Implementing a vertically hardened DNP3 control stack

Sven M. Hallberg, Sergey Bratus, Adam Crain, Meredith L. Patterson, Maxwell Koo, Sean W. Smith
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)

• Case study: a DNP3 filtering proxy

• Lessons learned / discussion
What is syntax & why check it?

• What we parse for (“syntax”):
  - object boundaries in message,
  - object embeddings in other objects,
  - whether it’s legal for objects to appear in message in a given position

• Validating syntax should:
  - create expected pre-conditions for the rest of the code
  - assure predictable behavior of code on input data
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...
  syntax complexity is likely at fault (the higher up Chomsky’s
  hierarchy of grammars, the harder to parse correctly)

• Some syntax is poison: (eg.: nested length, fields that must all
  agree; several sources of truth, context-dependence)
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**
The Recognizer design pattern

Input

Recognizer for input language

Language grammar Spec

Reject invalid inputs

Processing: only well-typed objects, no raw inputs

Only valid/expected inputs, semantic actions past this line
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, ...

• Overloaded fields
  • recent NTP vulnerabilities

• ... (see our IEEE SecDev 2016 paper)
DNP3 issues are not theoretical

• 2013 to 2014 – Over 30 CVEs related to input validation with DNP3 implementations. ("Robus Master Serial Killer", Sistrunk & Crain, 2014)

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

• Low-defect implementations chose a conservative subset
DNP3 Complex?

Application Layer Fragments

Transport Function Segments

Data Link Layer Frames

Transmission Sequence

A
H = Application Header
T
H = Transport Header
L
H = Link Header
DNP3 Complex??

4.2.2.1 General fragment structure

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

![Diagram showing fragment structure]

Figure 4-4—Fragment structure

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.
A.23.1.2.3 Notes

Read requests and responses shall use qualifier code 0x07 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 wants to set the current time in the outstation.
Vuln #1

Unsolicited Response
Group 1 Variation 0
Sizeless?! 4 byte start/stop

- infinite loop
- missing data
- integer overflow?
- accepts broadcast

From: Adam Crain, Chris Sistrunk “Project Robus, Master Serial Killer”, S4x14
Vuln #2

- UNSOL
- Group 10 Variation 2
- Binary Output Status
- 2 byte start/stop
- 0
- 65535

- infinite loop
- missing data
- unexpected data
- integer overflow?

From: Adam Crain, Chris Sistrunk “Project Robus, Master Serial Killer”, S4x14
Vuln #3

1 byte payload
unconfirmed user data

- transport header only
- unhandled exception

From: Adam Crain, Chris Sistrunk “Project Robus, Master Serial Killer”, S4x14
Vuln #4 (TMW integration)

Unsolicited Response
Control Relay Output Block
1 byte start/stop

- buffer overrun
- not malformed!
- unexpected objects
- accepts broadcast

From: Adam Crain, Chris Sistrunk “Project Robus, Master Serial Killer”, S4x14
Vuln #5 (TMW integration)

- stable infinite loop
- max range - 1 and no data
- accepts broadcast

From: Adam Crain, Chris Sistrunk “Project Robus, Master Serial Killer”, S4x14
Language Poison

• Range: \((\text{start, stop})\)
  • If we can't get this right in 2016...

• Better: \((\text{start, count})\), as in Modbus & IEC 104

• Would \textit{ideally} like to avoid counts in the first place
  \(\Rightarrow\) Context-free is much easier to parse
Syntax spills into semantics

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

Object group 50: **time** and **date**  
Read requests & responses;  
Write requests (to set time)

---

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 $T&D_1$:

```
  T&D_0  DO_0  DO_1  ...  DO_i  T&D_1  DO_{i+1}  DO_{i+2}  ...
```

“should the relative time variants generate an error unless preceded by a CTO object in the same message?”
Implementation Goals / Principles

- Be as grammatical as possible
  - Want the parser to look like a CFG, though we can't be

- Avoid code duplication (much abstraction)

- Capture DNP3's "true" syntax
  - Reject at syntax level what other checkers may (or may not!) do later in the code
Parser combinators: a natural choice

- **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 (1)
Parser combinators at a glance (2)

```
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);
```
Parser Combinators: code looks like the grammar

• Have primitives

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

• Combined to form higher-level structures

```c
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),  // op type
                 NULL));

<table>
<thead>
<tr>
<th>octet transmission order</th>
<th>bit position</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>TCC</td>
<td>CR</td>
</tr>
<tr>
<td>b7</td>
<td></td>
</tr>
<tr>
<td>b31</td>
<td></td>
</tr>
<tr>
<td>b31</td>
<td></td>
</tr>
<tr>
<td>RES</td>
<td>Status code</td>
</tr>
<tr>
<td></td>
<td>Status code &amp; reserved bit</td>
</tr>
</tbody>
</table>
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
UC-8100 Series

Communication-centric RISC computing platform

- ARMv7 Cortex-A8 300/600/1000 MHz processor
- Dual auto-sensing 10/100 Mbps Ethernet ports
- SD socket for storage expansion and OS installation
- Rich programmable LEDs and a programmable button for easy installation and maintenance
- Mini PCIe socket for cellular module
- Debian ARM 7 open platform
- Cybersecurity
Pretty printing of AST in log

```
c - send crob
ms(1449173437972) INFO tcpclient - Beginning task: Command Task
ms(1449173437972) --AL-> tcpclient - CD 03 0C 01 28 01 00 00 00 03 01 64 00 00 00 64 00 00
ms(1449173437972) --AL-> tcpclient - 00 00
ms(1449173437972) --AL-> tcpclient - FIR: 1 FIN: 1 CON: 0 UNS: 0 SEQ: 13 FUNC: SELECT
ms(1449173437972) --AL-> tcpclient - 012,001 Binary Command - CROB, 16-bit count and prefix [1]
ms(1449173437972) --AL-> tcpclient - FIR: 1 FIN: 1 CON: 0 UNS: 0 SEQ: 13 FUNC: RESPONSE IIN: [0x00, 0x00]
ms(1449173437972) --AL-> tcpclient - 012,001 Binary Command - CROB, 16-bit count and prefix [1]
ms(1449173437972) --AL-> tcpclient - CE 04 0C 01 28 01 00 00 00 03 01 64 00 00 00 64 00 00
ms(1449173437972) --AL-> tcpclient - 00 00
ms(1449173437972) --AL-> tcpclient - FIR: 1 FIN: 1 CON: 0 UNS: 0 SEQ: 14 FUNC: OPERATE
ms(1449173437972) --AL-> tcpclient - 012,001 Binary Command - CROB, 16-bit count and prefix [1]
ms(1449173437972) --AL-> tcpclient - FIR: 1 FIN: 1 CON: 0 UNS: 0 SEQ: 14 FUNC: RESPONSE IIN: [0x00, 0x00]
ms(1449173437972) --AL-> tcpclient - 012,001 Binary Command - CROB, 16-bit count and prefix [1]
```

```
2015-12-03 12:10:37,973 INFO [default] <-- [13] (fir,fin) SELECT {g12v1 qc=28 #0:(LATCH_ON 1x on=100ms off=100ms)}
2015-12-03 12:10:37,973 INFO [default] <-- primary frame from outstation 10 to 1: UNCONFIRMED_USER_DATA: C8 CD 01 00 00 00 CD 01 28 01 00 00 00 03 01 64 00 00 00 64 00 00
2015-12-03 12:10:37,973 INFO [default] <-- [13] (fir,fin) RESPONSE {g12v1 qc=28 #0:(LATCH_ON 1x on=100ms off=100ms)}
2015-12-03 12:10:37,974 INFO [default] <-- primary frame from master 1 to 10: UNCONFIRMED_USER_DATA: FE CE 04 0C 01 28 01 00 00 00 03 01 64 00 00 00 64 00 00
2015-12-03 12:10:37,974 INFO [default] <-- [14] (fir,fin) OPERATE {g12v1 qc=28 #0:(LATCH_ON 1x on=100ms off=100ms)}
2015-12-03 12:10:37,974 INFO [default] <-- primary frame from outstation 10 to 1: UNCONFIRMED_USER_DATA: C9 CE 01 00 00 00 CD 01 28 01 00 00 00 03 01 64 00 00 00 64 00 00
2015-12-03 12:10:37,974 INFO [default] <-- [14] (fir,fin) RESPONSE {g12v1 qc=28 #0:(LATCH_ON 1x on=100ms off=100ms)}
```
Validation: tools & techniques

- Unit tests, Unit tests, Unit tests! (easy for parser combinators)
- Tests based on common DNP3 implementation mistakes
- Dynamic analysis with Valgrind
- Fuzzing: coverage-guided (AFL) and model-based (Aegis)
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
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)
    uint32_t start = H_FIELD_UINT(0);
    uint32_t stop = H_FIELD_UINT(1);

    assert(start <= stop);
    assert(stop - start < SIZE_MAX);
    return H_MAKE_UINT(stop - start + 1);
}
Write tests as you write production code

// mixing CROBs, analog output, and PCBs

check_parse(dnp3_p_app_request,
    "\xC3\x03\x0\x02\x07\x01\x41\x03\xF4\x01\x00\x00\xD0\x07\x00\x00\x00"
    "\x0C\x03\x00\x05\x0F\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}"
);
Fuzzing in observe-only mode

DNP3 Fuzzer

Dissector #1

Bi-directional TCP Streams

Dissector #2

Outstation
Testing with American Fuzzy Lop
We survived AFL!

- Generic coverage-guided fuzzing (needs source)
- Program must accept input from stdin
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?“
  • ....
Challenges - Deep, generic stack traces
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
- **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 for ARM ICS
    - [https://grsecurity.net/ics.php](https://grsecurity.net/ics.php)
  - Exists for x86 and ARM
    - [http://elfbac.org/](http://elfbac.org/)
- Works for our DNP3 proxy!
SELinux is not enough

• Process is a set of permitted file operations (e.g., syscalls)

• Permitted operations can be executed in any order, any number of times, at any time

• Nothing about composition within a process: loaded libraries or objects in memory
Process as a **sequence of phases** (code units, sequentially executed)
Process as a set of relationships of code & data units within a process
Sample policy with two libraries

SSL initialization → SSL → libpng → app logic

- SSL keys
- Input buffer
- Output buffer

Access permissions:
- RW: Read-Write
- R: Read
- W: Write
Policy at a glance: isolating raw input buffers
Thank you

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

http://spw17.langsec.org/ (co-located with IEEE S&P)

LangSec talks & papers: http://langsec.org/