008 (pc 0x7f53b68698b9 bp 0x7ffc46935fa0 sp 0x7ffc46935f30 T0)
==30013==The signal is caused by a READ memory access.
==30013==Hint: address points to the zero page.
    #0 0x7f53b68698b8 in h_parse__m (/usr/local/lib/libhammer.so+0x1f8b8)
    #1 0x4eb336 in init_parser(unsigned char const*, unsigned long) /home/prashant/fuzzer.cc:7:2
    #2 0x4eb38e in LLVMFuzzerTestOneInput /home/prashant/fuzzer.cc:11:3
    #3 0x4f6f0b in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) /home/prashant/Fuzzer/FuzzerLoop.cpp:463:13
    #4 0x4f6785 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*) /home/prashant/Fuzzer/Fuzz
    #5 0x4f8120 in fuzzer::Fuzzer::ReadAndExecuteSeedCorpora(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, s
g<char, std::char_traits<char>, std::allocator<char> > > > const&) /home/prashant/Fuzzer/FuzzerLoop.cpp:623:5
    #6 0x4f82e5 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
its<char>, std::allocator<char> > > > const&) /home/prashant/Fuzzer/FuzzerLoop.cpp:660:3
    #7 0x4efe9d in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) /home/prashant/Fuzzer/FuzzerDr
    #8 0x4eb440 in main /home/prashant/Fuzzer/FuzzerMain.cpp:20:10
    #9 0x7f53b52c582f in __libc_start_main /build/glibc-bfm8X4/glibc-2.23/csu/../csu/libc-start.c:291
    #10 0x41c5c8 in _start (/home/prashant/a.out+0x41c5c8)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV (/usr/local/lib/libhammer.so+0x1f8b8) in h_parse__m
==30013==ABORTING
MS: 0 ; base unit: 0000000000000000000000000000000000000000
0xa,
\x0a
artifact_prefix='./'; Test unit written to ./crash-adc83b19e793491b1c6ea0fd8b46cd9f32e592fc
```

Thank you!

For more information:
langsec.org

Email: sergey@cs.dartmouth.edu
pa@cs.dartmouth.edu
mcm@cs.dartmouth.edu