Building a Literate Parser and Proxy for DNP3

Sven M. Hallberg, Sergey Bratus, Adam Crain
Outline

• Parsers, security, and the LangSec viewpoint
• Building a safer DNP3 parser from scratch
  “Make the parser code look like the grammar”
  A.k.a. *Parser combinators* (using the Hammer kit from UpstandingHackers.com)
• Case study: a DNP3 filtering proxy
  Validating (testing) our implementation
• Lessons learned / discussion
LangSec

• Many security issues are **language recognition** issues
  exploit = accepting bad input, letting it act on program internals. What to accept? What is expected? What is valid?

• If security seems like an uphill battle...
  Just look at the syntax complexities. (there’s a theory of it: Chomsky hierarchy of grammars)

• Some syntax is poison: (eg.: nested length, fields that must all agree; several sources of truth, ...)

Languages vs recognizers

Languages
- recursively enumerable language
- context-sensitive language
- context-free language
- regular language

Acceptors
- Turing machine
- linear-bounded automaton
- push-down automaton
- finite-state automaton

"Here be dragons"
"The Shire of validation"
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 sucks, but it’s the only way.

• Write the parser to look like the grammar: succinct, *incrementally testable* (from the leaf nodes/primitives up)

• Don’t start processing before you’re done parsing
FULL RECOGNITION

BEFORE PROCESSING
DNP3 issues are not theoretical

• 2013 to 2014 – Over 30 CVEs related to input validation with DNP3 implementations.

• Out of dozens of implementations only a small few were defect-free.

• Low-defect implementations chose a conservative subset
DNP3 Complex?
4.2.2.1 General fragment structure

Request and response fragments have similar, but slightly different, structures (Figure 4-4).

Each fragment begins with an application header that contains message control information. This is true for all fragments regardless of whether they appear in single or multiple fragment messages.
## DNP3 Complex!?!?

<table>
<thead>
<tr>
<th>Group</th>
<th>VAR</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>246</td>
<td>DFR</td>
<td>Attributes - User-assigned ID code/number</td>
</tr>
</tbody>
</table>

### Table 12-4—g3 double-bit binary input static objects

<table>
<thead>
<tr>
<th>Group</th>
<th>Variation</th>
<th>Subset levels</th>
<th>Request (outstation must parse)</th>
<th>Response (master shall parse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>X X X</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>✓ 1 (READ)</td>
<td>00, 01, 0b</td>
<td>129 (RESPONSE) 00, 01</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>X X</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>✓ 1 (READ)</td>
<td>00, 01, 0b</td>
<td>129 (RESPONSE) 00, 01</td>
</tr>
</tbody>
</table>

#### Notes

A.23.1.2.3 Notes

Read requests and responses shall use qualifier code 0x07/0 when an outstation receives this request, it implicitly indicates current time.

This object can be included in a write request. Write request value of 1 for this object. When an outstation receives it, it wants to set the current time in the outstation.
Syntax spills into semantics

// group 50 (times)...
g50v1_time_oblock = dnp3_p_single(G_V(TIME, TIME), time);

Object group 50: time and date

A.23.1.2.3 Notes

Read requests and responses shall use qualifier code 0x07 and a range field value of 1 for this object. When an outstation receives this request, it implicitly indicates that the master wants the outstation to return the current time.

This object can be included in a write request. Write requests shall use qualifier code 0x07 and a range field value of 1 for this object. When an outstation receives this request, it implicitly indicates that the master wants to set the current time in the outstation.
Syntax spills … where?

Object group 51: common time-of-occurrence

An example of an object that depends on a Time and Date Common Time-of-Occurrence object is a binary input change event with relative time, object group 2, variation 3.

The following shows how multiple Time and Date CTO objects may be included in a response when there are not enough bits in a data object to hold the relative time with respect to a single Time and Date CTO object. Each data object’s time is relative to the immediately preceding Time and Date CTO. In the figure, the time in DO_{i+1} is relative to T&D_{i}:

```
| T&D_0 | DO_0 | DO_1 | ●●● | DO_i | T&D_i | DO_{i+1} | DO_{i+2} | ●●● |
```

“should the relative time variants generate an error unless preceded by a CTO object in the same message?”
Language Poison

• Range: (start,stop)
  • If we can't get this right...

• Better: (start,count), ala Modbus & IEC 104

• Would *ideally* like to avoid counts in the first place
  => Context-free!
Implementation Goals / Principles

• Be as grammatical as possible
  • Want to look like CFG, though we can't be

• Avoid code duplication (much abstraction)

• Capture DNP3's "true" syntax
  • Reject at syntax level what others may do later
Parser Combinators: look like grammars

Have primitives

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

Combined to form higher-level structures

```
h_choice, h_many, h_many1, ...
define own combinators
```
Example – Fragment Header Flags

/* --- uns,con,fin,fir --- */
conflags = h_sequence(bit,zro,one,one, NULL);    // CONFIRM
reqflags = h_sequence(zro,zro,one,one, NULL);    // always fin,fir!
unsflags = h_sequence(one,one,ign,ign, NULL);    // unsolicited
rspflags = h_sequence(zro,bit,bit,bit, NULL);
Example - CROB Object

crob = h_sequence(h_bits(4, false),  // op type
    bit,       // queue flag
    bit,       // clear flag
tcc,
h_uint8(),   // count
h_uint32(),  // on-time [ms]
h_uint32(),  // off-time [ms]
status,      // 7 bits
dnp3_p_reserved(1),
NULL));
Example – SELECT Function

```c
pcb = dnp3_p_g12v2_binoutcmd_pcb_oblock;
pcm = dnp3_p_g12v3_binoutcmd_pcm_oblock;
select_pcb = h_sequence(pcb, h_many1(pcm), NULL);
select_oblock = h_choice(select_pcb,
                          dnp3_p_g12v1_binoutcmd_crob_oblock,
                          dnp3_p_anaout_oblock,
                          NULL);
select = h_many(select_oblock);
```

// empty select requests valid?
// is it valid to have many pcb-pcm blocks in the same request?
// ... to mix pcbs and crobs?
// langsec approach warns you of pitfalls!
Practical application: Validating Proxy

Master

Dissector #1

Bi-directional TCP Streams

Dissector #2

Outstation
Pretty printing of AST in log
Validation: familiar tools/techniques

• Unit tests, Unit tests, Unit tests
• Tests based on common DNP3 implementation mistakes
• Dynamic analysis with Valgrind
• Fuzzing: coverage-guided (AFL) and model-based (Aegis)
• No static analysis, but multiple compilers including Clang
No silver bullet, but correct tactic

• Langsec approach doesn’t guarantee success, but provides a **disciplined roadmap** for success

• Traditional testing techniques are just as important, but Langsec gives them more order (when to test what? What to test for? Factor your code so that it’s testable—parser before processing)

• Well-factored parsers will be more maintainable and extensible
Write tests as you write production code.
// mixing CROBs, analog output, and PCBs

check_parse(dnp3_p_app_request,
           "\xC3\x03\x0C\x02\x07\x01\x41\x03\xF4\x01\x00\x00\xD0\x07\x00\x00\x00\xC3\x0C\x03\x00\x05\x21\x04"\x29\x01\x17\x01\x01\x12\x34\x56\x78\x00", 34,
"[3] (fir,fin) SELECT {g12v2 qc=07 (CLOSE PULSE_ON 3x on=500ms off=2000ms)}"
" {g12v3 qc=00 #5..15: 1 0 0 0 0 1 0 0 0 0 1}"
" {g41v1 qc=17 #1:2018915346}");
Unit tests for known poison

// 4-byte max range - start = 0, stop = 0xFFFFFFFF
check_parse(dnp3_p_app_response,
"\x00\x81\x00\x00\x1E\x02\x02\x00\x00\x00\x00\xFF\xFF\xFF\xFF",
15,
"PARAM_ERROR on [0] RESPONSE");

static HParsedToken *act_range(const HParseResult *p, void *user)
{
  // p->ast = (start, stop)
  uint8_t start = H_FIELD_UINT(0);
  uint8_t stop = H_FIELD_UINT(1);

  assert(start <= stop);
  assert(stop - start < SIZE_MAX);
  return H_MAKE_UINT(stop - start + 1);
}
American fuzzy lop (AFL)

- Generic coverage-guided fuzzing
- Program must accept input from stdin
Fuzzing in observe-only mode

DNP3 Fuzzer

Dissector #1

Bi-directional TCP Streams

Dissector #2

Outstation
Challenges - Deep, generic stack traces

parse_choice (env=0x6597f0, state=0x6d6a08)
perform_lowlevel_parse (state=0x6d6a08, par)
parse_length_value (env=0x659830, state=0x6d6a08)
parse_action (env=0x6598b0, state=0x6d6a08)
parse_sequence (env=0x659b30, state=0x6d6a08)
Some Lessons Learned

• DNP3 is obviously well-intentioned :)  
  • Wants syntax to be simple

• Unfortunately ends up doing it wrong :'(  
  • "Uniform" syntax not so uniform

• Could almost be context-free

• Start/stop based index syntax is just plain dangerous.
Discoveries

• Several design/clarification questions
  • correct to ignore FCB on secondary frames?
  • is there a minimum number of bytes in the transport payload?
  • ....

• Spec bugs/issues
  • AN2013-004b: RESPONSE can also include g120v1
  • should status bits be 8 on anaout, but 7 everywhere else?“
  • ....
Future work

• Language subsetting, i.e. constraining grammar via configuration
• Structs -> output (aka un-parsing)
• Open questions WRT to protocol particularities
• Missing features in parser
  • g120 – authentication structures
  • g70 - File transfer
• Proxy that processes multiple sessions
OS protections for well-separated parsers

• Parser is the most dangerous part of the program
  • Most memory corruptions and exploits occur here
• When properly separated, it can be isolated by OS means
• ELFbac: a Linux kernel-based memory isolation for code and data in ELF binary files sections
  • Enforces ACLs between code and data units
    • E.g.: only the parser reads raw input buffers
  • Compatible with Grsecurity/PaX patches
  • Exists for x86 and ARM (public release this January)
• Works for our DNP3 proxy!
Fin - Questions?

https://github.com/pesco/dnp3
https://github.com/sergeybratus/proxy
open source, BSD license