Electric Utility Cyber Security

Taken Out of Context:
Language Theoretic Security & Potential Applications for ICS

Darren Highfill, *UtiliSec* darren@utilisec.com
Sergey Bratus, *Dartmouth* sergey@cs.dartmouth.edu
Meredith Patterson, *Upstanding Hackers* clonearmy@gmail.com

What's the Problem?

How do we distinguish between benign and malicious input?

- Trial and error → accumulation of malicious code profiles
  - What do we do about new exploits?
- Trust the source
  - What happens when our source is compromised?

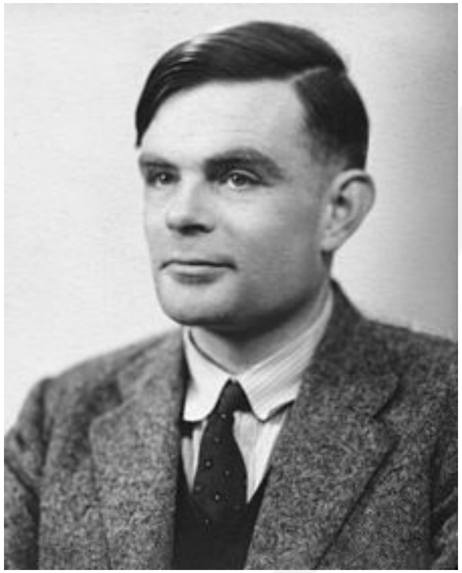
Bottom Line: Given a specific input, can we determine if it is safe to process?

Foundational Concepts

The Halting Problem

> *"Given a description of an arbitrary computer program, decide whether the program finishes running or continues to run forever."*

Alan Turing proved no algorithm can exist which will always correctly decide whether a given arbitrary program and its input will halt



- Any such algorithm can be made to contradict itself, and therefore cannot be correct.

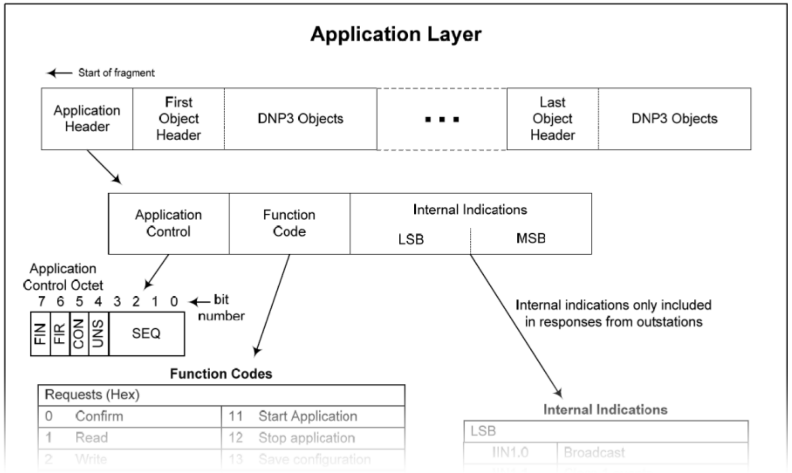
Foundational Concepts

Parsing vs. Processing

Simple: it matches or it doesn't

Harder: it can match multiple different things

Complex: matching depends on other information



Function Codes

Requests (Hex)			
0	Confirm	11	Start Application
1	Read	12	Stop application
2	Write	13	Save configuration

Internal Indications

LSB	
IIN1.0	Broadcast

Do we need to execute any "if" logic?

Separating the parsing from the processing turns out to be an achievable* and valuable step

> "Sufficiently complex input is indistinguishable from executable byte code."

Foundational Concepts

"Shotgun" Parsers

Many parsers do all kinds of input checking

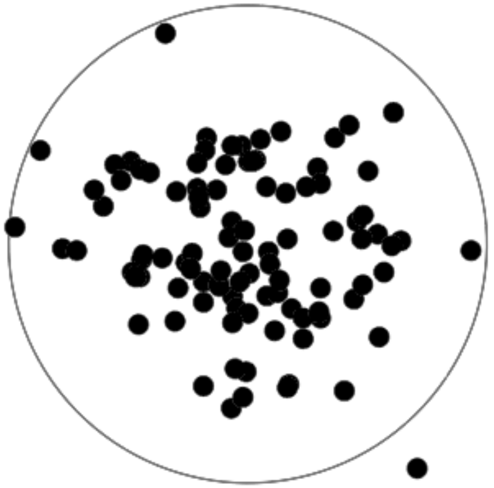
Unfortunately, much of this input checking is scattered all over the program

Have a dense-enough collection of checks, and you are likely to hit most things (although the attacker only has to find one miss!)

Fuzzing

Tends to find the white space between the individual pellet marks

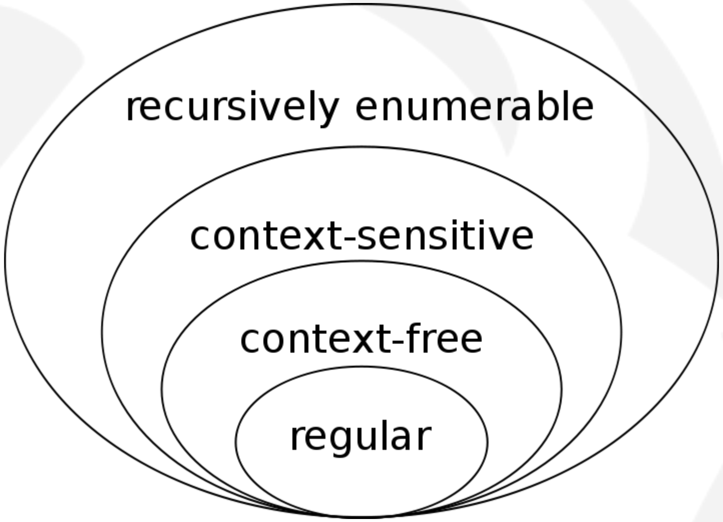
In a way, is the (semi-random) inverse of defining valid input



Foundational Concepts

Language Formalism

Noam Chomsky: containment hierarchy of formal grammars



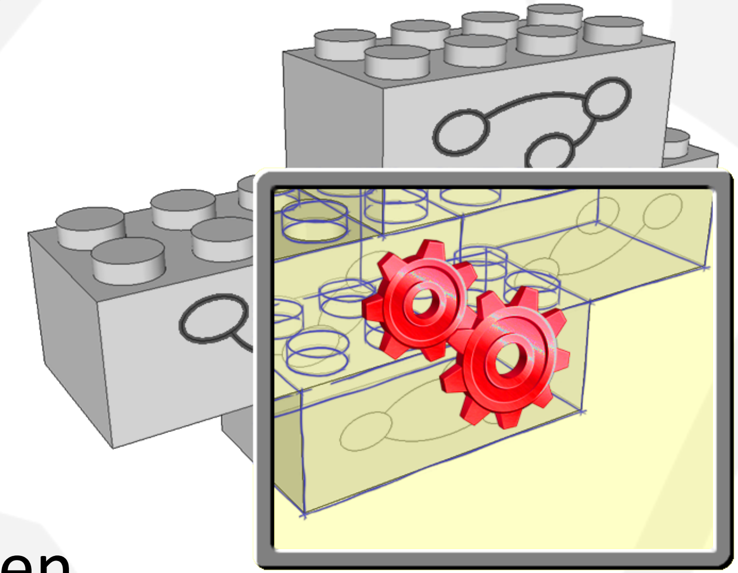
Context Dependency

Do you have to have additional information to determine value or meaning?

Foundational Concepts

Weird Machines

- Hidden functionality unintentionally built into a device
  - Discovered by security researchers
- Distinct from reprogramming
  - Using the intended functionality in unintended ways

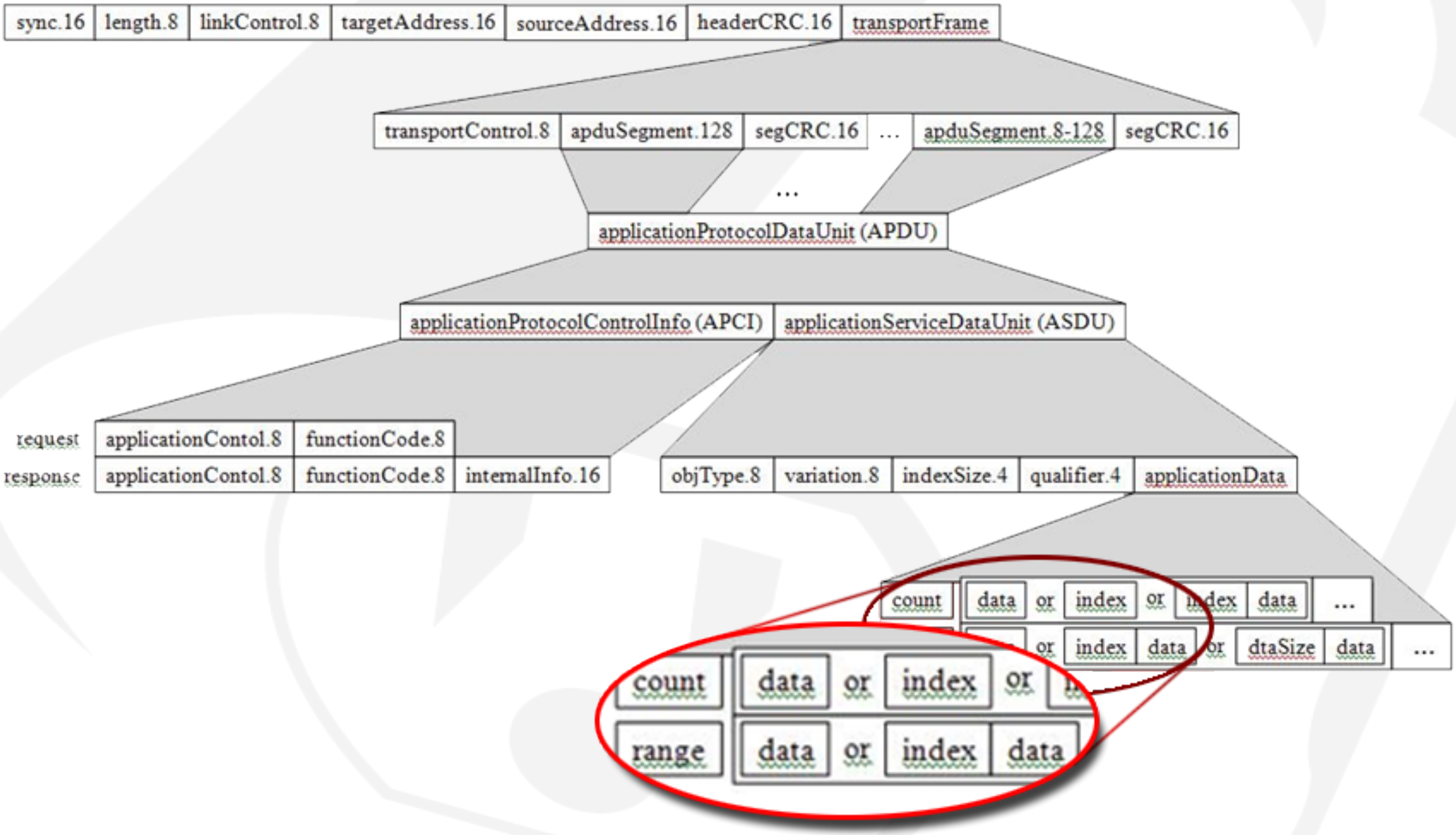


Hypothesis: Machine A has a hidden Machine B inside

- Exploit is proof of existence of Machine B

Applying Concepts to Technology

Parsers all the way down



Debunking a Myth

Hammer parser looks like an input grammar spec

vs. typical C code (difficult to tell what its supposed to parse)

Myth: in order to be fast, code must be unreadable

Example: **Apache**, **Nginx**, HTTP server/proxies

Debunked: **Mongrel**, Ruby HTTP parser

- Based on Ragel state machines (~ LangSec approach)
- Turned out to be much better than Apache at throwing out bad web requests; was put before Apache as proxy – for performance boost
- You save when you throw out bad input early
- And, you are safer from adverse effects

DNP3 Link Layer Parser (simplified)

```
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);
```

Introduction to Hammer

From syntax to semantics: semantic actions

- Wait to start processing until fully parsed & validated
- Clean separation of semantics & syntax

Well-governed feature addition

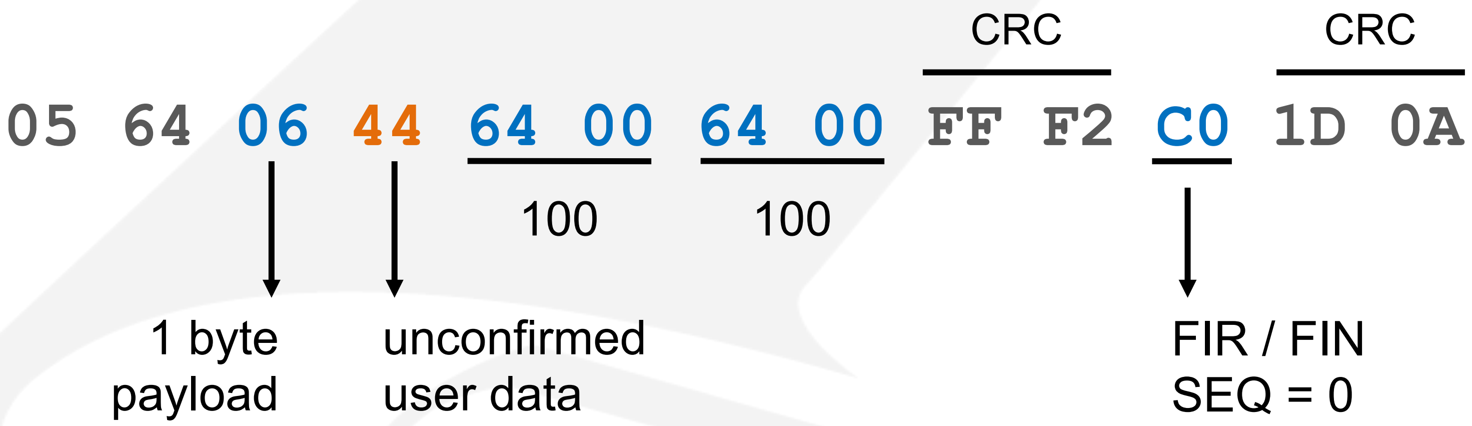
- Where to add new features/functionality?
- Boundary between parsing & processing guides code evolution

Computational power is privilege; don't expose it to attacker early

- Recognition: syntax vs semantics

Example: DNP3 Parser Bug

Sneak Preview (*thank you* to Adam Crain, Chris Sistrunk)



CRC

CRC

05 64 06 44 64 00 64 00 FF F2 C0 1D 0A

100

100

1 byte
payload

unconfirmed
user data

FIR / FIN
SEQ = 0

```
transport_frame =
    h_sequence(transport_ctrl, h_many1(valid_apdu), NULL);
```

Link layer header/transport control octet only

- No APDU (but there should be at least one…)

Unhandled exception

Context-Sensitivity Attacks!

Non-local length-value fields:

The graveyard of empires

- OpenSSH 3.3 pre-auth, 2002
- OpenBSD ICMPv6 remote root, 2007
- DNP3, pretty much everywhere

How much memory do you allocate when you don't know how many CRCs to expect?

- Octet strings
- File control

Object group/object variation are essentially the Interpreter pattern *in your protocol*

Conclusion

Potential Applications

- Open-source library of input parsers
  - Vendors can re-use well-examined code (instead of having to re-write)
- Refinement of fuzz-testing tools
  - Variations based on input-parsing definition

Impact

- Moving toward whitelisting-style input validation
- Proven track record of bug reduction